

© Copyright 2026

Enhao Zhang

AI-assisted video data management for compositional and high-level queries

Enhao Zhang

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:

Magdalena Balazinska, Chair

Ranjay Krishna, Chair

Dan Suciu

Brandon Haynes

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

AI-assisted video data management for compositional and high-level queries

Enhao Zhang

Co-Chairs of the Supervisory Committee:

Professor Magdalena Balazinska

Paul G. Allen School of Computer Science & Engineering, University of Washington

Assistant Professor Ranjay Krishna

Paul G. Allen School of Computer Science & Engineering, University of Washington

Video data is increasingly used in a variety of domains including drone analytics, robotics, urban traffic management, civil engineering, medical education, and biological research. Users of these datasets often need to find complex events in the videos that involve multiple objects, attributes, relationships, and temporal patterns. The rapid advances in artificial intelligence have significantly enhanced video analytics capabilities, including the ability to extract semantic content from videos, making video data queryable in new and interesting ways. As a result, video data management systems have re-emerged as an active area of research. However, existing systems still face three major challenges: users must often possess database expertise to formulate compositional queries that search for complex events in the video data, suitable visual modules that extract application-specific concepts from videos may not exist, and users may want to directly name domain-specific high-level events whose semantics require inferential reasoning and cannot be captured by a fixed composition of predicates. Directly applying large vision-language models (VLMs) to every video can address some of these limitations, but doing so is prohibitively expensive at scale.

In this dissertation, we present three systems that make compositional and high-level video queries easier to specify than in existing video data management systems and more

efficient to execute than direct VLM-based evaluation. The first system, EQUI-VOCAL, synthesizes declarative video queries from a small number of labeled examples, eliminating the need for users to manually express complex spatio-temporal query logic. The second system, VOCAL-UDF, is a self-enhancing video data management system that automatically generates user-defined functions (UDFs) for new application-specific concepts as it processes queries. VOCAL-UDF uses large language models (LLMs) and VLMs to identify missing modules, implement them as UDFs, and select the most effective ones for query execution. The third system, VOCAL-SIFT, targets high-level queries that name events directly without specifying the full visual logic needed to recognize them. It decomposes these events into necessary predicates, instantiates lightweight filters for those predicates, and invokes expensive VLM reasoning only on candidate videos that pass the filters.

Together, these systems show how AI models can be used not only as black-box video query evaluators, but also as collaborators in query synthesis, module generation, semantic decomposition, and query optimization. This dissertation develops an important suite of approaches that make video data management systems more practical, expressive, and scalable.

TABLE OF CONTENTS

	Page
List of Figures	iii
List of Tables	v
Chapter 1: Introduction	1
1.1 EQUI-VOCAL: synthesizing compositional queries from user examples . . .	6
1.2 VOCAL-UDF: self-generating compositional modules with LLMs	8
1.3 VOCAL-SIFT: optimizing high-level queries via predicate filtering	10
1.4 Dissertation organization	13
Chapter 2: Background and related work	14
2.1 Video database management systems (VDBMSs)	14
2.2 Visual representations	15
2.3 Query interfaces	17
2.4 Query execution	19
2.5 Foundation models	23
2.6 Agentic systems	24
2.7 Additional related work	27
Chapter 3: EQUI-VOCAL: synthesizing queries for compositional video events from limited user interactions	29
3.1 EQUI-VOCAL data model	34
3.2 Query synthesis	40
3.3 Query execution	50
3.4 Evaluation	54
3.5 Summary	74

Chapter 4:	VOCAL-UDF: self-enhancing video data management system for compositional events with large language models	76
4.1	Background	79
4.2	A UDF-based data model	81
4.3	VOCAL-UDF approach	84
4.4	Implementation	99
4.5	Evaluation	100
4.6	Limitations	116
4.7	Summary	117
Chapter 5:	VOCAL-SIFT: optimizing high-level video queries via necessary-predicate filtering	118
5.1	VOCAL-SIFT overview	125
5.2	Semantic decomposition	127
5.3	Filter selection	131
5.4	Filter instantiation	137
5.5	Evaluation	139
5.6	Summary	156
Chapter 6:	Interfaces for AI-assisted video querying	158
6.1	EQUI-VOCAL: query by example	158
6.2	VOCAL-UDF: querying with generated UDFs	159
6.3	VOCAL-SIFT: high-level search with predicate filters	164
6.4	Discussion	167
Chapter 7:	Conclusion and future directions	170
	Bibliography	175
	Appendix A: LLM prompts	199

LIST OF FIGURES

Figure Number	Page
3.1 EQUI-VOCAL workflow.	30
3.2 An example of user-provided labeled video segments.	31
3.3 Example frames of multiple, simultaneous car-motorcycle interactions.	34
3.4 Running example of the query synthesis algorithm.	43
3.5 Restrictive vs. relaxed query expansion.	46
3.6 Query execution example.	50
3.7 SQL snippets.	51
3.8 F1 score for queries without duration constraints on simplified tasks.	59
3.9 Impact of data noise, with a labeling budget of 20.	60
3.10 F1 score of EQUI-VOCAL for scene graph queries.	62
3.11 Query synthesis time with varying number of cores.	62
3.12 F1 scores and query synthesis time on the real-world dataset.	65
3.13 User study results.	66
3.14 F1 score for different numbers of initial examples.	69
3.15 F1 score for different numbers of initial examples and query complexities.	69
3.16 F1 score when using active learning vs. random sampling.	71
3.17 F1 score for different values of beam width.	71
3.18 F1 score for different levels of regularization.	72
3.19 F1 score for different numbers of synthesized queries.	72
3.20 Impact of query execution optimizations.	73
4.1 VOCAL-UDF workflow.	77
4.2 VOCAL-UDF system overview.	85
4.3 Program candidates example.	88
4.4 Data labeling by a vision language model.	92
4.5 End-to-end performance of generated queries.	103
4.6 F1 score comparing with direct LLM methods.	104

4.7	Detailed breakdown of efficiency and cost.	106
4.8	F1 score of VOCAL-UDF vs. a single-generation baseline.	112
4.9	F1 score without active learning and dummy UDFs in UDF selection.	115
4.10	Number of samples required to get at least 10 positive examples.	116
5.1	Comparison of approaches for high-level video query processing.	119
5.2	Empirical conditional probabilities and candidate counts for filter selection .	135
5.3	End-to-end filtering performance across datasets.	144
5.4	Validation F_1 distributions for relationship and attribute filters with and with- out object grounding.	150
5.5	Threshold sensitivity across datasets.	151
5.6	User study task completion time.	155
5.7	User study outcomes by condition.	156
5.8	User study questionnaire responses.	156
6.1	EQUI-VOCAL data preparation and query initialization interface.	159
6.2	EQUI-VOCAL labeling interface with history.	160
6.3	EQUI-VOCAL query inspection interface.	161
6.4	VOCAL-UDF query setup interface.	162
6.5	VOCAL-UDF UDF proposal and generation interface.	162
6.6	VOCAL-UDF labeling interface for UDF selection.	163
6.7	VOCAL-UDF result inspection interface.	164
6.8	VOCAL-SIFT query entry and optional clarification interface.	165
6.9	VOCAL-SIFT predicate and filter construction interface.	166
6.10	VOCAL-SIFT result inspection interface.	167
A.1	Prompt of query parsing.	200
A.2	Prompt of UDF proposal.	201
A.3	Prompt of program-based UDF generation.	204
A.4	Prompt of object-aware sampling.	205
A.5	Prompt of deciding the UDF type.	206

LIST OF TABLES

Table Number		Page
3.1	Relational schema representation of data model.	36
3.2	Comparison between compositional video analytics systems.	39
3.3	Example queries written for the trajectories dataset.	53
3.4	Example queries written for the real-world dataset.	54
3.5	Query synthesis time for each method to achieve at least 0.9 F1 scores.	56
3.6	Probability that a system returns at least one query on noisy data.	60
3.7	Number of queries explored and number of predictions per second to achieve at least 0.9 F1 scores.	61
3.8	Quivr’s success rate on the real-world dataset.	64
3.9	Queries used in the user study.	64
3.10	Labeling error rates of participants in the user study.	67
3.11	Median F1 scores of EQUI-VOCAL and ML methods.	74
4.1	Relational views for query execution with UDFs.	79
4.2	LLM cost estimation for executing one query.	108
4.3	Natural language to DSL correctness.	108
4.4	Proposing UDFs.	109
4.5	Program-based UDFs performance.	110
4.6	Program-based UDF types.	110
4.7	Distilled-model UDFs performance.	111
4.8	VLM labeling quality.	113
4.9	UDF selection performance and selected UDF distribution.	113
4.10	Number of correctly selected UDF types.	114
5.1	Notation used in filter selection.	132
5.2	VOCAL-SIFT evaluation datasets.	140
5.3	True positive rate, true negative rate, and video retention rate across datasets and methods.	144

5.4	Estimated dollar cost for processing all queries in each dataset.	146
5.5	Semantic decomposition variants.	147
5.6	Filter selection policies.	148
5.7	Filter instantiation variants.	150

LIST OF ALGORITHMS

1	Query synthesis algorithm.	41
2	UDF selection using active learning.	96
3	VOCAL-SIFT overview.	125

ACKNOWLEDGMENTS

First, I would like to thank my advisors, Magdalena Balazinska and Ranjay Krishna, for their guidance and support throughout my Ph.D. journey. I began as a new student with little research experience, and I am deeply grateful for their thoughtful mentorship in helping me grow into a more confident and independent researcher. Their wisdom, rigor, and optimism have shaped me in subtle but lasting ways. I feel very fortunate to have had the opportunity to learn from them over the past several years. I would like to thank Dan Suciu for introducing me to many theoretical perspectives on database research and for his encouragement and support at many important moments along the way. I am also grateful to Brandon Haynes for his long-standing collaboration on the video projects.

I also want to thank my undergraduate research advisors, Nikola Banovic and Michael Cafarella. Nikola, you first introduced me to the world of research, and for that I will always be deeply grateful. Mike, you introduced me to the field of video databases, which had a profound influence on my decision to pursue a Ph.D. and to continue working in this area.

I will always cherish the days I spent with the UW database group in CSE 291. Special thanks to Dong He and Junran Yang for striving and growing together, for all the laughter, and for all the shared complaints. I am also grateful to Maureen Daum, Nicole Sullivan, and Manasi Ganti for making me feel less alone on the path of video research. I would also like to thank Leilani Battle, Moe Kayali, Ameya Patil, Jiacheng Wu, Kyle Deeds, Guorui Xiao, Yihong Zhang, David Alexander, Michael Theologitis, Anton Lykov, Ryan Mass, Remy Wang, Jieyu Zhang, and Dongfang Zhao for their friendship, conversations, and support.

I am fortunate to have had two incredible internship experiences. I would like to thank Dmitry Lychagin and Gaurav Yadav at Snowflake for guiding my first internship and helping

me gain a deeper understanding of industrial database systems. I also feel very fortunate to have met Erkang Zhu at Microsoft Research. Thank you for introducing me to the world of agentic systems; I have learned so much from every conversation with you.

The friends I met at UW brought so much joy to my Ph.D. life beyond research. I especially want to thank my roommate of five years, Rock Pang. Although my first reaction to each of your party invitations was always to say no, you helped me meet many wonderful new friends. Thank you to Xiangfeng Zhu, Will Wang, Inna Lin, Ruotong Wang, Preston Jiang, Xia Su, Tuochao Chen, Xiaochuang Han, Liwei Jiang, and Weijia Shi for their friendship and support.

My friends from every stage of my life have been a tremendous source of support. Jiaxin Ye, thank you for being my friend since undergrad, for your sofa in the Bay Area, and for all the late-night battles we somehow convinced ourselves were “just one more game”. Thank you to Jiongli Zhu, Gibbs Geng, Renzhong Lu, Chunhao Shen, Yuming Zeng, Jiayi Zhu, Ziyi Zhou, Xiaoyi Wang, Kaifeng Xu, Xueyang Zhang, and Jiayan Chen.

Finally, I want to thank my parents for their unconditional love and support. I could not have done this without you.

I have always considered myself a fortunate person, and I am grateful for all the kindness and love I have encountered throughout my life.

Chapter 1

INTRODUCTION

Video has become one of the most common ways that people and organizations record the world. Cameras on phones, cars, drones, traffic lights, in hospitals, laboratories, and industrial equipment continuously capture what happens, where it happens, and how it unfolds over time. For instance, autonomous vehicles record complex traffic interactions, camera collars track the behavioral patterns of wild deer, and endoscopic cameras capture the intricate details of minimally invasive surgeries. Hidden inside these recordings are answers to questions that are difficult to obtain from text or tables alone: Which road interactions are dangerous? Which animal behaviors occur near a changing habitat? Which steps in a surgical procedure are performed correctly? Turning videos into answers to such questions requires systems that can search not only for objects in video frames, but also for activities, interactions, attributes (e.g., a *brown* bear), and temporal patterns.

Modern artificial intelligence has greatly expanded what video analytics systems can recognize. Computer vision models can detect objects and classify actions, while large language models (LLMs) provide advanced reasoning over natural language (NL) instructions. By bridging these two domains, vision-language models (VLMs) can jointly process visual and textual input in a variety of ways, from describing the content of a video frame to answering complex NL questions about events that evolve over time. These advances have enabled applications including drone analytics [207, 210], citywide traffic analytics [11, 76], civil engineering [73, 20], medical education [66], biological research [183, 54], and many others [158, 71, 185, 192, 140]. Across these domains, analysts often want to ask questions in NL rather than translate their intent into fixed detection tasks: a traffic analyst may ask when a risky interaction begins, a medical educator may ask for a procedural summary, and a biolo-

gist may ask what behavior is occurring in a clip. Recent models bring video analytics closer to this mode of interaction by supporting flexible video understanding tasks such as detailed video captioning [139], temporal localization [171], and video question answering [129, 16].

Yet recognizing or answering questions about individual videos is only one part of what video analytics systems must support. In practice, analysts querying a video corpus rarely want to apply one predefined visual predicate or inspect one clip at a time. Instead, they search for events that combine many pieces of visual evidence: objects, attributes, pairwise relationships, temporal order, duration, and sometimes broader domain knowledge. Some of these questions are *compositional*: their meaning can be described by composing atomic predicates such as “motorcycle”, “silver”, “near”, and “colliding”. Others are *high-level*: they name events directly and leave the visual logic implicit, such as “car accident”, “alley-oop in basketball”, or “Biellmann spin in figure skating”.

Below, we present an example use case that illustrates both kinds of queries:

Example 1.1 (Traffic Surveillance) *Consider a traffic analytics application: A traffic engineer may want to understand road hazards involving car and motorcycle interactions. They may first search for video segments where a motorcycle is swerving near a car and then colliding with it. This query has explicit visual structure: it involves two objects, a spatial relationship, and an ordered sequence of events. After reviewing these matching segments, the engineer may notice that several incidents involve a silver Subaru and refine the query to focus on that specific car model. This refinement requires additional visual concepts, such as car color and make. Once the engineer shifts from studying this particular motorcycle-collision pattern to assessing the broader risk associated with that vehicle, they may want to find any car accident involving a silver Subaru. This last query is higher-level: accidents can manifest through collisions, sudden stops, swerving, pedestrians entering the road, fallen objects, or many other visual situations.*

In this dissertation, we address the question of how best to support queries over video data. We consider both queries that are compositional—asking for specific objects and their

interactions in space and time—and queries that are high-level—naming events or actions directly while leaving the visual logic needed to recognize them implicit. We take a data management perspective on the problem, seeking to develop techniques and systems that make it easy for users to express their queries and then efficiently answer those queries.

Supporting these video queries raises several challenges. First, existing models struggle to identify complex events out-of-the-box. Traditional end-to-end models are typically limited to specific, fixed vocabularies of actions [30, 74], and novel event detection requires significant effort and annotated data to train new models from scratch [201, 241]. Specialized pretrained models excel at isolated tasks like image classification, object detection or video action classification [151, 78, 130], but they lack the native reasoning capabilities required to handle domain-specific compositions of objects, attributes, relationships, and time [95]. Recently, large foundation models such as VLMs and LLMs have shown remarkable promise for open-ended visual question answering [166, 129, 126]. However, they still struggle with complex visual reasoning [137], especially when temporal reasoning is required [228]. While new models continue to improve their spatial [35] and compositional [88] reasoning capabilities, their performance remains limited [206].

A second challenge is evaluation cost. While large foundation models offer the flexibility needed to handle open-vocabulary events better than traditional specialized models, this enhanced capability comes with a steep computational overhead. Deploying these massive models at scale is prohibitively expensive [163], and their inference speeds remain significantly slower than those of traditional computer vision models [202]. For example, current VLMs typically achieve a throughput of around 100 tokens per second [16, 3]. Because encoding a single video frame can consume several hundred to over a thousand tokens depending on the resolution [155], processing continuous video corpora that generate dozens of frames every second is a massive bottleneck. Consequently, the direct evaluation of large foundation models over video is fundamentally intractable.

To address both these reasoning limitations and computational bottlenecks, researchers use video database management systems (VDBMSs), which are specialized systems designed

to efficiently store, manage, and query large-scale video data. Rather than relying on a single monolithic model to process an entire query, most VDBMSs execute complex compositional queries by decomposing them into smaller subtasks [67, 216, 23, 143, 217]. Specifically, they represent queries as logical predicates over visual atoms (e.g., objects, their attributes, and their relationships) extracted from video frames. While most user queries are novel and unseen, they typically consist of these known, previously seen atoms. Existing computer vision models already handle many of these common atoms, and VDBMSs generally include built-in modules for these tasks [75, 143]. Furthermore, these systems often support extensibility through user-defined functions (UDFs), enabling customization for unsupported scenarios [216, 97]. Recent systems use this decomposition strategy to express declarative queries as compositions of extracted information [67, 227, 33, 131, 43, 217, 143, 23].

Despite significant progress, existing VDBMSs still face three key challenges:

First, articulating declarative queries for compositional events requires considerable database expertise (**C1**). While many VDBMSs support query formulation through SQL, SQL extensions, or imperative programs [99, 216], query complexity grows rapidly as events involve more objects, relationships, and temporal constraints. Even experts may struggle to express real-world events declaratively because there can be many plausible query formulations, and the best formulation depends on the data. Other systems ask users to sketch their target events [43, 31, 214], but sketches have the same brittleness: real event instances may deviate from the user-provided example.

Second, VDBMSs assume the existence of suitable modules for the subtasks that appear in a complex query (**C2**). This assumption is often false. A user may need to identify fine-grained categories, domain-specific concepts, or subjective visual properties for which no off-the-shelf model exists. In Example 1.1, for instance, the system may know how to detect cars but lack UDFs for “silver” or “Subaru”. Filling these gaps manually requires the user or system developer to find an appropriate model, adapt it to the target domain, or build a new classifier. In many cases, each missing UDF may require its own curated dataset and extensive training process to achieve satisfactory performance [13].

Third, users often issue high-level queries that name events directly while leaving the visual logic implicit (**C3**). Predicate-based representations are expressive for queries with detailed object, attribute, relationship, and temporal descriptions, but high-level queries ask the system to infer which visual conditions are relevant. A “car accident”, for example, may be indicated by collisions, sudden stops, swerving, pedestrians entering the road, fallen objects, or other visual evidence. Identifying such events therefore requires case-by-case inferential reasoning, for which VLMs are better suited than a single fixed predicate composition. However, invoking a large model on every video in the dataset is prohibitively expensive. For instance, directly applying GPT-4o to only seven hours of video can cost hundreds to thousands of US dollars for a single query [236].

In this dissertation, we present three systems that make compositional and high-level video queries easier to specify than in existing VDBMSs and more efficient to execute than direct VLM-based evaluation. Together, they show how AI models can assist a VDBMS not only by answering queries directly, but also by synthesizing queries, generating missing modules, decomposing high-level semantics, and optimizing expensive model calls:

- **EQUI-VOCAL** automatically synthesizes declarative queries for compositional events from a handful of labeled video segments, reducing the expertise needed to formulate queries (**C1**). **EQUI-VOCAL** first appeared in PVLDB Volume 16, Issue 11 [235], and we presented and demonstrated it [234] at the corresponding VLDB 2023 conference.
- **VOCAL-UDF** dynamically identifies missing modules and generates them as UDFs, enabling NL compositional queries even when required predicates are not predefined (**C2**). **VOCAL-UDF** appeared in SIGMOD 2025 [236].
- **VOCAL-SIFT** processes high-level queries by decomposing them into necessary predicates, training or reusing lightweight predicate filters, and invoking expensive VLM reasoning only on candidate videos that pass the filters (**C3**).

The three systems are part of our larger **VOCAL** (**V**ideo **O**rganization and Interac-

tive **C**ompositional **A**na**L**ytics) suite of systems [52]. In the remainder of this chapter, we elaborate on each contribution in more detail.

1.1 ***EQUI-VOCAL: synthesizing compositional queries from user examples***

The outermost layer of a VDBMS is the query interface that connects users to the rest of the system. Many VDBMSs enable query formulation by extending SQL [99, 216, 21, 136] or using imperative programming languages [67, 227]. For non-experts, this approach can be challenging and time-consuming, especially as target events become more intricate. Compositional queries add further complexity by requiring users to track multiple objects across frames, specify spatio-temporal relationships between objects, and manage temporal constraints.

The growing capacities of LLMs present a potential solution to translating NL queries into SQL queries via in-context learning without fine-tuning [160, 203]. While an LLM-powered interface allows users to describe their target event in NL, inherent ambiguity can frequently lead to inaccurate translations [118, 65]. Systems that ask users to sketch their events [43, 31, 214] face the same challenge: an event of interest may deviate from the exact user-provided sketch.

Although computer vision models are widely used for image and video classification [59, 62] and object detection tasks [78, 169], pretrained models rarely exist for arbitrary compositional events that involve multiple objects and their spatio-temporal interactions. While VLMs show strong visual reasoning capability, their performance on compositional queries remains limited [137, 206], and invoking them on large-scale video datasets is prohibitively expensive [236, 163, 3, 202].

To address these issues, we introduce EQUI-VOCAL, a system that automatically synthesizes declarative queries on behalf of users from a small number of labeled video segments. EQUI-VOCAL stands for **E**volving **Q**ueries for **U**sers **I**teratively. To use the system, the user provides a few positive and negative examples (i.e., video snippets) of the event they are looking for, along with a small number of additional labels on examples selected by the

system. EQUI-VOCAL then synthesizes a declarative query and executes it on the remaining video corpus to retrieve additional instances of the desired event.

EQUI-VOCAL provides an intuitive query-by-example interface, enabling users to formulate compositional queries directly from video data without writing precise declarative specifications or sketches. It follows the modular view introduced above: rather than training a new classifier for every target event, the system describes an event using visual atoms that existing computer vision models can often extract, such as objects (e.g., “car”, “backpack”), attributes (e.g., “red”, “leather”), and spatial or semantic relationships (e.g., “left of”, “holding”). In the computer vision community, such compositions of visual scenes are commonly represented as scene graphs [96, 113, 91], which have cognitive grounding in human perception [230, 117, 24]. A scene graph encodes objects as nodes, relationships as edges, and attributes as properties of objects. EQUI-VOCAL extends this idea to videos with spatio-temporal scene graphs, which represent a video as a sequence of per-frame graphs populated by user-provided functions such as object detectors, relationship classifiers, and attribute classifiers.

Key to EQUI-VOCAL’s contribution is to encapsulate these spatio-temporal scene graphs in a relational data model and use them to define a restricted declarative query language for compositional video events. In this domain-specific language (DSL), a query is a temporally ordered sequence of graph patterns with predicates over objects, relationships, and attributes, along with duration and window constraints. EQUI-VOCAL synthesizes queries by searching this DSL for a composition of extracted scene graph atoms that matches the user’s labeled examples. This design tackles the central tension in query synthesis: the language must be expressive enough to capture complex events, but small and structured enough that the system can search it efficiently from limited and potentially noisy labels.

By leveraging scene graphs, EQUI-VOCAL introduces a new query synthesis approach that finds user events with far fewer labeled examples than would be required to train a specialized machine learning model directly. To reduce computational effort, EQUI-VOCAL restricts the query search space using scene graphs, employs beam search to efficiently navigate

the query space, and generates optimized queries to avoid expensive database operations such as recursive joins. Unlike prior query-by-example approaches that may enumerate exhaustive SQL queries [63, 162, 124, 204, 195, 198], EQUI-VOCAL strategically limits exploration, enabling practical scalability.

EQUI-VOCAL reduces user effort through active learning and robust noise handling. Instead of requiring a user to provide all examples up front, EQUI-VOCAL uses active learning to iteratively request labels for carefully selected additional examples that reduce uncertainty in query synthesis. Additionally, recognizing the inherent noise in user labels, machine learning model outputs, and ambiguous real-world events, EQUI-VOCAL searches for queries that best match potentially noisy data and input. It retains imperfect query candidates at every iteration and uses regularization to prevent overfitting to noise or limited user input.

We evaluate EQUI-VOCAL on both synthetic and real datasets [224, 21] and show that it outperforms two baselines [195, 143] in F1 score, synthesis time, and robustness to data noise. EQUI-VOCAL can also synthesize complex queries that the baselines do not support. A user study further shows the performance of real users using EQUI-VOCAL.

Overall, EQUI-VOCAL is an important step toward making VDBMSs accessible to experts and non-experts alike by easing the task of expressing queries over videos. EQUI-VOCAL and its demonstration appeared in PVLDB Volume 16 [235, 234], and the code is available at <https://github.com/uwdb/equi-vocal> and <https://github.com/uwdb/equi-vocal-demo>.

1.2 VOCAL-UDF: self-generating compositional modules with LLMs

While EQUI-VOCAL can synthesize video queries from examples, it still requires users to provide UDFs for extracting semantic information from videos, which represents a burden for users. In general, VDBMSs make a critical assumption that suitable modules already exist for the subtasks needed to answer a complex query. In Example 1.1, if the user wants to refine a query to focus on a “silver Subaru”, they may need UDFs that classify “silver

color” and “Subaru make”. It is impractical for either a user or a system designer to provide all possible UDFs needed for arbitrary future queries.

VLMs embed extensive knowledge of visual concepts acquired during pretraining and can be used to extract application-specific concepts. However, repeatedly invoking a VLM to classify every relevant visual predicate, such as objects, attributes, relationships, or events, across a large video dataset is expensive and slow. A natural way to reduce cost is to invoke the VLM on only a subset of frames or clips, for example by sampling one frame per second rather than evaluating all frames. However, this strategy is non-trivial because low sampling rates can miss transient visual evidence and greatly degrade query performance, especially for compositional events with temporal constraints.

To address these challenges, we present VOCAL-UDF, a *self-enhancing* VDBMS that enables users to query compositional events even when required modules are unavailable. To use VOCAL-UDF, a user provides a video dataset and an NL query. VOCAL-UDF first translates the NL query into the system’s DSL, where the query is represented as a composition of predicates over objects, attributes, relationships, and time. During this translation, VOCAL-UDF compares the predicates required by the query with the UDFs already available in the system. When a required predicate is missing, VOCAL-UDF uses LLMs and VLMs to generate candidate UDFs, select the most effective ones, and add them to the system’s library of UDFs. Once the missing modules are available, VOCAL-UDF re-compiles the NL query and executes it over the video dataset.

To transform a multitude of missing semantic concepts into executable modules, VOCAL-UDF supports two types of UDFs: *program-based UDFs* and *distilled-model UDFs*. Program-based UDFs are imperative Python functions that operate on existing concepts, bounding box coordinates, or video pixels. This class of UDFs can classify many relationships and attributes with high quality, such as “left of” and “silver” [235, 112], but struggles with tasks that require richer visual understanding. Distilled-model UDFs are lightweight machine learning models trained on the fly to classify such nuanced concepts without requiring user labeling [196]. VOCAL-UDF leverages LLMs and VLMs for UDF generation: program-based

UDFs harness an LLM’s programming capability, whereas distilled-model UDFs rely on the VLM’s ability to annotate and convert visual concepts into ML models.

Handling ambiguity in user intent and errors in LLM generations is critical, especially for specialized or subjective concepts that cannot be reliably resolved without direct user input. Different users may interpret the same concept differently, and whether a program-based UDF or a distilled-model UDF performs better depends on the specific concept. For particularly challenging semantic concepts, *no* generated UDF may perform well, and using a poor UDF can harm overall query performance. VOCAL-UDF addresses these challenges through a UDF selection stage that actively solicits user feedback. VOCAL-UDF generates multiple candidate UDFs with different semantic interpretations, implementations, and properties. It then utilizes active learning at query time to strategically request binary labels from users to select the best candidate UDF.

We evaluate VOCAL-UDF on three video datasets from different domains [224, 64, 92] and show that it significantly improves query performance in F1 score compared with two baseline systems and two direct LLM methods. We further conduct a thorough analysis of VOCAL-UDF to understand its efficiency in terms of execution time and monetary cost.

Overall, VOCAL-UDF’s self-enhancing capability is an important step toward making VDBMSs practical to deploy and use across applications. VOCAL-UDF appeared in SIGMOD 2025 [236], and the code is available at <https://github.com/uwdb/vocal-udf>.

1.3 VOCAL-SIFT: optimizing high-level queries via predicate filtering

EQUI-VOCAL and VOCAL-UDF help users express and execute compositional queries: EQUI-VOCAL synthesizes declarative queries from examples, and VOCAL-UDF generates missing predicates needed to execute NL queries. Both systems ultimately rely on the premise that the target event can be represented as a declarative composition of atomic predicates. However, many real-world queries are high-level. In Example 1.1, determining whether a video contains a “car accident involving a silver Subaru” requires reasoning over diverse visual situations. A single fixed composition of predicates is unlikely to cover all such cases

without either missing valid cases or admitting many irrelevant ones.

Querying such events requires inferential reasoning case by case. VLMs are well suited to this role because they can combine many partial cues and domain assumptions when deciding whether a high-level event occurred. Prompting techniques [109, 147] can further improve such reasoning. However, applying large models to every video in a corpus is time-consuming and computationally expensive. For instance, our evaluations in VOCAL-UDF show that directly invoking GPT-4o over 7 hours of video costs \$207 for a single query. Using an unoptimized modular execution procedure, which parses the query into a logical plan and repeatedly invokes the VLM to identify missing concepts, increases the cost to over \$1,500. VOCAL-UDF reduces such costs when a query can be translated into a detailed declarative form, but this optimization does not directly apply when the query names a high-level event and leaves the recognition logic implicit.

This creates a different optimization problem from the one addressed by VOCAL-UDF. For high-level queries, the system must reduce the number of videos sent to the VLM without forcing the user to provide a complete compositional definition of the event. Because filtering happens before the VLM makes the final decision, the filter must preserve high recall while still removing enough unrelated videos to reduce cost. The key question is therefore how to infer filters that are selective enough to prune the corpus while retaining the videos that still require VLM reasoning.

To address this problem, we present VOCAL-SIFT, a video query processing system for efficient high-level NL querying over large video datasets. VOCAL-SIFT follows a “decompose, filter, then evaluate” strategy. It does not attempt to replace the final VLM judgment with a complete declarative definition. Instead, it identifies *necessary predicates*: visually detectable conditions that should hold whenever the high-level event occurs, even though they may not be sufficient to prove that the event occurred. For example, “dirt jumping in mountain biking” may require a person, a bike, a mound, and relationships such as the person riding the bike or the bike being above the mound. These predicates can safely reject many unrelated videos before invoking the expensive VLM on the remaining candidates.

VOCAL-SIFT must answer three hard questions to make this strategy work. First, it must derive necessary predicates from a high-level NL query. This is non-trivial because high-level queries are under-specified and admit many valid visual instantiations. The system must infer predicates that are implied by the event and visually detectable without over-specifying the event or introducing spurious conditions. This difficulty is compounded by the variability of LLM outputs: naively prompting an LLM may yield inconsistent decompositions across runs, omit critical predicates, or introduce spurious ones. VOCAL-SIFT addresses these issues with a semantic decomposition stage that performs multiple LLM decompositions independently. Rather than constraining the LLM to a predefined vocabulary, the system allows it to generate free-form predicates first, maps these concepts to the system’s existing library of supported predicates, and finally reconciles the results into a consistent set of necessary predicates.

Second, VOCAL-SIFT must decide which predicates are worth turning into filters. Predicates from the same query are often correlated: if a filter has already found clips with a cyclist and a ramp, another filter for outdoor sports may add little pruning benefit. At the same time, every new filter has a construction cost and may introduce additional false negatives. VOCAL-SIFT therefore estimates whether a missing predicate adds enough pruning benefit beyond existing filters to justify the one-time cost of filter construction.

Third, VOCAL-SIFT must instantiate selected predicates as filters that are cheap enough to run broadly and precise enough to be useful. This is especially important for attributes and relationships, whose meanings often depend on the objects involved. A generic “holding” predicate, for example, can refer to many interactions, while “player holding basketball” is a more precise condition for both annotation and inference. VOCAL-SIFT realizes selected predicates as lightweight filters using model distillation and grounds relationship and attribute filters in their participating objects during annotation and inference.

We evaluate VOCAL-SIFT on three datasets and show that VOCAL-SIFT preserves high recall during filtering while passing only a small fraction of videos to the oracle VLM. This reduces the estimated dollar cost by up to $13.6\times$ compared with direct VLM evaluation. A

user study further shows that VOCAL-SIFT’s automatic decomposition reduces task completion time compared with human clarification without a statistically significant degradation in filtering quality or task success.

In summary, EQUI-VOCAL, VOCAL-UDF, and VOCAL-SIFT address complementary challenges in VDBMSs. Together, they introduce AI-driven techniques that simplify the task of expressing queries over video data, accelerate query evaluation, and reduce the cost of that evaluation. These contributions help make video data management and thus video applications more practical.

1.4 Dissertation organization

The rest of the dissertation is organized as follows. Chapter 2 presents background information and related work. Following this, Chapter 3 introduces EQUI-VOCAL, Chapter 4 introduces VOCAL-UDF, and Chapter 5 introduces VOCAL-SIFT. Chapter 6 discusses the user interfaces for the three systems. Lastly, we conclude this dissertation in Chapter 7 and discuss potential future research directions.

Chapter 2

BACKGROUND AND RELATED WORK

This chapter situates AI-assisted video data management at the intersection of database systems and modern artificial intelligence. We begin with the foundations of video database management systems (VDBMSs), which turn videos into queryable visual facts. We then walk through the main components of a VDBMS in turn: the visual representations it exposes to the query processor, the interfaces through which users formulate queries against those representations, and the database techniques used to execute those queries efficiently at scale. Building on these foundations, we examine how foundation models and agentic systems extend the reach of VDBMSs, broadening what can be recognized, decomposed, and reasoned about. The chapter closes with related work that is specific to the systems presented later in this dissertation.

2.1 Video database management systems (VDBMSs)

Relational database management systems are the most widely studied and used database systems for efficiently storing, processing, and querying structured data in the form of relational tables. Driven by the increasing availability of video data and advances in artificial intelligence, VDBMSs have re-emerged as an active research area within the database community. To support unstructured video data, a VDBMS must rethink the traditional system architecture of a database management system. Consequently, researchers have proposed a wide array of novel systems, algorithms, and techniques spanning different components of the database engine, including the data model, query language, query interfaces, query optimization and execution, and data storage. Before detailing these core components, we

briefly trace the historical evolution of VDBMSs.

Early VDBMSs. The management of video data has been studied for decades. Prior to the widespread adoption of deep learning, however, the capabilities of VDBMSs were significantly restricted. Early architectures primarily allowed users to query over either metadata or manual annotations [50, 189, 57, 153] or content-retrieval based on low-level, hand-engineered features like shape, color, motion, and texture [31, 57].

Modern VDBMSs. Advances in AI have steadily broadened the scope of queries that VDBMSs can support. Many of these systems propose query languages and execution algorithms for simple queries over video data including selection filtering [136, 100, 86, 12], aggregations and limits [99], object tracking [21], object interactions [33], and trajectory interactions [143, 214]. While some systems have started to support more complex queries, their approaches remain limited. Zeus [49] can only detect actions for which a pretrained model exists. Caesar [131], Rekall [67], STAR [43], VQPy [227], VidCEP [217], Chen et. al. [44], and Koudas et. al. [111] support compositional queries, but they require precise user specifications and mandate exhaustive object detection across all frames before they can execute a complex query.

Sections 2.2 to 2.4 examine the main components of a VDBMS in turn: the visual representations it exposes to the query processor, the interfaces through which users formulate queries against them, and the database techniques used to execute those queries efficiently.

2.2 Visual representations

A visual representation is the intermediate form through which a VDBMS makes video content available to query processing. Raw video data consists of pixels that are high-dimensional, temporally redundant, and difficult to query directly. VDBMSs therefore rely on representations to bridge low-level visual signals and high-level query semantics. The choice of representation determines what queries can be expressed, what indexes or optimizations are possible, and how much visual computation must be performed at query time.

Early multimedia and video database systems often relied on hand-engineered visual feature representations: color histograms, texture descriptors, shapes, motion features, regions, trajectories, and other summaries computed from pixels. VideoQ [31] supported content-based video search using sketches and indexed videos on motion, shape, color, and texture. BilVideo [57] supported spatio-temporal queries by similarly combining a feature database that stored extracted color, shape, texture, and trajectory features with a knowledge base that stored segment and object metadata. These representations are model-independent, relatively cheap, and were usable before modern learned detectors became available. Their weakness is the semantic gap: low-level appearance and motion features do not directly identify concepts such as object categories, interactions, or complex events. As such, they could support only limited types of queries, either through content-based retrieval over low-level cues such as texture, color, shape, and motion, or over manually provided metadata.

To narrow this semantic gap, more recent systems increasingly rely on learned embedding-based representations rather than hand-engineered visual descriptors. Modern image and video encoders map visual inputs into dense feature vectors, where nearby vectors often correspond to visually or semantically similar content. CNN-based and vision transformer-based models are commonly used to extract such visual features [79, 59]. CLIP [166] and SigLIP2 [197] extend this idea by aligning image and text embeddings, enabling open-vocabulary retrieval and zero-shot recognition. VDBMSs use related representations for similarity search, clustering, indexing, active learning, and model building. TASTI [103] learns target-aware embeddings so that representative records can stand in for visually similar neighbors. VOCALExplore [53] uses embeddings to support interactive exploration and model construction. Zelda [173] uses CLIP embeddings to let users retrieve videos with NL queries. Embeddings are compact and flexible, so they are effective for ranking, clustering, and retrieving candidates. However, their semantics are implicit: an embedding can indicate that two candidates are close, but it does not expose which object, relationship, attribute, or time interval makes a query true. This makes embeddings less suitable for queries beyond retrieval.

A complementary direction is to represent videos through text. Text-based representations associate videos with NL descriptions, such as captions, tags, transcripts, or surrounding metadata. Early work [225, 51] often relied on human-provided or web-derived text, enabling videos to be indexed and retrieved using standard text retrieval techniques. More recent systems generate textual descriptions directly from videos using VLMs. NarVid [90], as an example, uses generated frame-level captions to enrich video representations and improve text-to-video retrieval performance. Text-based representations are attractive because they align naturally with user queries and can leverage existing information retrieval infrastructure. However, they are often lossy because they may fail to preserve fine-grained visual details, precise spatial and temporal relationships, or grounding to specific video regions.

Structured representations organize visual content extracted from videos into explicit entities, properties, relationships, and temporal structure. Scene graphs [96, 113] are a canonical example: they represent objects detected in images as nodes, relationships similarly detected in images as edges, and attributes as node properties. For video, Action Genome [91] extends this idea to spatio-temporal scene graphs that describe human-object relationships over time and shows that these representations improve action recognition. VDBMSs often instantiate the same idea through relational models [99, 67, 21]. In these systems, tables store detections, tracks, attributes, relationships, timestamps, and predicate outputs, exposing visual information to the query processor rather than leaving it hidden in pixels, text, or embeddings. Our VOCAL line of systems presented in this dissertation, including EQUI-VOCAL, VOCAL-UDF, and VOCAL-SIFT adopt this relational approach to represent video content, which provides rich expressiveness while enabling queries with precise and well-understood semantics.

2.3 Query interfaces

The query interface is the outermost layer of a VDBMS, connecting users to the rest of the system. The choice of interface shapes both how expressively a user can describe an

event and how much expertise is required to do so. Existing VDBMSs adopt a range of interfaces, from explicit declarative and imperative specifications to sketches, examples, and NL descriptions. Below we discuss the main approaches and indicate where the systems in this dissertation fit within them.

Declarative and Imperative Query Specifications. The most direct interface lets users write queries declaratively or imperatively. Many VDBMSs adopt this style by extending SQL with visual predicates and video-specific operators [216, 21, 116]. ReCall [67] introduces a flexible Python library for specifying spatio-temporal video events, while VQPy [227] offers an object-oriented programming interface rather than a relational abstraction. These approaches give users precise control over query semantics, but require them to manually translate visual intent into formal query constructs, which can be challenging for non-experts and time-consuming even for experienced users.

Query Sketching. To reduce this articulation effort, some systems instead let users specify target events through visual sketches [43, 31, 214]. Sketch interfaces are intuitive, but they can be brittle because real events of interest may deviate from the exact user-provided sketch, and predicates such as semantic relationships can be difficult to express in a sketch.

Query-by-Example. Query-by-example takes a complementary approach. Instead of writing a complete declarative query or sketching an event, users provide a small set of examples that illustrate the target semantics, and the system reverse-engineers a query that matches them. Quivr [143] is a query-by-example system for object trajectory queries. Given a few labeled trajectories indicating whether each trajectory matches the desired behavior, Quivr synthesizes a query that satisfies the examples using enumerative search with efficient pruning. Our EQUI-VOCAL work, which we present in Chapter 3, builds on this line of work but targets noisy, spatio-temporal scene-graph queries over videos. Rather than relying on exhaustive enumeration, it uses beam search to explore the query space efficiently and synthesizes queries that remain robust under noisy visual data.

NL interfaces. NL interfaces go a step further by letting users specify visual intent di-

rectly in text. Early work such as Visual Semantic Search [127] parses textual video queries into semantic graphs and matches them to visual concepts using object appearance, motion, and spatial relations. More recent systems use cross-modal models to directly align text with visual content. Modern VLM-based systems such as Zelda [173] support open-ended queries by using vision-language embeddings to retrieve relevant and diverse video frames. These systems reduce query articulation effort and are closely related to our goal of making video search more accessible. However, they may generalize poorly to new video domains or fine-grained event semantics, and they typically provide less control over the exact objects, relationships, temporal constraints, and execution semantics than declarative compositional queries. Our VOCAL-UDF and VOCAL-SIFT systems, which we present in Chapters 4 and 5 respectively, leverage advances in LLMs and VLMs and use NL as the query interface. VOCAL-UDF compiles NL queries into compositional programs over visual UDFs, which provides more precise control over the predicates and execution semantics than direct embedding-based approaches. VOCAL-SIFT relies on VLM’s visual reasoning capabilities to answer high-level queries, but introduces necessary-predicate filtering to reduce the high computational cost of direct VLM evaluation.

2.4 Query execution

Executing video queries efficiently is a central challenge for VDBMSs: a single query can require running heavy AI models over massive video frames, and naively doing so is quickly prohibitive. A substantial body of work addresses this problem. We highlight five common families of techniques: (i) specialized models that replace or reduce calls to expensive oracle models, (ii) indexing techniques that shift visual computation into preprocessing, (iii) sampling strategies that exploit temporal redundancy, (iv) configuration tuning that navigates accuracy–cost trade-offs, and (v) additional optimization techniques that leverage domain knowledge, reuse, and adaptive execution.

2.4.1 *Specialized models*

The first family of techniques speeds up query execution by replacing or reducing calls to an expensive oracle model with a cheaper specialized model whose scope is narrower than a general-purpose model, exploiting properties of a particular query or data distribution.

NoScope [100] trains difference detectors and specialized models on the output of the oracle model, applies them efficiently to short-circuit the evaluation, and only invokes the expensive oracle model when uncertain. Similarly, PP [136] constructs lightweight predicates to filter out irrelevant frames before running expensive models. Koudas et al. [111] design approximate filters to accelerate queries involving specific object types and spatial relationships. TAHOMA [12] trains a multi-level classifier cascade of increasing complexities and jointly optimizes model architectures and input data transformations to achieve better accuracy-cost trade-off. BlazeIt [99], ABAE [102], and SUPG [101] use proxy models to approximate different types of queries with statistical guarantees.

2.4.2 *Indexing*

Another line of work speeds up query execution by shifting visual computation from query evaluation to pre-processing and building indexes over the resulting metadata. The key design question is what to index before the system knows the exact future queries.

Some systems build approximate semantic indexes over frames. Focus [86] uses cheap CNNs at ingest time to index object classes and then selectively invokes a more accurate CNN at query time. SEIDEN [19] applies an accurate object detection oracle to selected frames to build a query-agnostic index of objects and samples additional frames during query execution. PAINE [82] builds a lossy index of object classes in each video during preprocessing. At query time, it uses commonsense knowledge to estimate conditional probabilities and prioritize videos that are likely to contain the requested object.

Other systems index visual similarity rather than explicit object labels. TASTI [103] learns embeddings such that semantically similar frames map to nearby points in the em-

bedding space. This allows query algorithms to run the expensive oracle on only a small set of representative frames and propagate those evaluation results to their un-evaluated neighbors in the embedding space. Voodoo indexing [80] uses hierarchical clustering of frame features to accelerate queries without requiring insight into predicate semantics or in-query model training. Video-zilla [87] builds an indexing layer over many video feeds by representing their semantic content with object feature vectors.

Finally, some indexes are tailored to reusable video structures. OTIF [22] preprocesses object tracks so later track queries do not repeatedly pay the full detection and tracking cost. Boggart [8] builds imprecise but model-agnostic indices with traditional computer vision algorithms, then characterizes index error and selectively runs CNNs for each query.

2.4.3 Sampling

Sampling techniques reduce inference cost by processing only a subset of video frames. They exploit temporal redundancy: adjacent frames are often similar, so a system can spend computation where the query is most likely to benefit.

ExSample [149] targets ad-hoc search over unindexed video repositories and uses importance sampling to prioritize frames that are likely to contain the target object. EKO [18] accelerates queries over compressed videos by adaptively sampling representative frames for inference, avoiding unnecessary decoding of skipped frames, and propagating labels within visually similar temporal segments.

Sampling techniques are also widely used in object tracking queries. MIRIS [21] and OTIF [22] adopt uniform sampling to avoid running the tracker at a uniformly high frame rate. LEAP [215] uses predictive sampling that exploits motion patterns of objects for trajectory predication.

2.4.4 Configuration tuning

Configuration tuning systems reduce cost by choosing how a video analytics pipeline should run rather than changing the query semantics. The knobs include frame rate, resolution,

model choice, batch size, hardware resources, etc.

For streaming workloads, the best configuration can change over time and across cameras. Chameleon [94] dynamically selects configurations such as frame rate, resolution, and model choice by exploiting temporal and cross-camera correlations in video streams. Zeus [49] applies a similar adaptive spirit to action localization, using reinforcement learning to choose the sampling rate, segment length, and resolution for the next video segment.

Other systems tune broader execution environments. Llama [174] models video pipelines as DAGs and uses a heterogeneous serverless optimizer to assign per-operator configurations and hardware resources under an end-to-end latency target. SmokeScreen [81] treats intentional video degradation as a configuration choice and helps administrators select operating points that remove visual information for policy goals while preserving downstream analytical accuracy.

2.4.5 Other optimization techniques

Beyond the major categories above, VDBMSs also exploit domain knowledge, reuse, and adaptive execution. These techniques are harder to classify by a single physical operator, but they share the same goal: avoid unnecessary model calls by exposing additional structure to the optimizer.

Some systems use domain-specific knowledge. Spatialyze [108] targets geospatial video analytics and leverages camera metadata, road networks, and physical constraints to prune frames and objects before expensive processing. VIVA [172] and ClueVQS [32] let users declare optimization hints such as model relationships and monotonic timestamps, and expose them to the query optimizer.

Other systems broaden the optimizer’s state through reuse or adaptive execution. EVA [216] materializes and reuses expensive UDF results across exploratory video queries by analyzing predicate overlap. Aero [97] applies adaptive query processing to ML queries, dynamically adjusting predicate order and resource allocation as execution reveals data-dependent behavior.

2.5 *Foundation models*

More recently, foundation models have transformed visual analytics by shifting recognition from fixed-label classification to open-ended, language-driven interfaces, broadening the queries that a VDBMS can answer. Initially, this progress centers on images. In the visual domain, this shift begins with models that align static images and text. CLIP [166] shows that NL supervision over massive image-text corpora can produce transferable visual representations and zero-shot classifiers. Subsequent models push beyond simple alignment into deep visual reasoning. Flamingo [9] introduces few-shot multimodal learning over interleaved images, videos, and text. BLIP-2 [126] efficiently bridges frozen image encoders with LLMs via a lightweight querying transformer, while LLaVA [129] demonstrates how visual instruction tuning could create an LLM-style conversational interface for images. This foundation rapidly expands to the temporal domain, extending language-driven queries to video. Models such as Video-ChatGPT [139] and TimeChat [171] enable detailed conversations about temporal narratives and time-sensitive localization, allowing systems to interpret complex video events using NL. To process these multimodal inputs, these models typically fuse pre-trained vision encoders onto language models using lightweight, task-specific modules. This approach is computationally efficient and highly effective for targeted tasks or domain-specific fine-tuning.

In contrast, natively multimodal model families, such as GPT [154], Gemini [72], and Qwen [17], are trained jointly across text, image, and video from the ground up. This deep integration unlocks massive context windows capable of processing thousands of frames simultaneously, enabling generalized, long-horizon spatio-temporal reasoning without the bottleneck of intermediate summarization. For a VDBMS, both paradigms are valuable: modular systems offer cheap adaptability for custom predicates, while native models provide deep zero-shot reasoning for complex, open-vocabulary queries.

Beyond pure visual understanding, foundation models possess non-visual capabilities that are increasingly critical for designing AI-assisted video data systems. One foundational

capability is semantic parsing and query translation. Building on a long history of text-to-SQL research [229, 232, 123], modern LLMs excel at translating NL into structured query languages or domain-specific languages (DSLs) [160, 69, 58, 203]. This allows a VDBMS to bridge the gap between a user’s high-level intent and the rigorous, executable semantics required by a query engine.

Another critical capability is dynamic code generation. Advanced LLMs can write, debug, and repair functional code snippets [38, 15, 41]. In the context of a VDBMS, this allows the system to autonomously generate user-defined functions (UDFs) or custom data-processing modules on the fly without relying on a static library of operators.

Despite these transformative capabilities, foundation models cannot independently replace the architecture of a video data management system. Current VLMs still frequently struggle with compositional reasoning, precise spatial relationships, and tracking long-range causal dependencies [137, 206, 35, 141]. Furthermore, directly invoking a frontier VLM is computationally expensive, introduces high latency, and remains fundamentally opaque [37, 25]. While a single model call might suffice for inspecting a short clip, executing analytical queries across a large-scale video corpus demands predictable semantics, verifiable accuracy, and strict cost controls. This underscores the necessity of hybrid AI-database systems that leverage foundation models for their open-world semantic flexibility, while rigidly constraining them within a framework of structured representations and efficient query optimization.

2.6 *Agentic systems*

While foundation models exhibit robust reasoning and planning capabilities, there has been a paradigm shift from static LLMs to autonomous agentic systems for their improving performance in complex data environments. Multiple agents collaborate jointly by iteratively interacting with the environment, maintaining stateful memory, and executing actions to solve complex tasks. Unlike a standalone LLM, an agentic system can detect its own

failures, self-reflect, and dynamically adjust its execution plan based on intermediate observations [208]. Two capabilities underpin this paradigm and recur throughout the rest of this section: *task decomposition and planning*, and *tool use*.

2.6.1 Task decomposition and planning

A core driver of the agentic paradigm is the LLM’s capacity for task decomposition and planning. When confronted with a complex analytical request, language models can act as high-level planners: they break down a goal into a sequence of intermediate subtasks, determine which external models or tools to invoke, and decide how to aggregate the results [222, 188, 190, 191]. This reasoning enables hybrid VDBMS architectures to intelligently orchestrate a pipeline of cheap specialized classifiers, heuristics, and expensive frontier models, rather than applying a single model homogeneously across all data.

2.6.2 Tool use

To execute the subtasks produced by a planner, agentic systems rely on tool use. Foundation models inherently lack access to real-time data or specialized execution environments. Agentic systems bridge this gap by equipping the LLM with a predefined set of external tools, such as web search APIs, Python interpreters, or database query engines [182, 188]. By integrating external tools, LLMs can address even more complex reasoning tasks [134, 182, 164, 240, 138], including vision tasks [147, 48, 75, 194, 188]. However, these systems generally rely on the availability of existing tools or modules. For instance, VisProg [75] and ProViQ [48] utilize LLMs to transform complex tasks into executable programs that invoke predefined tools. VOCAL-UDF also employs LLMs to handle compositional queries over videos but extends this capability by generating new UDFs. Several systems explore the capacity of LLMs to create new tools. LATM [29] generates reusable code snippets for NL tasks, while GENOME [45] generates and reuses code-based modules to solve visual tasks. VOCAL-UDF distinguishes itself as an end-to-end VDBMS that generates both program-based and distilled-model UDFs which significantly enhances performance.

2.6.3 *Agentic data systems*

Within the database community, this combination of planning and tool use is increasingly deployed to automate complex query execution and data analytics workflows. CAESURA [199] and Palimpzest [128] utilize LLMs as query planners to dynamically compile user intents into execution pipelines that navigate the complex trade-offs between inference cost, runtime, and accuracy. Galois [181] adapts traditional logical and physical optimization principles to execute SQL queries directly over LLMs, actively balancing execution costs with the model’s contextual reasoning constraints. Adda [133] utilizes specialized LLM agents to interpret NL requirements and dynamically compile high-quality, in-database features via just-in-time SQL and user-defined functions. Moving beyond structured tabular data, DocETL [187] employs an agent-based approach to rewrite and optimize complex document processing pipelines by leveraging LLMs to evaluate and synthesize task-specific execution plans. AVA [219] demonstrates how VLMs can use agentic retrieval and planning over continuous event knowledge graphs to conduct open-ended analytics on ultra-long video streams.

For video analytics, an agentic planner can decompose a user request, select or generate appropriate visual operators, decide when to invoke cheap proxies versus expensive foundation models, and revise the plan based on intermediate results. VOCAL-UDF and VOCAL-SIFT instantiate this idea in complementary ways. VOCAL-UDF uses LLMs not only to translate NL queries into compositional programs, but also to generate new visual UDFs when the required predicates are missing, thereby extending the system’s operator library at query time. VOCAL-SIFT uses foundation models for semantic decomposition: it identifies necessary predicates for a high-level query and then converts them into efficient filters, using LLM reasoning to expose structure that a traditional optimizer could exploit. Together, these systems show how agentic planning can be grounded inside a VDBMS: foundation models provide semantic flexibility, while structured query representations and database optimization provide control over cost, execution, and accuracy.

2.7 Additional related work

The preceding sections survey research aimed directly at video data management. The systems presented in this dissertation also build on broader research traditions that extend beyond the video setting. We close the chapter by reviewing the most directly relevant of these works for each system.

2.7.1 Additional related work for EQUI-VOCAL

Query by example. Many systems have been proposed for SQL queries over relational data [162, 63, 195, 124, 204]. EQUI-VOCAL focuses on learning queries for video events, with a more constrained form than general relational queries. Quivr [143] is most similar to our work, but it only operates over object trajectories and assumes noiseless inputs. SQuID [63] synthesizes queries based on semantic similarity but does not support self-joins and non-equi joins, which are necessary to find our video events, and S4 [162] ranks queries based on input containment and is limited to project-join queries.

Program synthesis. Program synthesis has been applied to a wide range of tasks, such as generating referring relational programs [89], authoring visualizations [205, 157], learning relational data transformation [98], and synthesizing programs for string processing [152]. Unlike these systems, EQUI-VOCAL synthesizes queries in the video domain, which is significantly different in terms of the spatio-temporal complexity and the prevalence of noise.

Active learning. Model Picker [104] uses active learning to distinguish the best model from a set of pretrained classifiers. EQUI-VOCAL adapts this approach to select the most promising queries to explore. Quivr [143] also prunes candidate queries using active learning, but only after enumerating all candidates. In contrast, EQUI-VOCAL integrates active learning and labeling into the search process to ensure both synthesis efficiency and query performance.

2.7.2 Additional related work for VOCAL-SIFT

Language-guided visual decomposition. Prior work has used auxiliary semantic descriptions, such as attributes, to recognize visual categories beyond those seen during training [119]. More recent work uses language models to enrich category names with visual descriptions or concepts. CuPL [161] generates descriptive prompts for zero-shot classification, Menon and Vondrick [145] compare images against LLM-generated category descriptors, and LaBo [221] uses LLM-generated concepts to construct interpretable concept bottleneck models. These methods show that language knowledge can expose visual properties useful for recognition. VOCAL-SIFT uses a similar source of knowledge for a different purpose: it decomposes high-level video queries into necessary predicates that can be aligned with executable visual filters. Rather than directly classifying images from class descriptions or concepts, VOCAL-SIFT uses these predicates as high-recall conditions for pruning videos before expensive VLM evaluation.

User clarification. Interactive clarification has long been used to resolve ambiguity in NL interfaces [159, 123, 70]. Recent LLM-based work extends this idea in two forms. One line trains or fine-tunes models to ask clarifying questions or otherwise handle ambiguity [107, 14, 238], while another relies on prompting or in-context learning without task-specific training [114, 237, 56]. These methods typically detect ambiguity, ask follow-up questions, and then refine the query or response using additional user feedback. Adapting this approach to video querying that includes high-level predicates could allow the system to ask the user for the objects, relationships, and attributes that should hold for the target event. However, this approach places substantial burden on the user. VOCAL-SIFT instead performs this reasoning automatically. Our user study in Section 5.5.7 shows that this automatic decomposition reduces task completion time compared with human clarification, without degrading filtering quality or task success.

Chapter 3

EQUI-VOCAL: SYNTHESIZING QUERIES FOR COMPOSITIONAL VIDEO EVENTS FROM LIMITED USER INTERACTIONS

In this chapter, we present EQUI-VOCAL (Figure 3.1), our system for querying video databases by enabling users to provide a few positive and negative examples of the compositional events they are trying to find. As we discussed in Chapter 1, a video database management system (VDBMS) enables users to query a video database declaratively. Systems that pre-date EQUI-VOCAL would typically either train a model for the event of interest [30, 74] or use extended SQL for this purpose [67, 33, 131, 43, 217, 143, 23]. The first approach can be impractical in many settings as it requires a large quantity of labeled examples for each new event [201, 241]. The latter approach is also challenging as it requires expertise in SQL, and it can even be tedious for SQL experts because there are typically many plausible ways to express the same event and the best formulation depends on the data.

Consider the following example. Suppose a traffic engineer wants to understand road hazards involving car and motorcycle interactions (e.g., motorcycles swerving abruptly in front of turning cars). Although many computer vision models exist that detect common objects (e.g., “cars” and “motorcycles”) [212] and relate objects spatially (e.g., “bottom of”, “left of”, “near”) [112], a specific classifier that identifies “a motorcycle swerving in front of a car, while the car is turning at an intersection” is unlikely to exist [30, 74]. Worse, training one would require many hours of user effort in labeling for a single query. Given the relative rarity of most interesting events, finding sufficient positive instances further

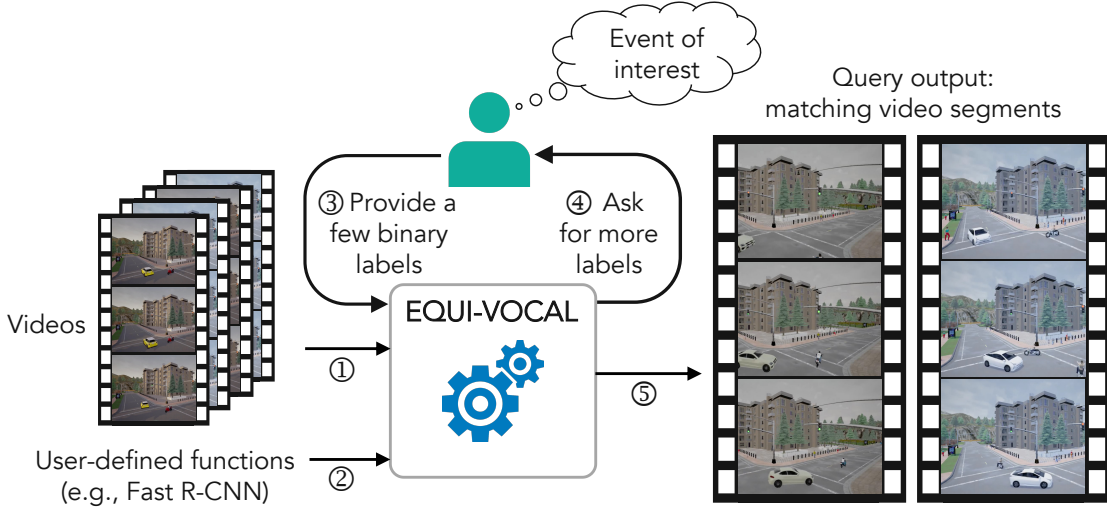


Figure 3.1: EQUI-VOCAL workflow. Given ① a video dataset, ② user-defined functions that extract semantic information from videos, and ③ a few user-provided labels, EQUI-VOCAL synthesizes a query to find instances of an event of interest. It iteratively ④ asks the user for more labels to reduce its uncertainty. Once synthesized, it ⑤ executes the query to return matching events on unseen videos.

exacerbates these labeling requirements. In our example, there will be many instances of cars and motorcycles in intersections. Only rarely would a motorcycle swerve in front of a turning car. Even when the engineer instead tries to express this event declaratively—for instance, by composing primitives such as **near** and **front_of**—there are multiple plausible formulations, and the best one depends on the data (e.g., in a given video, motorcycles may be swerving from outside the frame, as illustrated in Figure 3.3, discussed later). Systems that ask users to sketch their events [43, 31] have the same brittleness: an event of interest may deviate from the exact user-provided sketch.

EQUI-VOCAL avoids these difficulties by synthesizing declarative queries on behalf of users from a small number of labeled video segments. As shown in Figure 3.2, the user labels a few positive and negative examples of the event of interest. EQUI-VOCAL then synthesizes a declarative query and executes it on the remaining large pool of video data to identify other examples of the desired event.

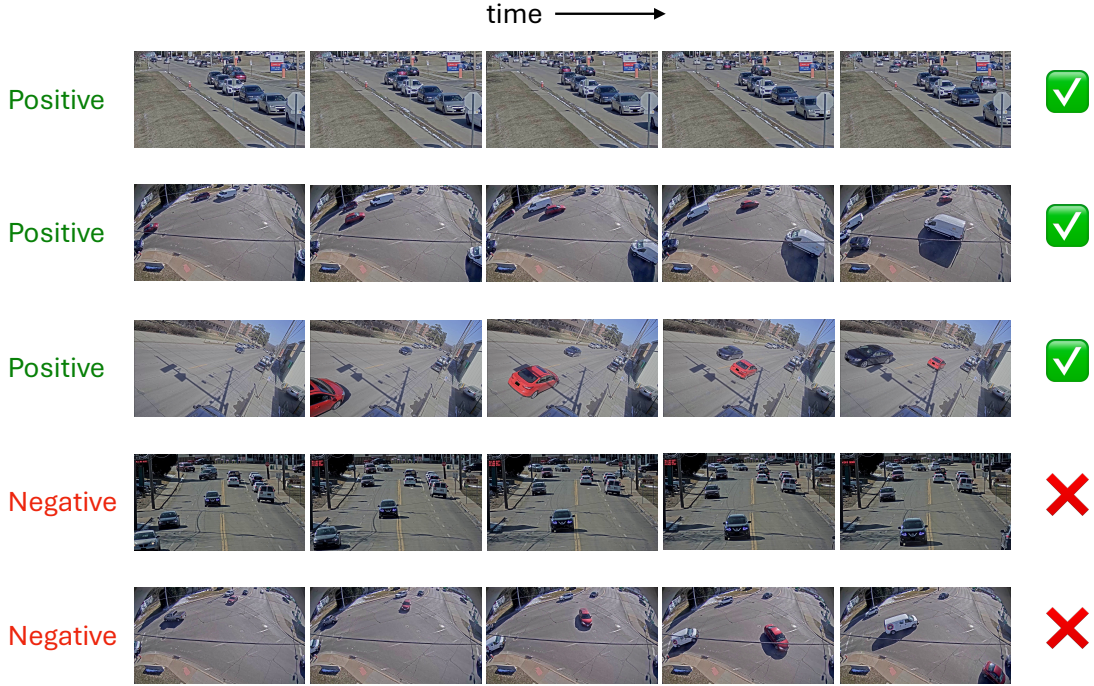


Figure 3.2: An example of user-provided labeled video segments. Each row shows frames sampled from one video segment, ordered from left to right in time. The user provides three positive examples and two negative examples for the query: “A red car is initially in front of another car, then drives behind and near that car.” In our experiments, EQUI-VOCAL can synthesize queries from as few as two positive examples and ten negative examples labeled with respect to the event of interest.

EQUI-VOCAL supports users’ search for complex events without requiring any specific domain knowledge, with limited effort, and without requiring precise declarative specifications or sketches. Similar to other systems, we use the insight that while most user queries are new and unseen, they are usually composed of known, previously seen atoms. Computer vision models for common atoms already exist such as objects (e.g., “car”, “backpack”) and their spatial and semantic relationships (e.g., “left of”, “holding”). Formally, such a composition of visual scenes is referred to as a scene graph in the computer vision community [113, 91] and was developed from its cognitive grounding in human perception [230, 117, 24]. Key to EQUI-VOCAL’s contribution is to encapsulate spatio-temporal scene graphs in its data

model and use them to define a query language. A spatio-temporal scene graph conceptualizes the contents of a video as a sequence of graphs, each graph representing a single video frame. Each graph contains vertices, which represent objects in the frame; edges represent the relationships between those objects. Each object can possess a set of attributes that describe its properties (e.g., “red”, “leather”). EQUI-VOCAL extracts relevant data from each video using user-provided functions: i.e., pre-existing detectors and classifiers; it synthesizes queries as a composition of extracted scene graph atoms. We show that EQUI-VOCAL’s data model and query language, both based on the relational model, can express a variety of compositional queries.

Leveraging scene graphs, EQUI-VOCAL contributes a new query synthesis approach that finds user events with far fewer labeled examples than would be required to train a specialized machine learning model directly, and that works on noisy, video-scale data and complex events. To support query synthesis in such an environment, EQUI-VOCAL solves two technical challenges: it reduces **computational effort** and **user effort**.

EQUI-VOCAL reduces **computational effort** by limiting query search using scene graphs, by pruning search paths using beam search, and by avoiding expensive database operations. First, unlike prior query-by-example techniques that synthesize arbitrary SQL [63, 162, 124, 204, 195], EQUI-VOCAL reduces the search space by limiting the query search to sequences of scene graphs. Second, synthesizing queries over sequences of scene graphs can still be a computationally slow process to traverse the search space of possible queries. Existing query-by-example systems enumerate all possible queries; although pruning techniques like equivalence classes [198], over-approximation [204], and lifting projection operators [195] can be used to reduce the search space, these mechanisms are not sufficient to make exhaustive exploration tractable. Instead, EQUI-VOCAL adopts a beam search strategy to explore the query space efficiently. Beam search limits exploration to a subset of the most promising branches at each step. Third, executing the many candidate queries on the user examples is prohibitively expensive. Existing systems [204, 195] evaluate candidate queries with many joins and thus do not scale well when the size of user examples

becomes large. EQUI-VOCAL carefully generates efficient queries that avoid expensive operations such as recursive joins. EQUI-VOCAL comes with a set of optimizations to generate efficient SQL statements and uses the PostgreSQL database engine to execute them.

EQUI-VOCAL reduces **user effort** by using active learning and by being robust to noise. With active learning, EQUI-VOCAL reduces the number of labeled examples needed: instead of asking a user to provide all examples up front, EQUI-VOCAL iteratively requests labels of carefully selected additional examples to reduce the uncertainty in query synthesis. Noise can naturally creep into systems that interface with user labeling, machine learning models, and potentially ambiguous real-world events. Distinct from other existing systems [195, 143], EQUI-VOCAL searches for queries that best match potentially noisy data and input. It also retains imperfect query candidates at every iteration and uses regularization to prevent overfitting to noise or limited user input.

In summary, EQUI-VOCAL makes the following contributions:

- We introduce an expressive data model and a query language based on spatio-temporal scene graphs (Section 3.1).
- We propose a new approach that efficiently synthesizes the user’s intended query from examples. We limit the query search space using scene graphs, prune search paths using beam search, leverage active learning to reduce user effort, and handle noisy data (Section 3.2).
- We implement a set of optimizations to generate efficient SQL query statements and reduce computational effort during query synthesis (Section 3.3).
- We evaluate our approach on synthetic and real datasets [224, 21] and show that it outperforms two baselines [195, 143]—in terms of F1 score, synthesis time, and robustness to data noise—and can flexibly synthesize complex queries that the baselines do not support. We also conduct a user study to show the performance of real users using EQUI-VOCAL (Section 3.4).

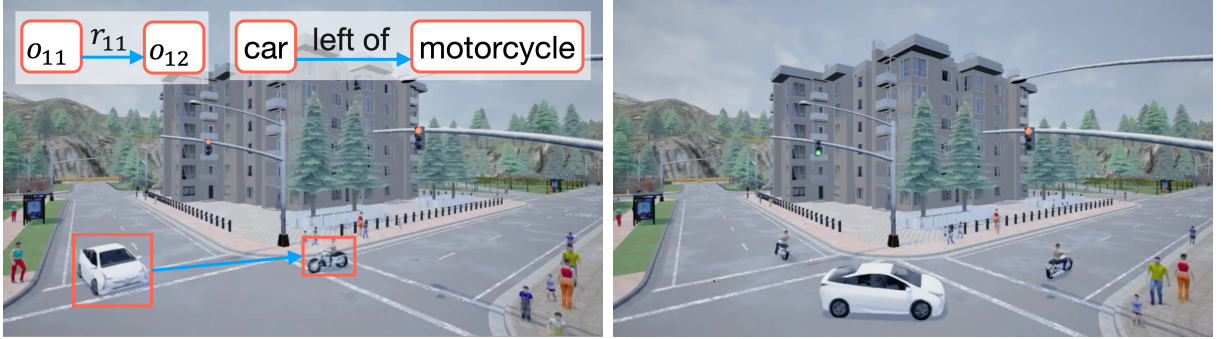


Figure 3.3: Example frames of multiple, simultaneous car-motorcycle interactions (generated using [77]). EQUI-VOCAL represents video content as a sequence of region graphs in its data model. Each region graph models a single video frame (left figure), with nodes representing objects and edges representing relationships. A region graph is a subset of the full scene graph (not shown).

Overall, EQUI-VOCAL is an important step toward making video database management systems more accessible to experts and non-experts alike, by easing the task of expressing queries over videos.

3.1 EQUI-VOCAL data model

This section describes EQUI-VOCAL’s data model and query language, which we briefly introduced in our vision paper [52], but develop in depth here. Section 3.2 shows how EQUI-VOCAL synthesizes queries from user input using this data model.

For ease of presentation, we use a simplified, running example, where a traffic engineer seeks to find instances of “a car arriving from the left and passing a motorcycle at the intersection.” Figure 3.3 shows two representative frames from a video that contains such an event. We show other example queries in Section 3.4.

3.1.1 Scene graphs as our data model

EQUI-VOCAL represents a video V as a set of short, non-overlapping *video segments*, $v \in V$ (5-second segments in our prototype implementation). Each video segment is a sequence

of N frames $\{f_1, \dots, f_N\}$. We represent each frame as $f = (vid, f_i, pixels)$, where vid and f_i identify the video segment and the frame index within that segment, and $pixels$ stores the frame’s pixel values as a $H \times W \times 3$ array (H is height, W is width, and 3 denotes the color channels). The **Frames** relation in Table 3.1 stores one row per frame; in our prototype, $pixels$ is implemented as a virtual column and is accessed only when a user-defined function requires raw frame data. The visual content of each frame is represented by a scene graph [113]: A *scene graph* $g_i = (\mathbf{o}_i, \mathbf{r}_i)$ contains the set of all objects $\mathbf{o}_i$ in a frame, along with a set of all relationships $\mathbf{r}_i$ between those objects. Often a scene graph contains more information than is necessary to identify an event, and so the literature also defines a *region graph* g_{ij} , which is a subgraph of g_i , i.e., $g_{ij} \subseteq g_i$. Figure 3.3 (left) shows an example frame and region graph.

We define an *object* in a frame as $o = (vid, f_i, oid, oname, bbox)$, where f_i is the sequence number of frame i in video segment, vid . oid is a unique identifier of the object in the video segment, $oname$ is the class name of the object (e.g., “car”, “motorcycle”), and $bbox$ is the bounding box containing the object in frame, f_i . A $bbox$ is represented by its upper-left and bottom-right coordinates, i.e., $bbox = (x_1, y_1, x_2, y_2)$.

Objects can have intra-frame *relationships* defined as $r = (vid, f_i, rid, oid_{\text{sub}}, rname, oid_{\text{tar}})$, where rid is a unique identifier of the relationship in frame, f_i . Subject, o_{sub} , is connected to target, o_{tar} , with the relationship class name, $rname$ (e.g., “near” or “holds”). Both subject and target belong to frame f_i : i.e., $o_{\text{sub}}, o_{\text{tar}} \in \mathbf{o}_i$.

Objects can have *attributes* $a = (vid, f_i, oid, aname)$, where $aname$ is the name of the attribute, and vid , f_i , and oid identify the video segment, frame, and object. EQUI-VOCAL distinguishes *state* and *property* attributes. The former change over time and are typically computed from the bounding box of an object (e.g., “bottom”). The latter capture intrinsic properties of objects and are immutable (e.g., “red”).

Finally, an *event* e is a temporally ordered sequence of region graphs $e = (eid, \{g_1, \dots, g_k\})$. Region graphs in an event do not need to be contiguous or distinct.

Table 3.1: Relational schema representation of data model.

Frames (vid, fid, pixels)
Objects (vid, fid, oid, oname, x_1 , y_1 , x_2 , y_2)
Relationships (vid, fid, rid, oid ₁ , rname, oid ₂)
Attributes (vid, fid, oid, aname)

Example. Suppose that the two frames in Figure 3.3 are the 10th and 15th frames of a video segment **V1**, and that the motorcycle and car are the 7th and 9th objects detected in **V1**. Then, for the left frame, we generate the region graph $g_1 = (\mathbf{o}_1, \mathbf{r}_1)$, where: $\mathbf{o}_1 = \{o_{11}, o_{12}\}$ represents the car $o_{11} = (\mathbf{V1}, \mathbf{F10}, \mathbf{OID9}, \text{car}, \text{bbox}_1)$ and the motorcycle $o_{12} = (\mathbf{V1}, \mathbf{F10}, \mathbf{OID7}, \text{motorcycle}, \text{bbox}_2)$ and $\mathbf{r}_1 = \{r_{11}\}$ contains a relationship $r_{11} = (\mathbf{V1}, \mathbf{F10}, \mathbf{RID1}, \mathbf{OID9}, \text{leftOf}, \mathbf{OID7})$. The car also has an attribute $a_{11} = (\mathbf{V1}, \mathbf{F10}, \mathbf{OID9}, \text{bottom})$. We can define the region graph g_2 for the right frame similarly. The only difference will be the relationship between the object will indicate that the car is now rightOf the motorcycle. Finally, the event $e = (\mathbf{EID1}, \{g_1, g_2\})$ represents a car arriving from the left and passing a motorcycle at the intersection.

The relational schema in Table 3.1 captures the above data model. The benefit of using a relational schema is that we can execute relational queries to specify region graphs and find events of interest, which is flexible and follows the well-understood semantics of the relational model. For each video (or collection of related videos), EQUI-VOCAL creates a *view* with this schema. In Section 3.3 we describe when and how relations in the view are materialized.

To populate each relation, EQUI-VOCAL uses available user-defined functions (Figure 3.1). User-defined functions can be provided by the user or be available publicly in the form of existing machine learning models, such as object detectors. Various user-defined functions can be declared in EQUI-VOCAL: (i) an object detector [170] that takes a video frame as input and outputs the set of objects and their bounding boxes, (ii) an object tracking algorithm [212] that takes objects in consecutive frames as input and, for each pair of

objects, determines if they are the same, and (iii) a set of pre-trained models (e.g., [112]) or rules that can take two objects in the same frame as input and determine their relationship (e.g., “near”, “behind”, “riding”, “holding”) or that can take one object as input and determine its attributes (e.g., “bottom”, “red”). In our experiments, EQUI-VOCAL uses a general-purpose object detector [78] to locate objects and intrinsic attributes of objects to generate trajectories across frames [224].

3.1.2 Scene graphs as our query language

EQUI-VOCAL could execute arbitrary relational queries on the view defined above. However, this would form an intractable search space, making query synthesis unusably slow for most real-world applications. Instead, we define a query language that is more restrictive, affording a smaller search space and therefore, faster synthesis. We constrain queries to (i) a temporally ordered sequence of region graphs, (ii) a set of predicates, (iii) a set of duration constraints, (iv) a window specification, and (v) to output video segment identifiers. Using Datalog and with some abuse of notation, a query in EQUI-VOCAL can be expressed as:

$$q(vid) \text{ :- } g_1, \dots, g_k, \mathbf{p}, \mathbf{d}, w, \text{ where:}$$

- A temporally ordered sequence of region graphs $g_1, \dots, g_k$ specifies that a matching event consists of g_1 , followed by g_2 , followed by g_3 , etc. Each g_i is specified with a set of atoms: **Objects**, **Relationships**, and **Attributes** joined on a shared vid and fid . Moreover, each g_i can persist for multiple frames and there can be other frames between g_i and g_{i+1} .
- A set of predicates $\mathbf{p}$ can be applied to objects, relationships, and attributes. In our example, predicates would specify that the query is looking for “car” and “motorcycle” objects, that the car needs to be “left of” then “right of” the motorcycle, and that the car should be at the “bottom” of the frame.

- A set of duration constraints $\mathbf{d}$ can be applied to region graphs and define the minimum number of contiguous frames that a region graph g_i should be valid before transitioning to the next region graph g_{i+1} .
- A window specification w is the maximum number of frames that can separate g_1 from g_k .

For example, following the above restricted template, the event in Section 3.1.1 can be expressed with the following Datalog rules:

```
g1(vid, fid, oid1, oid2) :-
    Objects(vid, fid, oid1, 'car', _, _, _, _),
    Objects(vid, fid, oid2, 'motorcycle', _, _, _, _),
    Relationships(vid, fid, _, oid1, 'leftOf', oid2),
    Attributes(vid, fid, oid1, 'bottom'), oid1 != oid2.
```

The above rule will find frames in video segments that contain a car and a motorcycle, such that the car is on the left of the motorcycle, and the car is at the bottom of the frame, where the intersection is located. Next, the event will likely consist of a sequence of such frames, which can be captured with the following recursive rules:

```
g1_star(vid, fid, fid, oid1, oid2) :- g1(vid, fid, oid1, oid2).
g1_star(vid, fid_start, fid_end, oid1, oid2) :-
    g1_star(vid, fid_start, fid, oid1, oid2),
    g1(vid, fid_end, oid1, oid2), fid_end = fid + 1.
```

We could use equivalent rules to define $g2$ and $g2_star$. Finally, the query that returns matching video segments takes the form:

The predicate $fid21 > fid12$ indicates that the second sequence of region graphs should follow the first one. The predicate allows for a gap between sequences, which may arise, for example, if something obstructs the vehicles from the camera's view. Finally, $fid22 - fid11 <$

```

q(vid) :- g1_star(vid, fid11, fid12, oid1, oid2),
          g2_star(vid, fid21, fid22, oid1, oid2),
          fid21 > fid12, fid22 - fid11 < 1800.

```

Table 3.2: Comparison between compositional video analytics systems.

	SVQ++ [33]	Chen et al. [44]	Caesar [131]	STAR [43]	VidCEP [217]	CVQL [116]	Quivr [143]	Rekall [67]	Ours
Object detection	✓	✓	✓	✓	✓	✓	✓	✓	✓
Object tracking		✓	✓	✓	✓		✓	✓	✓
Relationship	✓		✓	✓	✓	✓	✓	✓	✓
Attribute				✓	✓			✓	✓
Conjunction		✓	✓	✓	✓	✓	✓	✓	✓
Sequencing			✓	✓	✓	✓	✓	✓	✓
Iteration		✓				✓	✓	✓	✓
Window		✓			✓			✓	✓
Query by example							✓		✓

1800 puts a time constraint on the event (a 30-second time-window, assuming 60 frames per second).

To summarize, EQUI-VOCAL’s query language is a subset of Datalog with recursion, expressed on a specific schema. In Section 3.3, we explain how we avoid executing expensive recursive queries.

3.1.3 Expressiveness of our data model

We compare the expressiveness of EQUI-VOCAL’s data model against other compositional video analytics systems in Table 3.2. Among them, SVQ++ [33] only supports spatial relationships between two objects, Chen et al. [44] only supports temporal queries that count co-occurring objects. Caesar [131], STAR Retrieval [43], and VidCEP [217] lack the important feature of *iteration* that allows a region graph to persist for multiple frames and thus cannot support duration constraints. CVQL [116] does not track objects and all its predicates are defined at the object class level. Quivr [143] has similar expressiveness to us,

but is limited to trajectory queries. Recall [67] introduces a flexible Python library for video event specification, but requires that users manually write and refine queries.

Our data model currently does not support tertiary relationships between three objects, such as “a person hitting a ball with a bat”. The closest approximation we have is “a person holding a bat” and “a bat hitting a ball”. We also do not support disjunctions or negations. Several existing systems have support for these operators. For example, Chen et al. [44] supports arbitrary CNF queries, Caesar [131] supports disjunctions, and Recall [67] can flexibly support all three operators. However, the focus of these systems lies in query execution, rather than in learning query specifications from examples, which is the primary focus of this work.

Datalog and scene graphs serve as the foundation of our query language. By building on the relational model and Datalog, we ensure that our queries have precise and well-understood semantics. By focusing on queries that specify a desired sequence of region graphs, we constrain the search space for queries, which accelerates synthesis, we are able to optimize the execution of those queries (compared with trying to support arbitrary recursive Datalog queries), and we are still able to capture events that occur in videos, where objects interact with each other in space and time.

3.2 Query synthesis

With our data model and query language defined, we now formally present the query synthesis problem statement, then describe all the components of our proposed solution.

3.2.1 Query synthesis problem statement

Following the aforementioned data model, a user would like to execute a query q_u on video database D that returns a set of video segment identifiers, $V_{output} = q_u(D)$. Given the user’s intended query, q_u , each video segment v_i can be seen as having a ground-truth label $y_i \in \{0, 1\}$ indicating whether it matches q_u . Initially, both the ground-truth labels and q_u

Algorithm 1: Query synthesis algorithm that returns top- k synthesized queries matching user input.

Input : U - set of unlabeled video segments
 L - set of labeled video segments
 P - set of user-defined functions
 b, bw, s_q, s_v, k - hyperparameters
Output: Q_t - set of top- k target queries

```

1  $S \leftarrow \{q_\emptyset\}$ 
2 while  $|S| > 0$  do
3    $S' \leftarrow \{\}$ 
4   foreach  $q \in S$  do
5      $S' \leftarrow S' \cup \text{EXPANDQUERY}(q, P)$ 
6   if  $|L| < b$  then
7      $L, U \leftarrow \text{PICKNEXTSEGMENTS}(L, U, S')$ 
8    $S \leftarrow \text{SAMPLEQUERIES}(S', L, bw)$ 
9    $Q_t \leftarrow Q_t \cup S'$ 
10   $Q_t \leftarrow \text{RETAINTOPQUERIES}(Q_t, L, k)$ 
11 return  $Q_t$ 

```

are unknown because the user is unable to specify q_u and can only label video segments as positive or negative instances. The goal of EQUI-VOCAL is to synthesize a target query, $q_t \in Q$, that is the best approximation of q_u in its search space. When executed over database D , q_t should yield the best measure performance (e.g., F1 score):

$$q_t = \arg \max_{q \in Q} \text{measure}(q(D), q_u(D))$$

EQUI-VOCAL can request a label from the user O : $\hat{y}_i = O(v_i)$. Since user labels may be noisy, it is possible that $\hat{y}_i \neq y_i$. Given that enumerative search is intractable, EQUI-VOCAL uses a heuristic approach to find an approximately-best query $\hat{q}_t$ to the objective, while reducing both user effort and query synthesis time.

3.2.2 Query synthesis algorithm overview

Algorithm 1 provides an overview of EQUI-VOCAL’s synthesis algorithm. It traverses the space of possible queries to synthesize a final target query that matches the user’s intent. The algorithm takes as input the set of unlabeled video segments, U , and a small set of labeled segments, L . L is provided by the user and should include both positive and negative examples of the desired event. In our experiments, we require as few as two positive and ten negative examples. While providing two positive examples requires some work from the user, such a setting is common in query-by-example approaches [143, 63, 168]. We leave starting with zero examples for future work. The algorithm also takes as input a set of user-defined functions, P , comprised of indicator functions for object classes (e.g., “car”), relationships (e.g., “near”), and attributes (e.g., “red”). Algorithm 1 enumerates P when expanding its search space.

The algorithm returns a set of target queries, Q_t , which is the top- k synthesized queries ranked by a performance measure (F1 score in our prototype implementation) and is updated regularly through the search process. The learned queries can be applied to unseen videos to find video segments containing the matching event. By default, EQUI-VOCAL returns the top- k queries but then executes the best one over unseen videos. The user can also randomly sample one query from Q_t , execute multiple queries from Q_t and aggregate the results, or manually examine Q_t to pick the intended query. If the user is not satisfied with the synthesized query, they can edit it directly. The user can also restart the search with a larger number of initial examples by reusing the video segments labeled during the last session. Moreover, the user could provide a starter query to EQUI-VOCAL (e.g., a simple query or an earlier query) rather than starting from an empty query.

To populate Q_t efficiently, EQUI-VOCAL synthesizes queries in a bottom-up fashion. It starts with an empty query, $q_\emptyset$, and incrementally adds predicates to it, up to a certain complexity (EXPANDQUERY method on line 5 and Section 3.2.3). At each step, different predicates can be added to a query, expanding the search in multiple directions. To limit the size of the search space and ensure fast query synthesis, EQUI-VOCAL adopts a beam search-style strategy (SAMPLEQUERIES method on line 8 and Section 3.2.4).

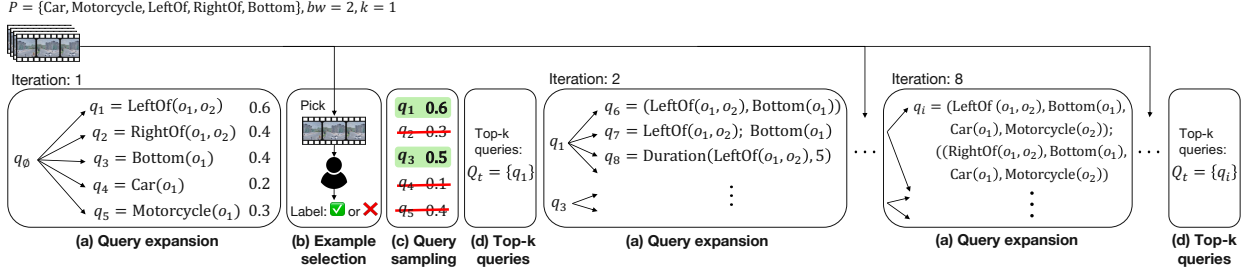


Figure 3.4: Running example of the query synthesis algorithm. The algorithm starts with q_0 and synthesizes queries by iteratively (a) expanding queries, (b) selecting new video segments for the user to label, (c) sampling a small set of queries for further expansion, and (d) updating a list of top- k queries after each iteration. The algorithm returns $Q_t = \{q_i\}$.

An important challenge for EQUI-VOCAL is that learning a query from a small number of user-provided examples is difficult. To address this challenge, EQUI-VOCAL uses active learning to effectively guide the query synthesis process and identify good target queries with limited initial and additional user effort. Specifically, method `PICKNEXTSEGMENTS` on line 7 and in Section 3.2.5 uses active learning to select additional video segments for the user to label in order to effectively differentiate between multiple candidate subqueries, and expand the search in the most promising directions.

At the end of each iteration, method `RETAINTOPQUERIES` (line 10 and Section 3.2.6) maintains a list of top- k queries seen so far, which is larger than the number of queries selected for additional expansion, in case a query seen earlier in the search ends up with the best score on the final labeled set of segments, or, as mentioned above, to give users options if they would like to try alternative, high-performing queries.

The algorithm has several hyperparameters, including a labeling budget b (i.e., the maximum number of labels the user is willing to provide), the number of candidate queries to retain during exploration bw (i.e., the beam width), the number of candidate queries, s_q , and the number of candidate video segments, s_v , sampled during active learning, and the number of queries in the final answer k . Figure 3.4 illustrates the algorithm using the running example.

3.2.3 Query expansion

Existing query-by-example systems use sketch-based query synthesis approaches [195, 204, 143] to enumerate candidate queries. A sketch query is a query with unspecified parts in the form of holes and these approaches enumerate the search space by first generating high-level sketch queries and then filling them with low-level details. However, enumerative search is slow and memory-intensive. Inspired by execution-guided synthesis approaches [42, 89], which treat a program as a sequence of manipulations and use the results of partial programs to guide the search, EQUI-VOCAL explores the search space based on the results of executing intermediate queries on the examples. Instead of synthesizing sketch queries with uninstantiated holes that cannot be executed directly, or directly applying techniques from [42, 89], which require a large amount of data to train a neural synthesizer, EQUI-VOCAL expands queries by adding instantiated and executable constraints, executing partial queries to assess the promise of each explored path, and iteratively refining a query towards the target query.

In our synthesis algorithm, method `EXPANDQUERY( $q, P$ )` takes as input a query q to expand, and a set of user-defined functions, P to construct more complex queries. The function returns a set of expanded queries as illustrated in Figure 3.4(a).

Our query synthesis approach uses a compact query notation, which can be seen as a DSL. The DSL captures the logical structure of the queries to synthesize and key query parameters, but omits the details of the full, underlying SQL (or Datalog). This approach is important for several reasons: First, since we do not generate arbitrary SQL, but rather queries that conform to the structure presented in Section 3.1.2, the DSL captures that structure precisely, simplifying the search space and guiding synthesis toward the correctly-structured queries. Second, this approach helps to decouple the logical query specification from the details of the SQL queries that are ultimately executed. As we present in Section 3.3, critical optimizations are necessary during the translation from our DSL to SQL to achieve efficient query execution.

In our DSL, we use a *variable* o to represent an object in a query. Different variables represent objects with different *oid*'s. All predicates of a region graph are connected by commas and are represented with shorthand notations that specify an attribute name (e.g., **Red**(o_1), **Bottom**(o_1)) or a class name (for objects and relationships, e.g., **Car**(o_1), **LeftOf**(o_1, o_2)). Then, region graphs are connected in sequence with semicolons. For example, the query for the event from Section 3.1.1 can be represented as:

$$q = (\text{Car}(o_1), \text{Motorcycle}(o_2), \text{LeftOf}(o_1, o_2), \text{Bottom}(o_1)); \\ (\text{Car}(o_1), \text{Motorcycle}(o_2), \text{RightOf}(o_1, o_2), \text{Bottom}(o_1)))$$

We further use the notation $\text{Duration}(g, d)$ to require that the region graph g exist in *at least* d consecutive frames.

During query synthesis, EQUI-VOCAL expands queries written in our DSL. **EXPAND-QUERY** takes any of the following three *actions*: (i) Graph construction (GC): Add a predicate to an existing region graph. (ii) Sequence construction (SC): Insert a new region graph consisting of one predicate into any position of the existing sequence of region graphs. (iii) Duration refinement (DR): Increment the duration constraint of an existing region graph in the sequence.

As shown in Figure 3.4(a), EQUI-VOCAL starts with an empty query $q_0 = q_\emptyset$. In iteration 1, EQUI-VOCAL takes action SC to expand q_0 . This results in q_1 to q_5 , each consisting of a single region graph with one predicate drawn from P . In iteration 2 of Figure 3.4(a), EQUI-VOCAL first expands $q_1 = \text{LeftOf}(o_1, o_2)$. Performing action GC leads to $q_6 = (\text{LeftOf}(o_1, o_2), \text{Bottom}(o_1))$; SC leads to $q_7 = \text{LeftOf}(o_1, o_2); \text{Bottom}(o_1)$; and DR leads to $q_8 = \text{Duration}(\text{LeftOf}(o_1, o_2), 5)$, assuming the granularity of **Duration** is 5 frames.

Deciding which actions to take to expand queries is an important design decision in EQUI-VOCAL. Imagine the search space as a directed acyclic graph (DAG), where each node represents a possible query in the search space, the root node represents the empty query $q_\emptyset$, and the query of each child node is constructed by applying one action to its parent node. We consider two extreme expansion rules: A restrictive rule (Figure 3.5a) only allows each query to have one construction path in the DAG, while a relaxed rule allows each

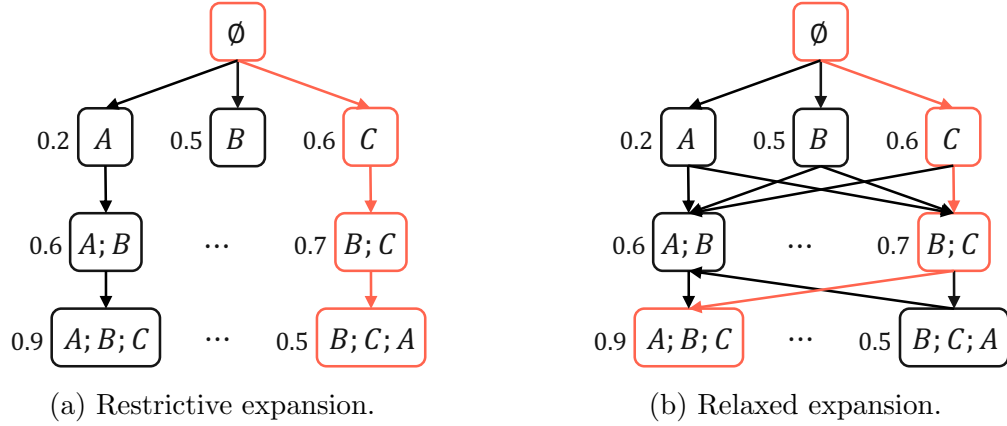


Figure 3.5: A restrictive rule constrains each query to have one construction path, while a relaxed rule allows for multiple paths. Though a relaxed rule results in a larger search space, it is more likely to find queries with high performance than a restrictive rule.

query to be generated through all possible permutations of actions with multiple construction paths (Figure 3.5b). Our early experiments showed that the restrictive rule led to poor performance: In some cases, queries with high performance were not found because their ancestor queries (i.e., intermediate parent queries) were not good and were getting pruned. Since different predicates and their combinations have different selectivity and contribute differently to the final query performance, the relaxed rule allows the algorithm to focus on synthesizing the more dominant part of the query during early iterations, avoiding all paths to the target query being accidentally pruned. The cost of a relaxed rule is a larger search space and slower query synthesis. In the example of Figure 3.5, queries are constructed following the paths highlighted in blue, assuming only the best query is expanded at each step. The restrictive rule selects the query $B; C; A$ with a score of 0.5 rather than $A; B; C$ with a score of 0.9, because A has a lower F1 score than C and B . On the other hand, using the relaxed rule selects $A; B; C$ by first constructing C , then adding B and A to the sequence. EQUI-VOCAL follows the relaxed rule: GC can add any predicates that are not already in the region graph (predicates of the same user-defined function but different variables are considered different); SC can insert a new region graph before and after any existing region

graph; DR can increment the duration constraint of any existing region graph.

3.2.4 Beam search

When traversing the search space, we can greedily expand only the top query or exhaustively expand all. The former is quick to compute but may not find the target query at the end, while the latter is optimal but comes with a prohibitive computational cost. EQUIVOCAL uses beam search to balance query performance and synthesis efficiency. Beam search has a wide range of applications in problems with large search spaces, especially in NLP [135, 115, 200]. The greedy approach can be viewed as a special case of the beam search with a beam width of one, and the exhaustive search is equivalent to a beam search with an infinitely large beam width. Beam search reduces runtime by limiting the number of explored branches at each iteration. However, the search outputs are not guaranteed to be optimal. Search quality depends on how branches are expanded, scored, and pruned. In our approach, method $\text{SAMPLEQUERIES}(S', L, bw)$ retains the top bw queries to expand by evaluating the F1 score of the S' candidate queries on the set of labeled video segments L (see Figure 3.4(c)). Prior work has analyzed the theoretical properties of beam search under certain assumptions (e.g., monotonicity of scoring functions [142], Bayes optimality of tree models under approximations [242]). While in this work, we directly use the F1 score as the scoring function and to retain top queries, other more sophisticated scoring functions and pruning techniques can be used as alternatives [142, 200, 7]. We show the effectiveness of our synthesis algorithm empirically in Section 3.4.

3.2.5 Active learning

One challenge with asking the user for only a handful of examples of the intended event is that we risk overfitting, but asking the user to find a larger number of initial examples is difficult. To address this challenge, we use active learning during query synthesis.

Many active learning methods have been proposed in the literature with the goal of labeling samples that maximally improve a model trained on those samples. E.g., uncertainty-

based sampling [122], estimated error reduction [175], and core-set approach [184]. In our work, we use active learning to label samples that help us identify which candidate queries are better than others. To do so, we need to label samples where candidates disagree: Therefore, we use disagreement-based active learning [104]. At every iteration, EQUI-VOCAL asks the user for a handful of additional labels to differentiate between the bw queries retained from the previous iteration. Intuitively, this method picks the video segments where the retained queries disagree the most. A similar idea appears in the query-by-committee algorithm [186] and has been used in other program synthesis and query synthesis systems [93, 143]. EQUI-VOCAL’s contribution is to incorporate active learning in each iteration of the search process, ensuring an interaction-level labeling experience. Using active learning reduces the number of data points a user has to provide; additionally, we posit that it is easier for users to label system-selected examples than to come up with their own.

During each call to $\text{PICKNEXTSEGMENTS}(L, U, S')$, EQUI-VOCAL computes a score for a sample of unlabeled video segments U over the candidate queries S' and then picks the video segments with the largest disagreement. The score of each sampled, unlabeled video segment is computed as the weighted disagreement between the candidate queries. The weight of each candidate query is set to its regularized performance over the labeled set L (Section 3.2.7). While the algorithm in [104] is designed for the setting where labeling candidates are streamed, EQUI-VOCAL instead maintains a pool of candidates. Therefore, for each call to PICKNEXTSEGMENTS , EQUI-VOCAL computes the score for a sample of unlabeled video segments s_v over a sample of candidate queries s_q and then picks the best one. The function updates L and U given the new user labels. As shown in Figure 3.4(a), EQUI-VOCAL generates five candidate queries q_1 to q_5 in iteration 1. Among them, q_1 has the highest score 0.6, while q_2 and q_3 give the same second highest score 0.4. EQUI-VOCAL requests a new video segment to be labeled (Figure 3.4(b)), which distinguishes q_3 from q_2 (Figure 3.4(c)).

Before query synthesis begins, users can set the labeling budget as a hyperparameter. EQUI-VOCAL precomputes the number of iterations of query expansion it will perform and

uniformly divides the labeling budget among iterations. At every iteration, EQUI-VOCAL asks for new labels only if it has a labeling budget left to do so; otherwise, it proceeds to synthesize queries without requesting new labels. EQUI-VOCAL precomputes the number of iterations by relying on the hyperparameters (Section 3.3) that bound the search process and the query expansion rules (Section 3.2.3), which increments the complexity of synthesized queries in each iteration.

3.2.6 Retaining top queries

At the end of each iteration, EQUI-VOCAL updates its list of candidate final queries in order to retain k best performing ones, as measured by their F1 scores on the labeled dataset. $\text{RETAINTOPQUERIES}(Q_t, L, k)$ evaluates all candidate queries Q_t on labeled set L and returns the top k (Figure 3.4(d)).

In Figure 3.4(d), at the end of iteration 1, EQUI-VOCAL stores q_1 in Q_t because $k=1$, and q_1 has the highest score. When the algorithm terminates, EQUI-VOCAL returns q_i as the final query. In our evaluation, we set the default value of k to 100 (and to 1000 for a more complex dataset), which ensures both low overhead and high-performing final queries.

3.2.7 Regularization

EQUI-VOCAL retains top queries using their F1 scores on the smaller L labeled set. This can lead to overfitting since there are likely too few training examples to accurately evaluate each candidate query. To prevent overfitting, we regularize the score of each candidate query by adding the term: $\text{score}_{\text{reg}}(q) = \text{score}(q) - \lambda \cdot R(q)$, where λ controls the importance of the regularization term. $R(q)$ represents the complexity of the query q : $R(q) = \sum_{i=1}^k (\alpha_1 n_{pi} + \alpha_2 n_{di} + \alpha_3 n_{di} n_{pi})$. Here, k is the number of region graphs in q , n_{pi} is the number of predicates in the i th region graph g_i , and n_{di} denotes the scale of the duration constraint of g_i . α_1 , α_2 , and α_3 are hyperparameters that control the importance of each term.

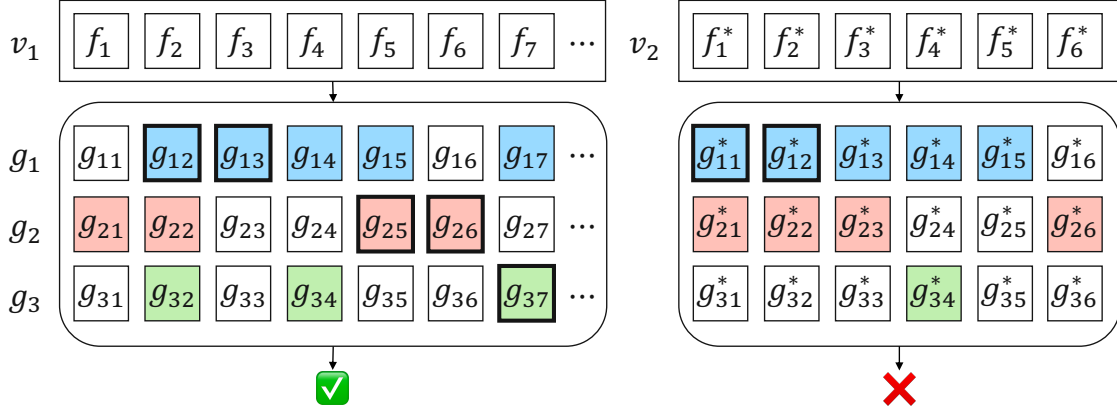


Figure 3.6: Query execution example. EQUI-VOCAL reduces intermediate result sizes by computing the *earliest* matching sequences for each region graph specification.

3.3 Query execution

As discussed in Section 3.2.3, we convert queries in our DSL into SQL and use a relational engine (PostgreSQL in our prototype) to execute them. In this section, we discuss important optimizations that we apply during this translation process.

Query translation algorithm. In Section 3.1.2, we show that queries over videos are naturally recursive. Such queries, however, are slow to execute. To avoid recursion, we leverage the observation that those queries express the idea of iteratively matching a region graph in a *contiguous sequence* of frames, rather than arbitrary recursion. Therefore, we can express them using window functions instead of recursion.

As a second optimization, we note that a query q in our DSL (Section 3.2.3) takes as input a set of video segments V and returns all segments in V containing at least one event that matches q . Therefore, instead of finding all satisfying events for each video segment, the query execution needs to find only one. We leverage this observation to reduce intermediate result sizes by producing SQL that efficiently computes the *earliest* matching sequences for each region graph specification. This optimization applies to queries without window specifications and whose duration constraints are of the form “ i ” or “ $i=$ ”, which are the

<pre>CREATE VIEW g1_windowed AS (SELECT vid, fid, oid1, oid2, lead(fid, d1 - 1, 0) OVER (PARTITION BY vid, oid1, oid2 ORDER BY fid) as fid_offest FROM g1);</pre>	<pre>CREATE VIEW g1_iterated AS (SELECT vid, min(fid_offest) AS fid, oid1, oid2 FROM g1_windowed WHERE fid_offest = fid + (d1 - 1) GROUP BY vid, oid1, oid2);</pre>	<pre>CREATE VIEW g2_filtered AS (SELECT t1.vid, t2.fid, t1.oid1, t1.oid2 FROM g1_iterated t1, g2 t2 WHERE t1.vid = t2.vid AND t1.oid1 = t2.oid1 AND t1.oid2 = t2.oid2 AND t1.fid < t2.fid);</pre>
(a) Q1: window	(b) Q2: iterate	(c) Q3: filter

Figure 3.7: SQL snippets. EQUI-VOCAL generates efficient SQL queries by avoiding recursions and pruning intermediate results.

only types of queries that EQUI-VOCAL currently generates.

As a concrete example, consider Figure 3.6. Assume we want to execute a query $q = \text{Duration}(g_1, 2); \text{Duration}(g_2, 2); g_3$, where $g_1 = \text{Far}(o_1, o_2)$, $g_2 = \text{Near}(o_1, o_2)$ and $g_3 = \text{Behind}(o_1, o_2)$. Also, we only consider one permutation of objects for each video segment in this example for simplicity. The event that q looks for consists of a contiguous sequence of g_1 for at least $d_1 = 2$ frames, followed by a contiguous sequence of g_2 for at least $d_2 = 2$ frames, etc.

Consider video segment v_1 . For the first region graph g_1 , we only need to extract the earliest matching segment of g_1 , which is $e_{11} = \{g_{12}, g_{13}\}$. This is because, for every satisfying event for the query, we can replace its matching segment of g_1 with the earliest one and the resulting event will still be a match to the query. $e_2 = \{g_{13}, g_{14}, g_{25}, g_{26}, g_{37}\}$ is a matching event for q . By replacing $e_{12} = \{g_{13}, g_{14}\}$ with $e_{11} = \{g_{12}, g_{13}\}$, we get $e_1 = \{g_{12}, g_{13}, g_{25}, g_{26}, g_{37}\}$, which is still a matching event for q . For subsequent region graph g_2 , we first use the earliest matching segment from g_1 to retain matching frames of g_2 that are temporally after the end frame of the segment, which are g_{25} and g_{26} . Then, we can use the same method as before to find the earliest matching segment from g_2 , which is $e_{21} = \{g_{25}, g_{26}\}$. The same procedure can be applied to g_3 , giving us $e_{31} = \{g_{37}\}$. Thus, the earliest matching event for q in v_1 is e_1 . Video segment v_2 does not contain any matching event of q .

The above algorithm can be expressed using SQL queries, as shown in Figure 3.7. Q1

partitions the data on video ID and each unique permutation of groups of objects so that each group of objects is analyzed separately, and sorts the data on frame ID. Next, for each row, it gets the current frame ID as well as the frame ID that is d_1 frames after the current frame ID, or 0 if no such frame exists. The result of Q1 gives a table of intervals. Q2 then finds the earliest matching segment for each combination of video and groups of objects. Together, Q1 and Q2 find the earliest matching segment for $\text{Duration}(g_1, 2)$ without using recursive joins. Q3 uses the earliest matching segment from g_1 to determine where to look for matching segments in g_2 . The example SQL snippets only contain two objects `oid1` and `oid2`. In general, Q1 and Q2 partition and group the data by all objects in the region graph g_i , and Q3 specifies all the constraints on objects across two region graphs in the `where` clause. We demonstrate the impact of both optimizations in Section 3.4.5.

Caching. We implement an application-level cache to reuse sub-query results. EQUI-VOCAL generates SQL queries as described above to find matching region graphs in the sequence specified by the DSL query one by one. We cache the intermediate results of all prefixes on the set of video segments to allow other queries sharing the same sub-queries to reuse the results. For example, when executing the query $q = g_1; g_2; g_3$ on video segments v_1 and v_2 , we cache the results of $q_1 = g_1$, $q_2 = g_1; g_2$, and $q_3 = g_1; g_2; g_3$ on v_1 and v_2 . Later on, if a query $q' = g_1; g_2; g_4$ is executed on v_1 and v_3 , EQUI-VOCAL will reuse the cached results of $q_2 = g_1; g_2$ on v_1 to improve performance.

Bounding the search space. EQUI-VOCAL synthesizes queries up to a certain maximum size to ensure that the synthesis algorithm eventually terminates. Assume the number of user-defined functions for object classes and attributes is m_1 and the number of user-defined functions for relationships is m_2 . We bound the query search space using a set of hyperparameters: the maximum number of region graphs (n_g) in a query, the maximum number of predicates (n_p) in the query, the number of possible values a duration constraint can take (n_d), and the number of unique objects allowed in a query (n_v). The size of the search space is then $|Q| = O((n_g(n_v m_1 + n_v^2 m_2))^{n_p} \cdot n_d^{n_g})$.

Predicate evaluation. The current implementation of EQUI-VOCAL populates the

Table 3.3: Example queries written for the trajectories dataset. “Pos %” is the percentage of positive examples in the dataset.

ID	Query	Description	Pos %
TQ1	$(\text{Near}(o_1, o_2), \text{Bottom}(o_1))$	o_1 is close to o_2 while o_1 is at the bottom.	7.0%
TQ2	$\text{Far}(o_1, o_2); \text{Near}(o_1, o_2); \text{Far}(o_1, o_2)$	Two objects move from far to near, then to far.	7.5%
TQ3	$\text{Far}(o_1, o_2); (\text{Near}(o_1, o_2), \text{Behind}(o_1, o_2))$	o_1 and o_2 are far away, then they move close and o_1 is behind o_2 .	12%
TQ4	$\text{Far}(o_1, o_2); (\text{Near}(o_1, o_2), \text{Behind}(o_1, o_2), \text{Left}(o_1))$	o_1 and o_2 are far apart, then they move close and o_1 is behind o_2 and o_1 is on the left.	6.5%
TQ5	$(\text{FrontOf}(o_1, o_2), \text{Top}(o_1))$	o_1 is in front of o_2 while o_1 is at the top.	45%
TQ6	$\text{Near}(o_1, o_2); \text{Far}(o_1, o_2)$	Two objects move from close to far apart.	10%
TQ7	$(\text{Near}(o_1, o_2), \text{Left}(o_1), \text{Behind}(o_1, o_2))$	o_1 is close to and behind o_2 while o_1 is on the left.	8.5%
TQ8	$(\text{Far}(o_1, o_2), \text{Bottom}(o_1)); \text{Near}(o_1, o_2)$	o_1 at the bottom is far from o_2 , then they move close.	6.9%
TQ9	$(\text{Far}(o_1, o_2), \text{Left}(o_1)); (\text{Near}(o_1, o_2), \text{Left}(o_1))$	o_1 and o_2 move from far to close while o_1 is on the left.	10%
TQ1D	$\text{Duration}(\text{Far}(o_1, o_2), 5); \text{Near}(o_1, o_2); \text{Far}(o_1, o_2)$	Two objects are far apart for at least 5 frames, then they move close, then they are far again.	5.6%
TQ2D	$\text{Duration}(\text{LeftOf}(o_1, o_2), 5); (\text{Near}(o_1, o_2), \text{Top}(o_1)); \text{Duration}(\text{RightOf}(o_1, o_2), 5)$	o_1 is on the left of o_2 for at least 5 frames, then they move close and o_1 is at the top, then o_1 is on the right of o_2 for at least 5 frames.	6%
TQ3D	$\text{Duration}((\text{FrontOf}(o_1, o_2), \text{Left}(o_1)), 15); \text{Duration}((\text{Left}(o_1), \text{RightOf}(o_1, o_2), \text{Top}(o_1)), 5)$	o_1 is in front of o_2 while o_1 is on the left for at least 15 frames, then o_1 moves to the right of o_2 while o_1 is at the top left for at least 5 frames.	8.3%

Objects relation and the property attributes in the **Attributes** relation before query synthesis (see Table 3.1), by executing ML models on all video frames. The **Relationships** relation and the state attributes in the **Attributes** relation are computed lazily during query execution since they do not require ML models in our prototype, and are thus inexpensive to compute. In general, optimizing what information is computed before query synthesis and what is computed lazily is not the focus of this work.

To simplify SQL query generation, we implement all predicates (except for join predicates) in EQUI-VOCAL as user-defined functions that take attributes and bounding box coordinates as input and return boolean values. In our prototype, we use PostgreSQL and implement user-defined functions for state and relationship predicates that operate on bounding boxes, and for property attribute predicates that operate on attribute names.

Table 3.4: Example queries written for the real-world dataset. “# Pos ” is the number of positive examples in the dataset.

ID	Query	Description	# Pos
WQ1,2,3	(LaneA(o_1), LaneA(o_2), Near(o_1, o_2))	o_1 and o_2 are near and in the same lane.	164, 140, 228
WQ4,5,6	(LaneA(o_1), HighAccel(o_1)); (LaneA(o_2), HighAccel(o_2))	In lane 1, o_1 accelerates rapidly and then o_2 accelerates rapidly.	790, 495, 657
WQ7	(LaneA(o_1), LaneB(o_2)); (LaneB(o_1), LaneB(o_2))	o_1 is turning from lane A into lane B while o_2 is in lane B.	105
WQ8,9	(LaneA(o_1), LaneC(o_2)); (LaneB(o_1), LaneC(o_2))	o_1 merges from lane A into lane B while o_2 is in lane C (which is next to lane B).	492, 216
WQ1,2D	Duration((LaneA(o_1), LaneB(o_2), Near(o_1, o_2)), 5)	o_1 and o_2 are in adjacent lanes and close for at least 5 frames.	177, 435
WQ3,4D	Duration((LaneA(o_1), LaneB(o_2), Faster(o_1, o_2)), 5)	o_1 and o_2 are in adjacent lanes and o_1 is faster than o_2 for at least 5 frames.	471, 148

While EQUI-VOCAL provides the above user-defined functions, the user can also provide additional user-defined functions incrementally as they use the system, and they can be shared and reused across queries and users [209]. In practice, EQUI-VOCAL has to limit the number of user-defined functions to a reasonable amount to ensure the efficiency of query synthesis.

3.4 Evaluation

We conduct an experimental evaluation of EQUI-VOCAL. First, we show that on both synthetic and real-world datasets, compared to existing systems, EQUI-VOCAL reduces the query synthesis time by 1-2 orders of magnitude, achieves comparable or better F1 scores under the same labeling budget, is more robust to noisy data, and explores and executes queries more efficiently. Second, we show that EQUI-VOCAL is capable of synthesizing more complex, flexible queries over arbitrary scene graphs, which existing systems cannot handle. Third, We conduct a user study to demonstrate the effectiveness and usability of EQUI-VOCAL. We further compare EQUI-VOCAL with a machine learning method and conduct an ablation study by varying the various design choices outlined in the previous

sections.

Baselines. To the best of our knowledge, there are no existing systems that can synthesize queries over scene graphs. The most similar system to EQUI-VOCAL is Quivr [143], which synthesizes queries over trajectories. Since object trajectories can be represented using scene graphs, we compare our system against Quivr. We also compare against PATSQL [195], which is a state-of-the-art query-by-example system for relational data.

Predicates. For the synthetic dataset, we use six relationship predicates (**Near**, **Far**, **LeftOf**, **RightOf**, **FrontOf**, and **Behind**) and four state attribute predicates (**Left**, **Right**, **Top**, **Bottom**) on all benchmarks. On the scene graphs dataset, we additionally use thirteen property attribute predicates corresponding to eight colors, two materials, and three shapes (e.g., **Red**, **Rubber**, **Cube**). For duration constraints, we consider three possible values: $\text{duration}(g) \geq 5, 10, \text{ or } 15$ frames. Section 3.3 discussed how EQUI-VOCAL evaluates these predicates. For the real dataset, we use nine predicates: two relationship predicates (**Faster**, **Near**) and eight state predicates (**LaneK**, **Stopped**, **HighAccel**). **LaneK** detects whether a car is in lane K, and we identify five lanes in the video.

Data. We evaluate our system on both synthetic and real-world datasets. We first evaluate on the CLEVRER dataset [224], which comprises synthetic videos of moving objects. This dataset includes a variety of geometric shapes interacting in space and time, and comes with ground truth data, facilitating testing queries with varying complexities. We create two benchmarks from this data: trajectories and scene graphs datasets.

Trajectories dataset. We create a first dataset to test queries over trajectories, which baseline systems support. We extract 10,080 pairs of object trajectories that overlap in time from 500, 5-second video segments. Each trajectory pair is essentially a temporally ordered sequence of bounding box pairs $b = (x_{a1}, y_{a1}, x_{a2}, y_{a2}, x_{b1}, y_{b1}, x_{b2}, y_{b2})$ representing two objects in a video segment. We manually generate a set of queries with varying complexities (Table 3.3). To generate ground truth labels, we run each target query on the dataset. We sample 500 trajectory pairs as training data (i.e., data used during query synthesis) and use the rest as test data (i.e., to measure the quality of synthesized queries).

Table 3.5: Query synthesis time (median, in seconds) for each method to achieve at least 0.9 F1 scores. EQUI-VOCAL successfully learns all queries with at least 0.9 F1 scores and is faster than (or at least comparable to) the two baselines. (NA: not applicable, —: failed due to insufficient F1 score or timeout)

(a) Simplified tasks.

Method	TQ1	TQ2	TQ3	TQ4	TQ5	TQ6	TQ7	TQ8	TQ9
PATSQL	1.54	—	538	—	4.00	1.93	552	—	—
Quivr	10.5	5.77	27.9	94.2	13.5	4.36	47.7	16.7	—
Ours	3.57	1.85	17.1	46.6	4.36	1.66	18.5	22.7	20.6

(b) Normal tasks.

Method	TQ1	TQ2	TQ3	TQ4	TQ5	TQ6	TQ7	TQ8	TQ9	TQ1D	TQ2D	TQ3D
PATSQL	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Quivr	7228	7428	8185	—	7839	3683	—	8756	—	—	—	—
Ours	75.6	125	106	185	50.4	61.1	106	110	107	223	166	183

Scene graphs dataset. This more complex dataset contains scene graphs extracted from the CLEVRER dataset, which baseline systems do not support. We extract scene graphs from 10,000, 5-second video segments. For every frame of a video segment, we store the object track ID (*oid*), bounding box coordinates, and object attributes (shape, color, and material) of every object in the frame (the **Objects** and **Attributes** relations in Table 3.1). We automatically generate three classes of queries with different complexities: *easy*, *medium*, and *hard*. Each generated query contains exactly three variables (i.e., three distinct objects). Easy queries have exactly three predicates on relationships and states, one property attribute predicate, one region graph, and no duration constraints; medium queries have exactly five predicates on relationships and states, two property attribute predicates, three region graphs, and no duration constraints; hard queries have the same complexity as medium queries but also include duration constraints with three possible values. Each class contains 40 queries.

Real-world dataset. We test on a 50-minute traffic video from the YTStreams dataset [21]. The dataset comes with car trajectories, and we further extract the velocity and acceleration of each car trajectory as the object attributes, following the same procedure described

in [143]. From the video, We create 72,159 pairs of overlapping car trajectories, using half of them as training data and the rest as test data. As shown in table 3.4, we adopt 13 queries from [143] for our evaluation, with some adjustments, including the use of a different in-lane detection method, removal of real-valued parameters from predicates, and expression of queries using scene graphs, and the introduction of additional queries with two scene graphs. These queries capture a wide range of car behaviors.

Metrics. We report the F1 score and query synthesis time. We synthesize queries using the training set and report the F1 score of the query (or the median F1 score if there are multiple queries with the same best score on the training set) over the test set. Each experiment is run 20 times for the trajectories and real-world datasets and five times for the scene graphs dataset, and we report the median F1 score and query synthesis time over these runs.

Implementation details. We implement our prototype in Python with PostgreSQL as the backend. The source code for EQUI-VOCAL is available at <https://github.com/uwdb/EQUI-VOCAL>. We conduct all experiments except for the user study on a computing cluster with Intel Xeon Gold 6230R CPUs at 2.10GHz. When measuring runtimes, we request one node with one core and 100GB of RAM. Unless otherwise specified, we configure EQUI-VOCAL as follows. For the trajectories dataset and the real-world dataset, we search for queries with up to 5 predicates across up to 3 region graphs. We use beam width $bw = 10$, $\lambda = 0.01$ for regularization (with $\alpha_1 = \alpha_2 = 1$, and $\alpha_3 = 0.1$, see Section 3.2.7), and we set $k = 100$, $s_q=100$, and $s_v = 100$. For the scene graphs dataset, we search for queries with up to 7 predicates, 3 region graphs, and 3 objects. Since the dataset is more complex and challenging, we use a smaller λ (because the query complexity term, $R(q)$ has a greater absolute value) and a greater k . We use beam width $bw = 10$, $\lambda = 0.001$ for regularization (with $\alpha_1 = \alpha_2 = 1$, and $\alpha_3 = 0.1$), and we set $k = 1000$, $s_q=100$, and $s_v = 100$.

The user study is conducted on an AWS EC2 `c6id.4xlarge` instance with 16 vCPUs and 32GB of RAM. We search for queries with up to 7 predicates, 3 region graphs, and 3 objects. For duration constraints, we consider three possible values, 25, 50, and 75 frames, which translate to 1, 2, and 3 seconds. To improve the interactivity of the system, we use

a smaller beam width $bw = 5$ and a smaller sample of candidate queries $s_q = 25$ for active learning. We set $s_v = 100$, $\lambda = 0.001$ and $k = 100$.

We consider two variants of Quivr. As per the original paper, we limit the number of atomic predicates in Quivr’s queries to 5 and the depth of the nested constructs to 3, which leads to a similar search space as EQUI-VOCAL. When considering queries without duration constraints (e.g., TQ1-TQ9), we omit Kleene star operators from its search space. Otherwise (e.g., TQ1D-TQ3D), we include Kleene star in the query expansion and add one more predicate `MinLength $_{\theta}$` , which checks whether the duration of the input is at least θ frames. Quivr returns all queries that match the examples, so we select the queries with the simplest structure (which is determined by the number of atomic predicates, and, if the former is the same, by the depth) and report their median F1 score.

Because PATSQL requires that *all* user-provided input tables be used in the solution query, when comparing against PATSQL, we restrict all systems to only the candidate predicates that appear in the target query. Because PATSQL cannot handle large tables efficiently, we also downsample each trajectory by 75% (we keep one frame out of four) to reduce the size of the input tables. We refer to this configuration as *simplified tasks*.

3.4.1 Results against baselines on trajectories

We evaluate EQUI-VOCAL against the two baselines on the trajectories dataset and the set of queries in Table 3.3. For TQ1-TQ9, we omit duration constraints from the search space. We run each method as follows. For each target query, we randomly select 2 positive and 10 negative examples from the training set and use them as the input to the method. Each method then asks for b additional examples during the search. For PATSQL, since it is not interactive, we simply sample b more examples randomly from the remaining dataset and provide the $12 + b$ examples to the system at the beginning. For Quivr and EQUI-VOCAL, we input the 12 initial examples, and each system actively requests more examples during the search process.

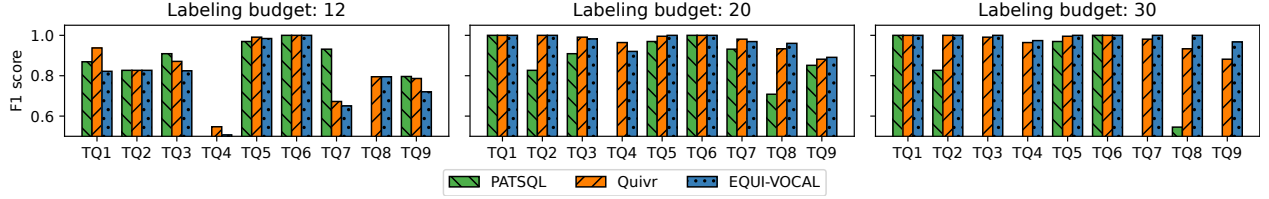


Figure 3.8: F1 score for queries without duration constraints on simplified tasks, under different user labeling budgets. PATSQL performs well only when the target query is simple. EQUI-VOCAL performs worse than Quivr when no additional examples are requested, but catches up and outperforms Quivr with a larger labeling budget.

EQUI-VOCAL is faster than baselines and can find high-performant queries even for complex queries. Table 3.5 shows the query synthesis time of each query to achieve at least a 0.9 F1 score. For simplified tasks, EQUI-VOCAL outperforms baselines on 6 queries. PATSQL performs the best or close to the best when queries are simple (TQ1, 5, and 6) because these queries include only two predicates and PATSQL terminates once it finds one solution query. Quivr is slower than EQUI-VOCAL but comparable to it except for TQ9, in which case Quivr fails due to an insufficient F1 score of the synthesized queries. Under the normal setting, EQUI-VOCAL is significantly more efficient than Quivr and can find high-performance queries in hundreds of seconds even for complex queries with duration constraints, while Quivr cannot synthesize TQ4, TQ7, TQ9, and TQ1D-TQ3D within 4 hours due to the large number of queries that need to be explored in the enumerative search.

On simplified tasks, EQUI-VOCAL and QUIVR find queries with similar F1 scores and outperform PATSQL for the same labeling budget. Figure 3.8 shows the F1 score of each system when varying the user labeling budget on the simplified tasks (which all systems can perform). We assign an F1 score of 0 if a system fails to find a query in 1 hour. Across all labeling budgets tested, as above, PATSQL performs well when the target query is simplest (TQ1, 5, and 6). For other queries, PATSQL either fails to find a solution query within 1 hour or the solution query has low performance (below a 0.9 F1 score). When the budget is 30, we observe a decrease in F1 scores for TQ3, 7,

Table 3.6: Probability that a system returns at least one query on noisy data. (false positive rate is 0.1 of false negative rate)

FN rate	0.1	0.2	0.3	0.4	0.5
Quivr, Mean (Range)	68% (20%-100%)	54% (35%-75%)	47% (5%-75%)	34% (5%-55%)	17% (0%-50%)
Ours, Mean	100%	100%	100%	100%	100%

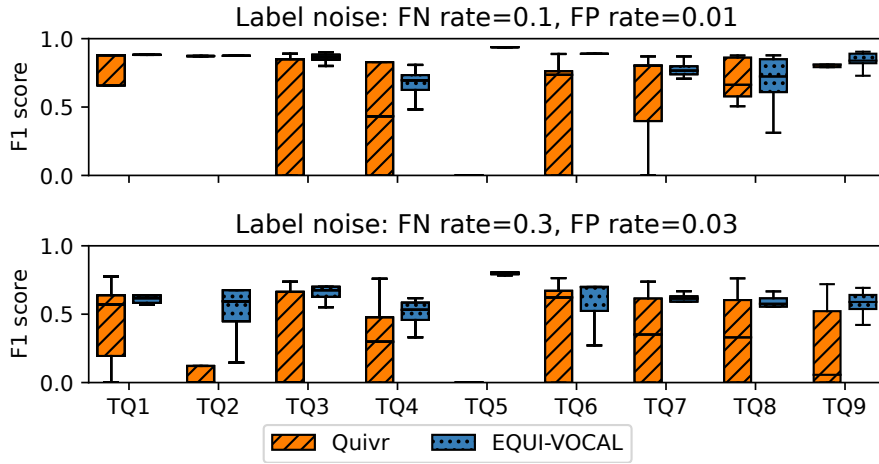


Figure 3.9: Impact of data noise, with a labeling budget of 20. EQUI-VOCAL constantly outperforms Quivr and is thus more robust to data noise.

8, and 9, because PATSQL has a less constrained search space than EQUI-VOCAL and does not scale well when the size of input and output tables increases as more examples are provided. Quivr performs better than EQUI-VOCAL when no additional examples are requested (budget=12) because it enumerates the entire search space and finds all consistent queries using the initial examples, while EQUI-VOCAL prunes candidate queries at every iteration to ensure efficient query synthesis. With a larger labeling budget, EQUI-VOCAL can request more labels during the search process to select better paths to explore, thus it catches up with Quivr and outperforms it when the budget is 30.

EQUI-VOCAL is more robust to noisy data and produces higher quality queries than Quivr. We compare the performance of Quivr and EQUI-VOCAL when

Table 3.7: Number of queries explored and number of predictions per second to achieve at least 0.9 F1 scores. EQUI-VOCAL reduces computational effort by exploring fewer queries and by executing queries faster.

ID	# queries explored	
	Quivr	Ours
TQ1	222748	1123
TQ2	245112	1071
TQ3	220627	1101
TQ5	212066	1118
TQ6	224504	1077
TQ8	263040	1080

ID	# predictions/s	
	Quivr	Ours
TQ1	154	574
TQ2	124	666
TQ3	140	664
TQ5	140	480
TQ6	151	547
TQ8	145	792

the data is noisy. We do not compare against PATSQL since its performance on video queries is already low even with perfect data. We inject noise into the original dataset by randomly flipping a fraction of the labels. We vary the false negative rate from 0.1 to 0.5 and set the false positive rate to 0.1 the false negative one. Table 3.6 shows the percentage of runs when the system returned *any* queries for different noise rates. We evaluate the systems over 9 queries (TQ1-TQ9) in the simplified setting. For Quivr, we also report the range of the success rate besides the mean value. When the false negative rate is 0.1, Quivr has a 68% success rate, and this goes down to only 17% when the false negative rate is 0.5. In contrast, EQUI-VOCAL always returns k queries. To demonstrate that EQUI-VOCAL also produces higher quality queries, Figure 3.9 shows the F1 score of EQUI-VOCAL and Quivr with a labeling budget of 20, under two different noise rates. When Quivr fails to return any queries, we assign an F1 score of 0. EQUI-VOCAL performs better than Quivr for different queries and different noise rates.

EQUI-VOCAL is more computationally efficient than Quivr by exploring fewer queries and executing every query faster. Table 3.7 measures the number of queries explored and the number of predictions per second to achieve at least a 0.9 F1 score. We define a prediction as evaluating a query on a video segment, so the number of predictions per second reflects the efficiency of query execution. We evaluate over TQ1-TQ9

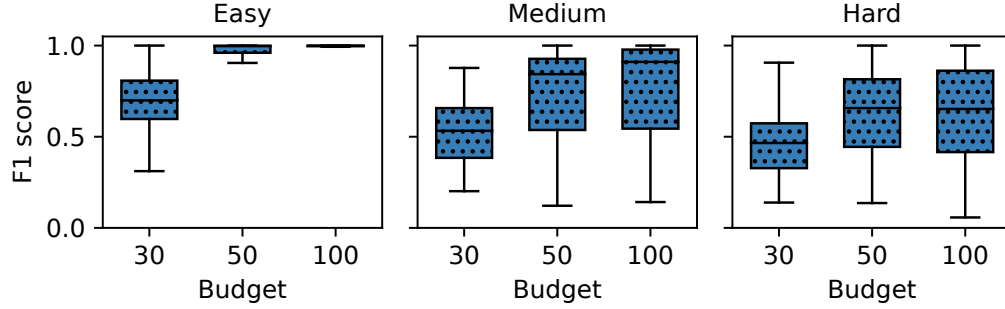


Figure 3.10: EQUI-VOCAL can synthesize high-quality queries within a reasonable budget, even for scene graph queries.

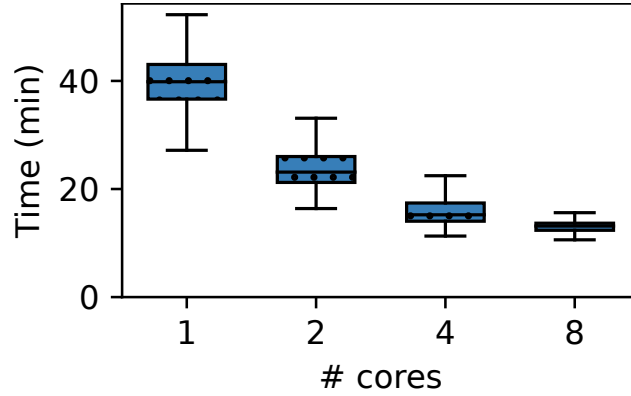


Figure 3.11: Vary # cores. EQUI-VOCAL can be easily parallelized to reduce synthesis time.

in the normal setting and report the numbers for queries that Quivr can synthesize. By limiting the query search to valid sequences of region graphs and by using the beam search strategy, the number of queries explored by EQUI-VOCAL is as small as 0.41% of Quivr. EQUI-VOCAL also executes queries faster than Quivr by up to $5.5\times$ in terms of the number of predictions per second. These two factors together make EQUI-VOCAL more efficient than Quivr.

3.4.2 Results on scene graphs

Next, we evaluate EQUI-VOCAL on the scene graphs dataset. Neither PATSQL nor Quivr support such flexible queries at video scale. Because the dataset is more complex, EQUI-VOCAL requires that the user provide a slightly larger (although still quite small) number of initial examples to avoid overfitting. We randomly select 15 positive and 15 negative examples from the training dataset as the initial examples. Later in Section 3.4.5, we discuss how varying the initial number of examples affects the system performance.

Figure 3.10 shows the F1 score of EQUI-VOCAL under different user labeling budgets. With a larger budget, EQUI-VOCAL can learn queries with higher F1 scores, and achieves a median F1 score of 1.0, 0.91, and 0.67 for easy, medium, and hard queries when the budget is 100. Results for a budget of 50 are nearly identical, which shows that **EQUI-VOCAL can synthesize high-quality queries within a reasonable budget, even for complex queries.**

EQUI-VOCAL does not learn good queries in all cases. We note that some queries are easier to synthesize than others, which results in the high variance of the F1 score in Figure 3.10. In particular, EQUI-VOCAL assumes that the ancestor queries of the target query are informative and leverages the performance of those queries to guide the search. However, if the ancestor queries cannot be distinguished from other queries in the same iteration (because they all have the same, typically low, F1 score), EQUI-VOCAL struggles with learning good queries. One direction of future work could explore other types of more fine-grained user feedback besides binary labels to help EQUI-VOCAL learn intermediate queries [178].

EQUI-VOCAL’s query synthesis time on the scene graphs dataset is greater than on the trajectories dataset. It increases from tens of seconds (Table 3.5) to minutes. **However, EQUI-VOCAL can easily be parallelized to reduce synthesis time, as executing queries in PostgreSQL is embarrassingly parallel.** Figure 3.11 shows the query synthesis time for the hard, scene-graphs queries when varying the number of cores and for a labeling budget of 50. Query synthesis time decreases significantly by using multiple cores and reduces to 13 minutes with eight cores. EQUI-VOCAL achieves $1.76\times$ speedup with

Table 3.8: Quivr’s success rate on the real-world dataset.

Query	WQ1	WQ4	WQ7	WQ8	WQ1D	WQ3D
Quivr success rate	40%	55%	35%	60%	5%	5%

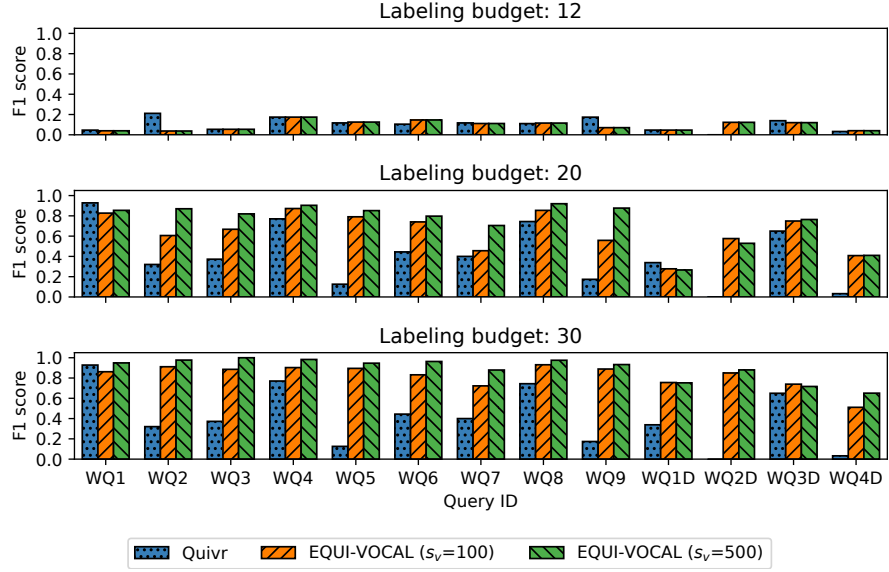
Table 3.9: Queries used in the user study. “Pos %” is the percentage of positive examples in the dataset.

ID	Query	Description	Pos %
UQ1	(Red(o_1), Cylinder(o_2), Far(o_1, o_2)); Near(o_1, o_2)	A red object is far from a cylinder, then they get close.	16.7%
UQ2	(Purple(o_1), Metal(o_1), Behind(o_1, o_2), Bottom(o_2))	A purple metal object o_1 is behind another object o_2 at the bottom of the screen.	24.2%
UQ3	(Red(o_1), Cylinder(o_2), Far(o_1, o_2)); (Near(o_1, o_2), Top(o_3), Right(o_3))	A red object is far from a cylinder, then they get close while a third object is at the top right of the screen.	12.8%
UQ4	(Purple(o_1), Metal(o_1), Behind(o_1, o_2), Bottom(o_2)); Top(o_2)	A purple metal object o_1 is behind another object o_2 at the bottom of the screen, then o_2 moves to the top.	12.8%
UQ5	Duration((Red(o_1), Cylinder(o_2), Far(o_1, o_2), 25), (Near(o_1, o_2), Top(o_3), Right(o_3)))	A red object is far from a cylinder for at least a second, then they are near each other while a third object is at the top right of the screen.	4.2%
UQ6	(Purple(o_1), Metal(o_1), Behind(o_1, o_2), Bottom(o_2)); Duration((Bottom(o_3), Right(o_3)), 25)	A purple metal object o_1 is behind another object o_2 at the bottom of the screen, then o_2 moves to the top, then a third object o_3 is at the bottom right of the screen for at least a second.	3.8%

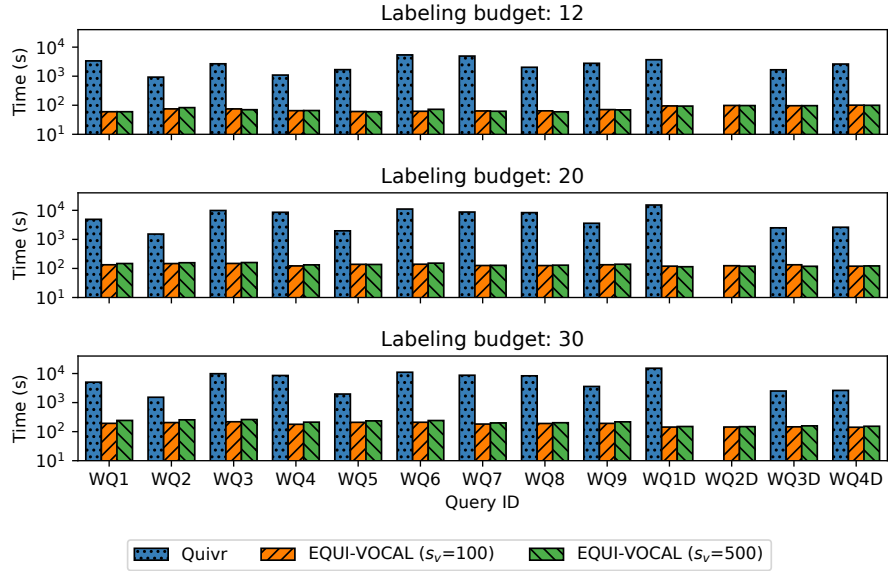
2 cores and $3.03\times$ speedup with 8 cores. Note that this is the largest labeling budget for our experiments. With a lower budget, EQUI-VOCAL is even faster. The query synthesis time can be further reduced by using more cores, further decreasing the labeling budget, decreasing bw , or decreasing k .

3.4.3 Results on real-world dataset

We compare EQUI-VOCAL and Quivr on the real-world dataset and queries in Table 3.4. For WQ1-WQ9, we omit duration constraints from the search space. We randomly select 2 positive and 10 negative examples as initial examples. Since the target events are extremely rare in the dataset, we consider EQUI-VOCAL with two different s_v values, 100 and 500, to



(a) F1 scores.



(b) Query synthesis time.

Figure 3.12: F1 scores and query synthesis time of EQUI-VOCAL and Quivr on the real-world dataset. EQUI-VOCAL can synthesize queries with high performance and is significantly faster than Quivr.

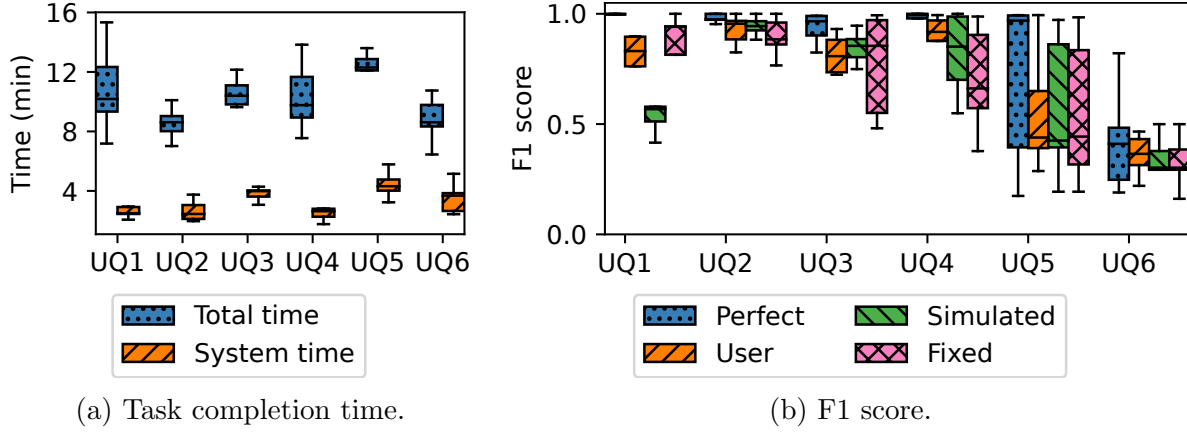


Figure 3.13: User study results. (a) Participants can complete the task in a reasonable time. (b) Participants can help EQUI-VOCAL synthesize UQ1-UQ4 with at least 0.8 F1 scores.

increase the number of sampled positive examples during active learning.

EQUI-VOCAL outperforms Quivr in terms of F1 scores and query synthesis time on the real-world dataset. Figure 3.12 shows the F1 score and query synthesis time of both systems under different labeling budgets. The timeout is set to 4 hours for both systems. EQUI-VOCAL can synthesize all queries in hundreds of seconds with a labeling budget of 30, while Quivr suffers from a low success rate (as shown in Table 3.8) and, even when it succeeds, takes thousands of seconds to synthesize queries. EQUI-VOCAL performs worse or similar to Quivr with a labeling budget of 12, but with some additional examples, EQUI-VOCAL can easily achieve higher F1 scores. In addition, sampling more candidate video segments during active learning ($s_v=500$) increases the F1 score in most cases but also slightly increases the synthesis time.

3.4.4 User study

We conduct a user study to understand the effectiveness and usability of EQUI-VOCAL. We use internal validation since asking participants to provide their own queries would make it difficult to compare across participants. Our goals include observing the system performance

Table 3.10: Labeling error rates of participants in the user study.

Error type	UQ1	UQ2	UQ3	UQ4	UQ5	UQ6
FP rate	0.250	0.065	0.169	0.028	0.308	0.029
FN rate	0.061	0.123	0.091	0.133	0.286	0.714

in the presence of user noise, measure task completion time, and collect qualitative feedback. We recruited 18 university students studying Computer Science. The study is conducted over the scene graphs dataset with six query tasks with different complexities, as shown in Table 3.9.

We started each session by explaining our definitions of predicates and walking through an example task with participants to get them familiar with the interface and the task. Each participant then completed three query tasks in sequence (either UQ1, 3, 5 or UQ2, 4, 6) with increasing complexity from easy, to medium, to hard. Each query task described the target complex event using natural language. In the user study, we provided the initial examples with ground-truth labels to EQUI-VOCAL so that participants could focus on interacting with the system during query synthesis. We randomly selected 10 positive and 10 negative examples for UQ1, 2 and 4 as the initial examples, and 15 for other queries to account for the increased query complexity. EQUI-VOCAL iteratively selected video segments to label, with a budget of 50 per task. After each task, participants reviewed the synthesized queries and provided subjective ratings and qualitative feedback via a questionnaire.

Participants are able to complete the task in a reasonable amount of time.

Figure 3.13a shows the task completion time of each query. The median completion time of each task ranges between 8.6 and 12.3 minutes. The average time to label a video segment for each query task is 22, 16, 25, 20, 30, and 22 seconds, respectively. Also, the system time only takes a small portion of the total time, suggesting that EQUI-VOCAL is efficient.

EQUI-VOCAL is resilient to user noise, whereas Quivr is not. Figure 3.13b shows the F1 score of synthesized queries under different labeling scenarios: using the ground-truth labels (**Perfect**), using real user labels (**User**), using injected label noise with a false

negative rate of 0.3 and a false positive rate of 0.03 (**Fixed**), and using injected label noise with the same noise rate as users for each query task (**Simulated**). Participants’ labeling error rates are shown in Table 3.10. Each experiment is run nine times. **F1 scores of User lie between Perfect and Simulated.** Due to the user noise, the query performance decreases compared to using perfect labels. Because the same error rate is used, **Simulated’s** performance is worse or similar to **User’s**, which justifies our expectations for the simulation experiment. **For queries that EQUI-VOCAL can synthesize with high quality, participants can help to find them with at least 0.8 F1 scores.** Not surprisingly, for queries (UQ5 & 6) that EQUI-VOCAL would fail even with perfect labels, participants cannot instruct the system to achieve better performance using the current prototype. Interestingly, most video segments that EQUI-VOCAL selects for hard queries are negative examples, making it difficult to learn good queries. However, these segments are not equally negative, as many participants consider some negative samples to be almost positive that fail to satisfy one predicate. This suggests that EQUI-VOCAL could potentially benefit from more fine-grained user feedback.

Participants would like EQUI-VOCAL to be more responsive. In the questionnaire, the average rating of system responsiveness is 3.53 out of 5. Although the average system time across all tasks is only 195 seconds, participants observe greater latency in early and middle iterations, since there are more branches to expand and more candidate queries to evaluate. Taking UQ1 as an example, the average wait time to receive the first video segment in the 5th iteration is 31 seconds, while in the 10th iteration, it is only 1.3 seconds. Other techniques can be used to further hide the latency of the system [53], which is not the focus of this work.

3.4.5 Ablation study

We study the impact of hyperparameters and the sensitivity of EQUI-VOCAL to hyperparameter tuning. Unless otherwise specified, we evaluate EQUI-VOCAL on the scene

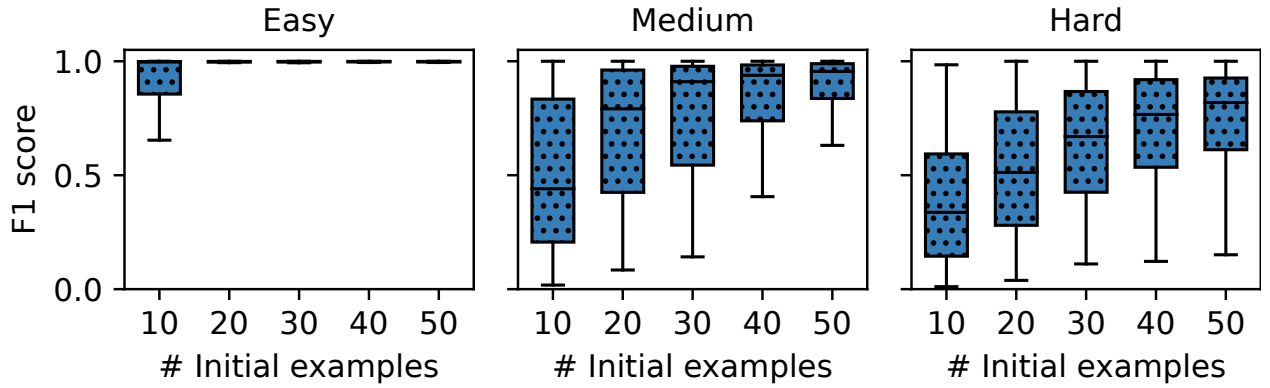


Figure 3.14: Providing more initial examples helps EQUI-VOCAL learn better queries and avoid overfitting.

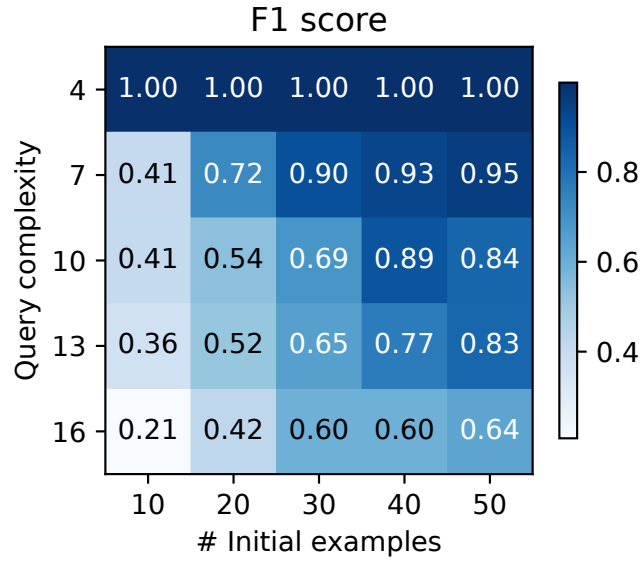


Figure 3.15: F1 score for various numbers of initial examples and queries with different complexity. EQUI-VOCAL obtains higher F1 scores with more initial examples and when the target queries are simpler.

graphs dataset and use the default values for hyperparameters that we are not studying.

EQUI-VOCAL synthesizes useful queries with a small number of initial examples, but providing more initial examples improves performance. Figure 3.14 shows EQUI-VOCAL’s F1 score for different numbers of initial examples, from five positives and

five negatives (ten in total), to 25 positives and 25 negatives (50 in total), while the total labeling budget remains 100. When the number of initial examples is smallest, EQUI-VOCAL can incorrectly retain queries that overfit the examples in early iterations, leading to final results with low F1 scores. Results, however, quickly improve, and with as few as 30 initial examples, median F1 scores are high at 0.998, 0.911, and 0.670 for the easy, medium, and hard queries respectively. Figure 3.15 shows the F1 score of EQUI-VOCAL under different numbers of initial examples and queries with different complexity, while the total labeling budget remains 100. As expected, EQUI-VOCAL obtains higher F1 scores with more initial examples and when the target queries are simpler. The heatmap aggregates the results of all easy, medium, and hard queries, and we use $\alpha_1 = 1, \alpha_2 = 0.9, \alpha_3 = 0.2$ when computing the query complexity for better visualization. Because manually finding initial examples requires more user effort than labeling video segments selected by EQUI-VOCAL, our approach is to balance user effort and system’s performance, using 30 initial examples in all other experiments on the scene graphs dataset. Interestingly, since EQUI-VOCAL can produce good queries in many cases even with a few examples, users always have the option to first start with a small set and restart with more examples if needed.

Active learning helps EQUI-VOCAL learn better queries quicker than randomly sampling video segments to label. Figure 3.16 shows EQUI-VOCAL’s F1 score when either using active learning or randomly selecting additional examples during query synthesis. We set the labeling budget to 50, and we vary the number of initial examples. The improvement of active learning over random sampling is more significant when the number of initial examples is larger since fewer labels are requested per iteration and the selection of informative examples becomes more important.

Increasing the beam width improves performance, but also increases the synthesis time. Figure 3.17 shows the F1 score of EQUI-VOCAL under different values of bw . Increasing bw from 1 to 10 increases median F1 scores on easy queries by 3.10%, on medium queries by 23.8%, and on hard queries by 34.3%, but increasing from 10 to 20 only further improves median scores by up to 3.07%. At the same time, increasing bw from 10 to 15

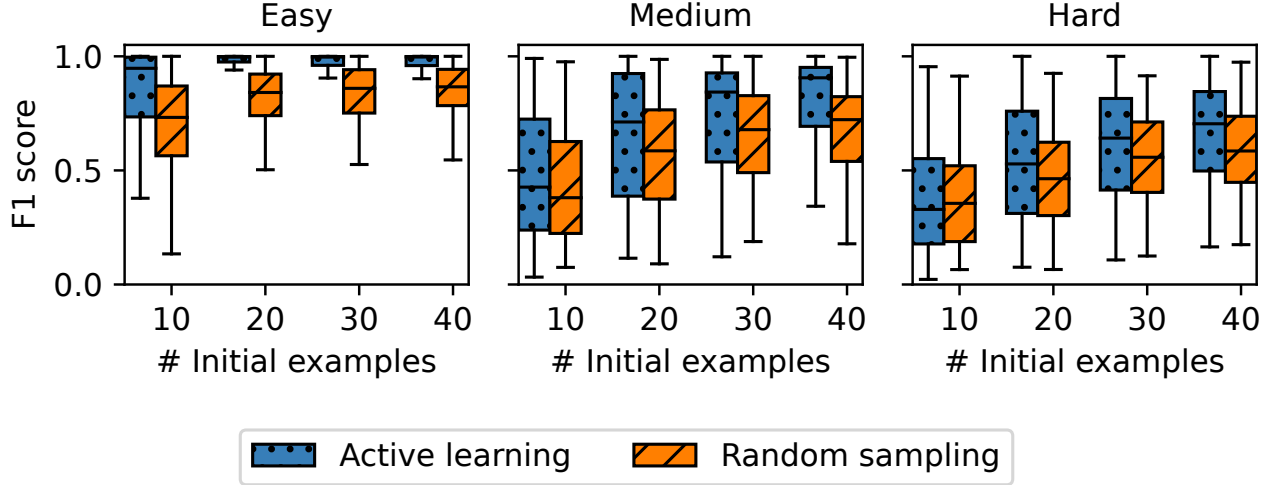


Figure 3.16: Active learning helps EQUI-VOCAL learn better queries compared to random sampling, and the improvement is more significant when the number of initial examples is larger.

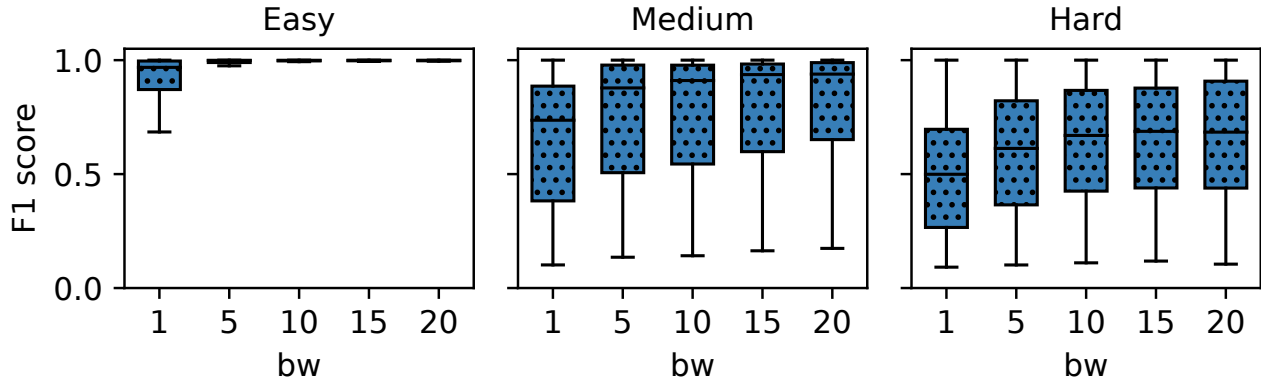


Figure 3.17: Increasing bw improves the performance of EQUI-VOCAL.

increases the median synthesis time on hard queries by 27.8%. EQUI-VOCAL defaults to $bw = 10$ to strike a balance between query performance and synthesis time.

Regularization helps EQUI-VOCAL learn better queries in many cases and does not harm performance in other cases. Figure 3.18 shows the F1 score of EQUI-VOCAL on both the trajectories dataset and the scene graphs dataset, under different values of λ . For the trajectory pairs dataset, we set the labeling budget to 20 and the initial

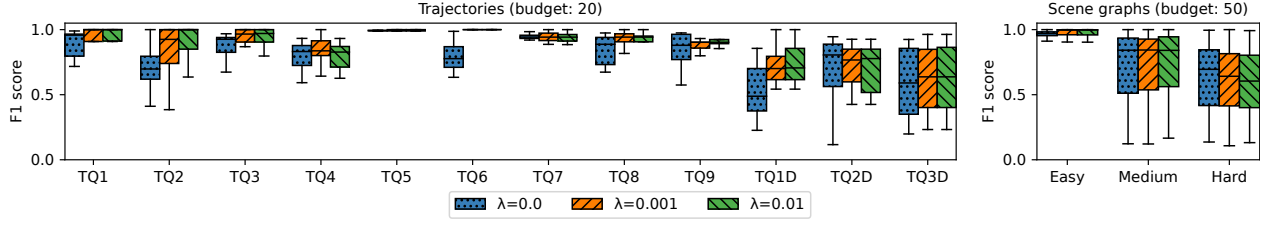


Figure 3.18: Regularization ($\lambda = 0.001$ and $\lambda = 0.01$) helps EQUI-VOCAL to learn better queries in many cases and does not harm the performance in other cases.

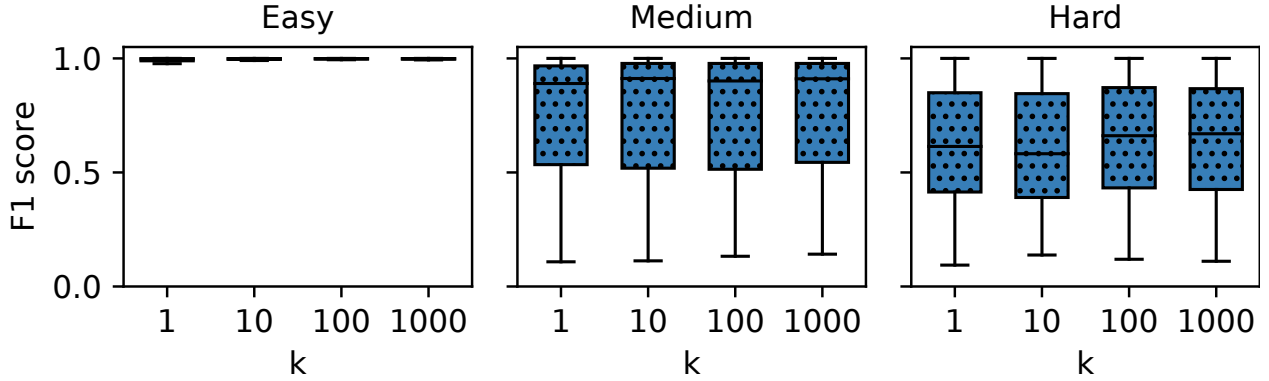
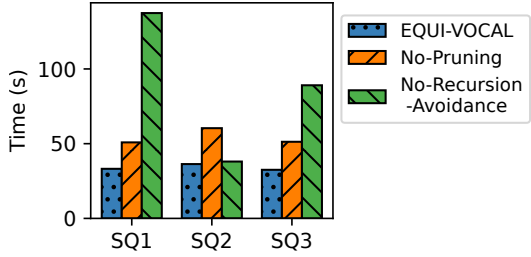


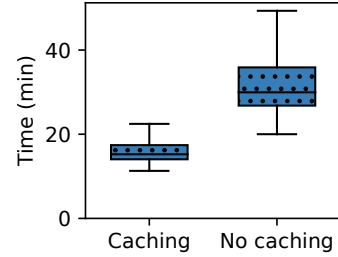
Figure 3.19: Increasing k slightly improves the performance of EQUI-VOCAL.

number of examples to 12 (i.e., two positive and ten negative examples); for the scene graphs dataset, we set the labeling budget to 50 and the initial number of examples to 30. For the trajectories dataset, we see an improvement for many queries with regularization, especially when the queries are simple (e.g., with fewer predicates). With $\lambda = 0.01$, we see a median improvement between 44.6% and -3.44% for the queries from Table 3.3. TQ4, TQ9, TQ2D, and TQ3D are more complex than other queries and have four predicates, so applying regularization is unlikely to improve the performance. For the more complex scene graphs dataset, regularization does not improve F1 scores but also does not compromise system performance. With $\lambda = 0.001$, the median change in F1 score stays between 2.84% and -7.69% . For this reason, EQUI-VOCAL defaults to using a small regularization factor.

increasing k slightly improves performance. We evaluate the F1 score of EQUI-



(a) Single query execution time with different SQL optimizations.



(b) Synthesis time with and without caching.

Figure 3.20: Impact of query execution optimizations. EQUI-VOCAL reduces query synthesis time by (a) generating efficient SQL queries that prune intermediate results and avoid recursion, and (b) using a caching mechanism.

VOCAL under different values of k , as shown in Figure 3.19. For medium and hard scene graph queries, increasing k from 1 to 1000 slightly improves the median F1 score by 2.36% and 9.12% respectively. Since the target queries are complex, they are not explored until later iterations of the algorithm, when most labeling budget has been used. Labels requested afterward have little impact on the ranking of the queries in the list, so the performance improvement from increasing k is also limited. Easy queries already have near-perfect F1 scores, so increasing k does not improve performance.

EQUI-VOCAL’s query execution optimizations significantly reduce query synthesis times. Figure 3.20a shows the average *query execution time* for variants of our query translation algorithm on scene graph queries. We generate SQL queries using our complete query translation algorithm, and two variants that omit optimizations of pruning intermediate results and avoiding recursion, respectively, and then execute them over all 10,000 video segments from the scene graphs dataset in PostgreSQL. We evaluate EQUI-VOCAL on SQ1-SQ3, and each query is executed 20 times. As the figure shows, the query execution time increases dramatically without either optimization. Pruning intermediate results boosts performance by up to 1.66 $\times$. Recursion avoidance is important when the query contains duration constraints (SQ1 and SQ3), providing up to a 4.14 $\times$ speedup.

Table 3.11: Median F1 scores of EQUI-VOCAL and ML methods. EQUI-VOCAL achieves higher F1 scores for all queries.

Queries	Ours	CLIP	MViT	CLIP (all)	MViT (all)
Easy	0.997	0.249	0.222	0.327	0.322
Medium	0.843	0.201	0.182	0.278	0.255
Hard	0.638	0.219	0.203	0.286	0.265

Figure 3.20b shows the impact of our caching mechanism. Caching query results leads to a $1.97\times$ speedup. We experimented on hard queries using four CPUs and a labeling budget of 50.

3.4.6 Comparing with machine learning method

To determine whether a video segment contains an event of interest, EQUI-VOCAL’s query synthesis approach could be substituted by training an ML model. However, this approach requires a large number of labeled examples and complicates interpretability [63]. We compare EQUI-VOCAL against an existing ML approach that builds domain-specific models for videos [53] on the scene graphs dataset. For EQUI-VOCAL, we adopt the same setting as outlined in Section 3.4.2, with a labeling budget of 50. The ML approach extracts features using pretrained video and image models and trains a linear model using the same set of user-labeled video segments selected by EQUI-VOCAL. We use both MViT [62] and CLIP [166] as the feature extractors. Table 3.11 shows that **EQUI-VOCAL achieves higher F1 scores than the ML approach for all queries, even when using CLIP with all 500 training samples.**

3.5 Summary

In this chapter, we presented EQUI-VOCAL, a new system that synthesizes compositional queries from examples. EQUI-VOCAL models compositional events as spatio-temporal scene graphs, explores the query search space using results of executing intermediate queries and

beam search, leverages active learning to reduce user effort, and generates efficient SQL queries to reduce computational effort.

Chapter 4

VOCAL-UDF: SELF-ENHANCING VIDEO DATA MANAGEMENT SYSTEM FOR COMPOSITIONAL EVENTS WITH LARGE LANGUAGE MODELS

In this chapter, we present VOCAL-UDF, a *self-enhancing* video data management system that enables users to issue compositional queries in natural language over video data, even when the modules required to answer those queries are not predefined in the system.

As we discussed in Chapter 1, VDBMSs answer compositional queries by decomposing them into granular subtasks [13, 75, 194, 188, 67, 23, 235, 143]. Submodules identify and track objects, attributes, and relationships across frames, populating spatio-temporal scene graphs [113, 92] that the system then composes to answer the query [132]. These systems make a critical assumption: that suitable modules already exist for every subtask in a query. In practice, users frequently encounter domain-specific applications, fine-grained object classes, or subjective concepts for which no off-the-shelf model exists, and filling those gaps manually is tedious—each missing module may require its own curated dataset and extensive training to achieve satisfactory performance [13]. Frontier VLMs offer broader semantic coverage but remain limited on compositional [137, 206] and temporal [228] reasoning, and their direct application at video scale is prohibitively slow and expensive [163, 202, 16, 3]. EQUI-VOCAL (Chapter 3) reduces the burden of writing declarative queries by synthesizing them from examples, but it still relies on the user to supply UDFs for any predicates that the system does not already implement.

To address these challenges, we present VOCAL-UDF (Figure 4.1), a *self-enhancing* VDBMS that empowers users to flexibly issue and answer compositional queries, even when

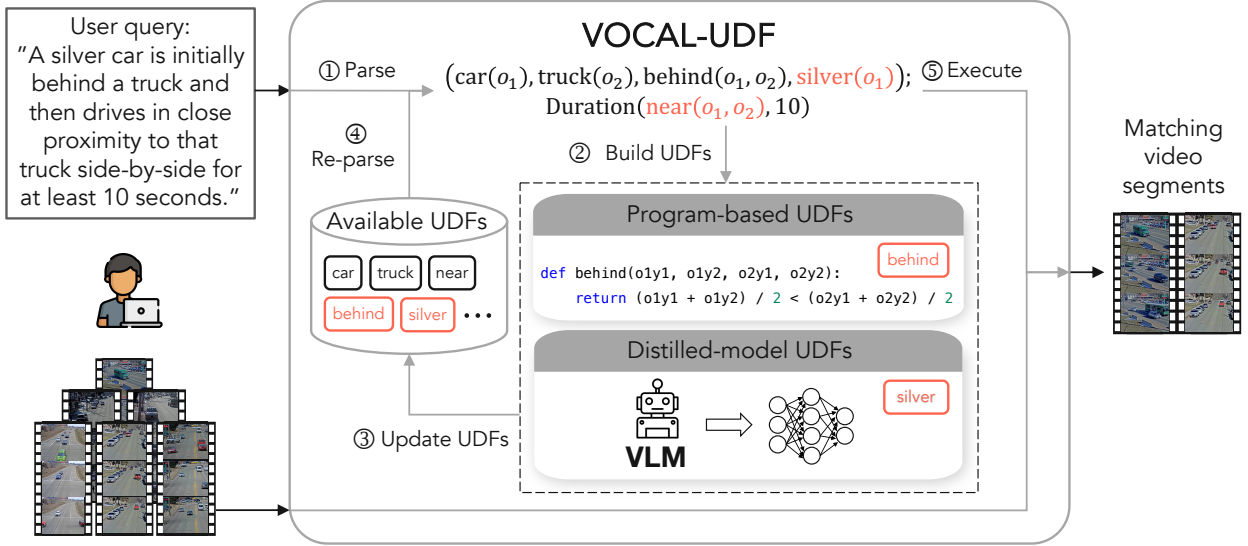


Figure 4.1: Given a video dataset and a user query in natural language, VOCAL-UDF ① parses the query into a DSL notation. If the query contains predicates that existing UDFs cannot answer, VOCAL-UDF ② automatically builds new UDFs, ③ updates its available UDF list, ④ reparses the query, and ⑤ executes the query to return matching video segments. VOCAL-UDF supports both program-based UDFs (i.e., Python functions) and distilled-model UDFs (i.e., ML models) for diverse concepts.

the necessary modules are unavailable. To use VOCAL-UDF, a user only needs to provide a video dataset and a natural language (NL) description of the query. VOCAL-UDF then *automatically* identifies and builds the necessary modules and encapsulates them as new UDFs to expand its querying capabilities. It then compiles the NL query into a declarative one that it executes over the video dataset.

There are several challenges in building VOCAL-UDF. First, given an NL query, VOCAL-UDF converts it into a declarative one that it can execute. Unfortunately, existing methods assume the existence of predefined modules that can be invoked to construct the declarative query [235, 52]. VOCAL-UDF addresses this challenge by identifying any semantic concepts that are not supported by existing modules or UDFs during compilation and leveraging an LLM’s reasoning to determine when and which new UDFs are needed.

The second challenge involves transforming the various identified missing semantic con-

cepts into executable modules. While LLMs can produce useful code in various contexts [38, 15, 223, 46], queries on video data are often highly ambiguous and their performance varies significantly (see Section 4.5.3). To address the challenge of handling a multitude of missing concepts, VOCAL-UDF automatically implements two types of UDFs—*program-based UDFs* and *distilled-model UDFs*. In our system, program-based UDFs are imperative Python functions that operate on relational tables and video pixels. While this class of UDFs can classify many relationships and attributes with high quality [235, 112], we show in Section 4.3.3 the need for distilled-model UDFs, which are lightweight machine learning (ML) models that are trained on the fly to classify more nuanced concepts [196]. VOCAL-UDF leverages LLMs for both UDF types: program-based UDFs harness an LLM’s programming capability, whereas distilled-model UDFs rely on the LLM’s ability to annotate and convert visual concepts into ML models [196]. To mitigate potential erroneous LLM-generated UDFs, VOCAL-UDF implements syntax and semantic verification steps.

The third challenge involves managing the inherent ambiguity in user intent when articulating a query. This is especially important for specialized and subjective concepts that are difficult to resolve without user feedback. For example, users may have different interpretations of the “near” relationship. Further compounding this difficulty is our observation that for some inputs a program-based UDF might perform better than a distilled-model UDF, while in other cases the opposite is true. Finally, for the most challenging semantic concepts, *no* UDF may perform well and employing one might risk overall query performance. To address this formidable combination of challenges, VOCAL-UDF generates *multiple* candidate UDFs each with different semantic interpretations, implementations, and properties. VOCAL-UDF then employs active learning to efficiently identify the variant that best matches the user’s intent.

Finally, the self-building nature of our system necessitates a consistent and unified UDF model. Typically, a UDF can be an arbitrary function that operates on database tuples. However, one challenge lies in modeling UDFs in a structured way to handle concepts of objects, relationships, and attributes while seamlessly integrating and interacting with var-

Table 4.1: Relational views for query execution with UDFs.

ObjView (vid, video_pixels)
RelView (vid, fid, pixels, rid, o1_o2_rnames, o2_o1_rnames, o1_oid, o1_x1, o1_y1, o1_x2, o1_y2, o1_anames, o2_oid, o2_x1, o2_x2, o2_y1, o2_y2, o2_anames)
AttrView (vid, fid, pixels, oid, oname, x1, y1, x2, y2, anames)

ious system components. To address this, we propose a unified UDF scheme for different semantic concepts, which enables the LLM to generate UDFs in a structured format and simplifies the compilation process. By incrementally growing the database, future UDFs can be composed using existing ones.

We evaluate VOCAL-UDF on three video datasets from different domains [224, 64, 92] and show that it significantly improves query performance, in terms of F1 score, by automatically selecting, implementing, and executing its automatically-generated UDFs. We finally conduct a thorough analysis of VOCAL-UDF to understand its efficiency in terms of execution time and monetary cost.

Overall, VOCAL-UDF’s self-enhancing capability is an important step toward making VDBMSs more practical to deploy and use in a variety of applications.

4.1 Background

VOCAL-UDF leverages EQUI-VOCAL’s [235] scene graph data model and query language, which models compositional video events as spatio-temporal scene graphs. This approach draws from cognitive foundations in human perception [230, 117, 24] and enables a variety of compositional queries [235]. This section summarizes key background information about these concepts.

Data model. In VOCAL-UDF, each video comprises a series of N frames $\{f_1, \dots, f_N\}$. The visual content of each frame is represented by a *scene graph* $g_i = (\mathbf{o}_i, \mathbf{r}_i)$, capturing all *objects* $\mathbf{o}_i$ and all *relationships* $\mathbf{r}_i$ between those objects within the frame at some time.

Listing 4.1: An example of the query language.

```

g1(vid, fid, fid, oid1, oid2) :- Objects(vid, fid, oid1, 'car', _, _, _, _),
    Objects(vid, fid, oid2, 'truck', _, _, _, _),
    Relationships(vid, fid, _, oid1, 'far', oid2), oid1 != oid2.
g2(vid, fid, oid1, oid2) :- Relationships(vid, fid, _, oid1, 'near', oid2), oid1 !=
    oid2.
g2_star(vid, fid, fid, oid1, oid2) :- g2(vid, fid, oid1, oid2).
g2_star(vid, fid_start, fid_end, oid1, oid2) :- g2_star(vid, fid_start, fid, oid1,
    oid2),
    g2(vid, fid_end, oid1, oid2), fid_end = fid + 1.
q(vid) :- g1(vid, fid11, fid12, oid1, oid2), g2_star(vid, fid21, fid22, oid1, oid2),
    fid21 > fid12, fid22 - fid21 + 1 > 10 * 24. %

```

Objects may also possess *attributes*. While a relationship links two objects, an attribute is attached to one object. A *region graph* g_{ij} is a subgraph of g_i , i.e., $g_{ij} \subseteq g_i$, that contains information critical for identifying an event. Finally, an *event* e is a sequence of region graphs $e = \{g_1, \dots, g_k\}$, where region graphs with a smaller index occur earlier in time than those with a larger index, but they do not need to be contiguous or distinct. The relational schema in Table 3.1 captures the scene graphs data model.

Query language. In VOCAL-UDF, a query identifies video segments that match a user-specified event. VOCAL-UDF supports relational queries over the schema in Table 3.1, which are of the following form. Using Datalog notation: $q(vid) :- g_1, \dots, g_k, \mathbf{p}, \mathbf{d}, w$, where $g_1, \dots, g_k$ is a temporally ordered sequence of region graphs specifying that a matching event consists of g_1 , followed by g_2 , followed by g_3 , etc. Each g_i can persist for multiple frames and there can be other frames between g_i and g_{i+1} . $\mathbf{p}$ is a set of predicates that are applied to objects, relationships, and attributes in region graphs. $\mathbf{d}$ is a set of constraints on the duration for which a region graph must remain valid before transitioning to the next one. Lastly, w is the maximum number of frames between g_1 and g_k . As an example, the event “A car is initially far from a truck, then remains close to the truck for more than 10 seconds” can be expressed as in Listing 4.1.

DSL. VOCAL-UDF adopts EQUI-VOCAL’s domain-specific language (DSL) [235], which encapsulates query logic while abstracting away SQL details. The query executor compiles these DSL queries into efficient SQL for execution over relational tables.

Listing 4.2: SQL query to identify frames where a silver car is behind a truck.

```
SELECT DISTINCT f.vid, f.fid, o1.oid, o2.oid
FROM frames f, objects AS o1, objects AS o2
WHERE f.vid = o1.vid AND f.fid = o1.fid AND o1.vid = o2.vid AND o1.fid = o2.fid
      AND o1.oid <> o2.oid AND car(o1.oname) = TRUE AND truck(o2.oname) = TRUE
      AND behind(o1.y1, o1.y2, o2.y1, o2.y2) = TRUE
      AND silver(f.pixels, o1.x1, o1.y1, o1.x2, o1.y2) = TRUE
```

Using the DSL, the same event can be expressed as (assuming 24 frames per second): $(\text{Car}(o_1), \text{truck}(o_2), \text{far}(o_1, o_2); \text{Duration}(\text{near}(o_1, o_2), 240))$. In this DSL, the *variable* o represents an arbitrary object in a query, with distinct subscripts indicating objects with different *oid*'s. All predicates in a region graph are separated by commas. Region graphs are then sequenced in temporal order using semicolons. Each region graph can persist for multiple frames and there can be other frames between two adjacent region graphs. Finally, $\text{Duration}(g, d)$ stipulates that the region graph g exists in *at least* d consecutive frames. The query returns a set of video segment identifiers.

4.2 A UDF-based data model

In this section, we formalize the types of UDFs supported by VOCAL-UDF. These UDFs allow users to define custom objects, relationships, and attributes. We then describe how UDFs are compiled and executed in VOCAL-UDF.

Table 3.1 shows VOCAL-UDF's relational schema. The **Frames** relation includes a virtual **pixels** column that stores the frame pixel values in a 3D array ($H \times W \times 3$, where H is height, W is width, and 3 represents the color channels). The **Objects**, **Relationships**, and **Attributes** relations store corresponding detected frame elements.

A UDF extends database functionality. A typical UDF takes columns as input, returns a scalar value or a row set, and is used in SQL statements, e.g., in **WHERE** clauses. When querying video databases, VOCAL-UDF supports generating and executing UDFs to identify custom objects, relationships, and attributes, enabling users to find complex, compositional

events. Listing 4.2 shows an example SQL query with UDFs for the objects `car` and `truck`, the relationship `behind`, and the attribute `silver`. VOCAL-UDF supports imperative and declarative Python-based UDFs, and are categorized into the following classes: relationship, attribute, object, and value-lookup.

A **relationship UDF** or **attribute UDF** is a deterministic, scalar predicate that indicates whether an input exhibits a specified relationship or attribute. It accepts zero (i.e., a dummy UDF; see Section 4.3.4) or more columns as arguments and produces a boolean result. The parameters can include any of the following columns from each table in the SQL query’s `FROM` clause: `pixels`, `oname`, `x1`, `y1`, `x2`, `y2`, `rname`, and `aname`. VOCAL-UDF restricts relationship UDFs and attribute UDFs to be frame-level, i.e., they operate on object(s) within the same frame. Therefore, input arguments are all from the same video frame (identified by `vid` and `fid`) and are associated with one or two distinct objects. As an example, we might define a relationship UDF to indicate whether object o_1 is behind another object o_2 by comparing their centroid y -coordinates:

```
def behind(o1_y1, o1_y2, o2_y1, o2_y2):
    return (o1_y1 + o1_y2) / 2 < (o2_y1 + o2_y2) / 2
```

We could also define an attribute UDF to indicate if a detected car is silver by running an ML model over the frame pixels:

```
def silver(pixels, x1, y1, x2, y2):
    cropped_img = pixels[y1:y2, x1:x2]
    is_silver = awesome_color_classifier(cropped_img)
    return is_silver
```

An **object UDF** requires localizing, classifying, and tracking objects in videos. Instead of returning a boolean value, it is a table-valued function that takes a video segment, `video_pixels`, as input. A video segment comprises frames with the same `vid` concatenated into a 4D array with an additional dimension for the frame index. An object UDF detects

and tracks objects of a specific class. Given a video segment, the UDF makes calls to a custom object detection and tracking model and returns a row set, which follows the `Objects` schema listing the detected and tracked objects. As an example, we can define an object UDF that detects all cars in a video:

```
def car(video_pixels):
    obj_tuples = []
    car_detector, tracker = load_models()
    for frame in video_pixels:
        detected_cars = car_detector(frame)
        tracked_objs = tracker.update(detected_cars)
        obj_tuples.extend(tracked_objs)
    return obj_tuples
```

In this dissertation, we assume that object UDFs are given and focus on proposing and generating relationship and attribute UDFs. We leave the extension to object UDFs for future work.

A **value-lookup UDF** is a class of UDFs that simply encapsulate a predicate over existing column values. As an example, suppose the value “car” is in the domain of the object `oname`. Then, a value-lookup UDF can be defined as:

```
def car(oname):
    return oname == 'car'
```

While value-lookup UDFs are not strictly necessary, as they can be directly and easily expressed in SQL statements, we wrap all predicates of our DSL queries in UDFs to simplify the compilation from the DSL to SQL.

UDFs can have different lists of parameters in their signatures. To execute a query with UDFs, VOCAL-UDF first constructs the relational views as shown in Table 4.1 derived from the relations in Table 3.1. The views `ObjView`, `RelView` and `AttrView` contain all attributes that an object UDF, relationship UDF, and attribute UDF can potentially accept as input

arguments, respectively. For simplicity, VOCAL-UDF generates UDFs with a uniform list of parameters for each class.

Object, relationship, and attribute UDFs can be expensive to evaluate, as they may operate on image pixels and invoke ML models. To optimize query execution, VOCAL-UDF caches the results of all UDFs and replaces them with value-lookup UDFs. When executing a UDF over a video corpus for the first time, VOCAL-UDF materializes the results to make it available as a value in the corresponding column and substitutes the UDF with a value-lookup UDF of the same name. For example, when running a query with the **silver** attribute UDF for the first time, VOCAL-UDF evaluates the UDF for each object in the video corpus. If an object **OID1** in a frame **F1** of a video segment **V1** is classified as silver, VOCAL-UDF inserts a new row with values (**V1**, **F1**, **OID1**, ‘silver’) into the **Attributes** relation. Then, VOCAL-UDF replaces the **silver** attribute UDF with a value-lookup UDF that checks whether the **aname** is ‘silver’. Reusing the results of predicate evaluation for query optimization is a long-standing research topic [216, 176, 146, 61] and is not the focus of this dissertation. Batch inference can further accelerate ML-based UDFs.

4.3 VOCAL-UDF approach

VOCAL-UDF needs to address several challenges. First, it must determine whether existing UDFs can adequately answer a user query, or if new UDFs should be created (C1). Second, VOCAL-UDF should support implementing UDFs drawn from diverse range of semantic concepts (C2). Third, since VOCAL-UDF utilizes error-prone LLMs to generate UDFs, it is crucial to ensure high quality in the produced UDFs (C3). We now discuss our solutions to each challenge.

Figure 4.2 shows the architecture of VOCAL-UDF. The user initializes the system with a video dataset and an optional set of UDFs. After preprocessing (Section 4.4), the user can issue NL queries to identify compositional events within the videos. The *Query Parser* (Section 4.3.1) parses the query into the DSL notation described in Section 4.1. If suc-

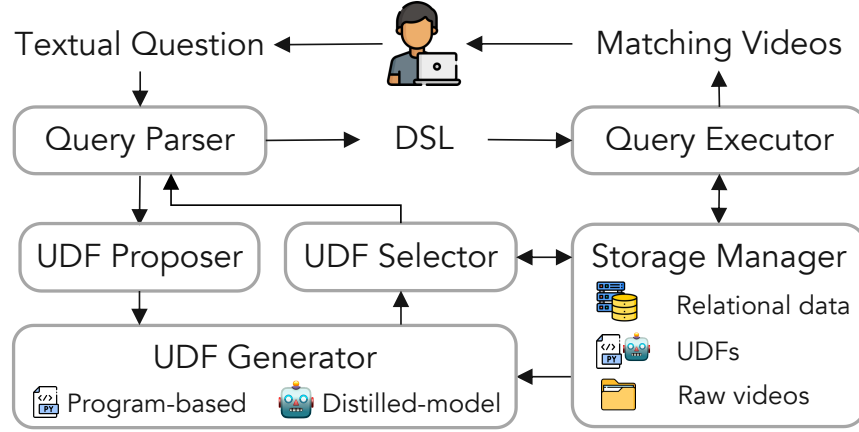


Figure 4.2: VOCAL-UDF system overview.

successful, the DSL query is passed to the *Query Executor* to find all matching videos in the dataset. If the query contains predicates that existing UDFs cannot resolve, the *UDF Proposer* (Section 4.3.1) is invoked to propose the names and descriptions of new UDFs. The *UDF Generator* (Sections 4.3.2 and 4.3.3) creates executable candidates for each proposed UDF. Next, the *UDF Selector* (Section 4.3.4) solicits user labels to select the implementation that best aligns with the user’s intent. With updated UDFs, the *Query Parser* re-parses the query for execution. The *Storage Manager* maintains raw videos, relational data, and available UDFs.

4.3.1 Query parsing and UDF proposal

The Query Parser converts NL queries into our DSL and determines the need for new UDFs. VOCAL-UDF must understand the semantics of the user’s query and available UDFs, mapping each part of the query to an existing UDF or suggesting the creation of a new one. Moreover, VOCAL-UDF needs to be resilient to the linguistic ambiguities and synonymous terms in NL queries.

VOCAL-UDF utilizes LLMs to convert complex NL to a consistent DSL format. LLMs

show strong capabilities in SQL and program generations [160, 203, 75] via in-context learning without fine-tuning. We incorporate domain-specific constraints into LLM prompts to ensure adherence to DSL grammar and conduct post-verification to ensure syntax correctness. Inspired by Wang et. al. [203] and Hsieh et. al. [84], VOCAL-UDF provides the DSL definition, UDF format, and descriptions of available UDFs as NL documentation. This enables VOCAL-UDF to generate grammatically correct DSL queries, determine whether new UDFs need to be created, and interact with LLMs in a zero-shot manner (C1). While VOCAL-UDF focuses on zero-shot prompting, introducing few-shot examples [26] could potentially further improve UDF proposal performance. However, referencing unavailable UDFs in examples may lead LLMs to incorrectly assume their availability at query time. Since the available UDFs can change as the database evolves, examples must be carefully curated for different database states. Figuring out the optimal prompting strategy, though beneficial, is not the focus of this dissertation. An example prompt is shown in Figures A.1 and A.2.

To enhance LLM response reliability, VOCAL-UDF performs post-verification of the generated DSL query to ensure it is syntactically valid and only uses available UDFs. If parse errors occur, VOCAL-UDF appends the error message to the context and asks the LLM to make another attempt.

When the query contains predicates that cannot be resolved using the available UDFs, we prompt the LLM to identify this and propose new UDFs. The output from this process is the function signature and textual description of the proposed UDF generated by the LLM. In case of a query with multiple missing UDFs, VOCAL-UDF returns a list of proposed UDFs and will generate them one by one. An example proposed UDF of **behind** is:

```
{"signature": "behind(o0, o1)", "description": "Whether o0 is behind o1"}
```

Since both the Query Parser and UDF Proposer utilize LLMs, their outputs may sometimes be inaccurate. If the LLM proposes fewer UDFs than needed, it may make performance improvements less noticeable but will not degrade performance. Conversely, proposing more UDFs can increase system runtime and monetary costs; however, as detailed in Section 4.3.4,

we have implemented techniques to ensure that generated UDFs (whether superfluous or inaccurate) do not adversely affect performance. Finally, users may also rephrase their NL query if they are not fully satisfied with the query results.

We show the effectiveness of our approach empirically in Section 4.5 for queries with detailed and explicit descriptions. Semantic parsing (e.g., text-to-SQL) is a highly active research area with significant recent advances [105, 55, 239, 229, 69]. While current methods still fall short of human-level performance [125], improving text-to-SQL is not the goal of this dissertation. VOCAL-UDF focuses on the efficient generation of missing UDFs, relies on LLMs currently adequate performance for text-to-SQL in our context, and will benefit from future advances in this area.

4.3.2 *Program-based UDF generation*

The UDF Generator implements executable UDFs based on the LLM-proposed UDF signatures and descriptions. As discussed, VOCAL-UDF should produce high-quality UDFs (C3) for a wide range of semantic concepts (C2). While VOCAL-UDF leverages the programming capabilities of LLMs [177, 27, 41, 34] to generate UDFs as Python programs, more work is required. Some tasks require complex visual understanding, rendering it challenging to solve via programs (e.g., determining a car’s make). Additionally, even the most advanced LLMs remain error-prone and their programming performance for nuanced tasks is not yet on par with humans.

To address the first problem, VOCAL-UDF supports two types of UDF implementations: program-based UDFs, which are Python programs generated by LLMs (Section 4.3.2), and distilled-model UDFs, which are lightweight vision models distilled from strong pretrained models (Section 4.3.3). To improve generated UDF quality, VOCAL-UDF uses a two-step approach by verifying both syntactic (Section 4.3.2) and semantic (Section 4.3.4) correctness.

At a high level, given a UDF signature h and description d , a video database instance I over schema R , VOCAL-UDF performs the following steps to generate a UDF p :

① Interpretation s_1: check if the center of o0 is positioned behind the center of o1 along the y-axis.	Program p_1: <pre>def behind(o1_y1, o1_y2, o2_y1, o2_y2): o1_center_y = (o1_y1 + o1_y2) / 2 o2_center_y = (o2_y1 + o2_y2) / 2 return o1_center_y < o2_center_y</pre>
Interpretation s_2: check if the center of o0 is behind the center of o1 along the y-axis, based on a threshold distance.	Program p_2: <pre>def behind(o1_y1, o1_y2, o2_y1, o2_y2, **kwargs): thresh = kwargs.get('threshold', 50) o1_cy = (o1_y1 + o1_y2) / 2 o2_cy = (o2_y1 + o2_y2) / 2 return o2_cy - o1_cy > thresh</pre>
Interpretation s_3: Uses the respective positions of the two objects to determine if o0 is behind o1.	Program p_3: <pre>def behind(o0_aname, o1_aname): return o0_aname == 'location_top' and o1_aname == 'location_bottom'</pre>

Figure 4.3: Program candidates, using **behind** as an example.

1. *Generate UDF candidates using an LLM.* Given h , d , I , and R , VOCAL-UDF prompts an LLM to generate a set of k candidate Python functions $\{p_1, \dots, p_k\}$.
2. *Syntactically verify candidates.* VOCAL-UDF executes each candidate p_i on a small dataset sampled from I to verify syntactic correctness, ensuring that all passing candidates are executable.
3. *Semantically verify candidates.* VOCAL-UDF finally evaluates the semantic correctness of the remaining candidates and selects the best one (see Section 4.3.4).

We next describe each step in further detail.

Generating candidate programs using an LLM

In its most basic form, VOCAL-UDF prompts the LLM to generate a program p based on the UDF signature h and description d . To ensure p can be eventually expressed as a SQL predicate, VOCAL-UDF rewrites h in the DSL format to h' that accepts columns from the

schema R as inputs. For example, the `behind` function is rewritten from `behind(o0, o1)` to `behind(o1_y1, o1_y2, o2_y1, o2_y2)`, where `o1_y1`, `o1_y2`, `o2_y1`, `o2_y2` are columns of R . The generated Python program’s input is a set of attribute values that correspond to one or two objects in the video database, depending on the number of variables in h . It generates code that operates on those values, including the `pixels` column, and returns a boolean indicating if the predicate is satisfied or not. Figure 4.3 (program p_1) shows an example program UDF for the `behind(·)` predicate.

To generate quality UDFs, VOCAL-UDF must resolve three challenges. First, NL descriptions are often ambiguous and may not entirely capture the user intent [118] (e.g., user-specific definition of “far”). Second, predicates in video compositional queries often include hyperparameters that require tuning [143, 144] for different datasets and user intents, e.g., determining the threshold distance for the “far” relationship. Lastly, as VOCAL-UDF expands its collection of UDFs and database incrementally, it is essential that new UDFs be able to utilize results from previously established ones.

To address linguistic ambiguity, we provide the LLM with h' and d and ask it to generate a list of k candidate programs with a variety of semantic interpretations: $C = \{(p_1, s_1), \dots, (p_k, s_k)\}$, where p_i and s_i respectively denote a Python program and its semantic interpretation. Figure A.3 shows an example prompt used by VOCAL-UDF in this step. Later, VOCAL-UDF verifies and selects one program for each proposed UDF. See Figure 4.3 for candidate program examples.

To resolve the challenge of parametric predicates, VOCAL-UDF additionally prompts the LLM to generate UDFs with optional numeric hyperparameters and their valid ranges. More formally, VOCAL-UDF prompts the LLM to produce a (possibly empty) list $\Theta = \{(\theta_1, df_{\theta_1}, \min_{\theta_1}, \max_{\theta_1}), \dots, (\theta_i, df_{\theta_i}, \min_{\theta_i}, \max_{\theta_i}), \dots, (\theta_k, df_{\theta_k}, \min_{\theta_k}, \max_{\theta_k})\}$ of parameter names θ_i , default value df_{θ_i} , and range $[\min_{\theta_i}, \max_{\theta_i}]$. In later steps, VOCAL-UDF instantiates each hyperparameter with its default value as well as values sampled from the range and then selects the best program. See Figure 4.3② for an example of a program that relies on a hyperparameter.

To resolve the final challenge of incrementally building the video database, our approach

is to additionally include in the LLM prompt, the active domain of all attributes populated by UDFs, which include `oname`, `rname`, and `aname`. This additional information enables the LLM to leverage the results of existing UDFs as building blocks to dynamically compose more complex UDFs. Figure 4.3③ shows an example program inlining an existing UDF.

Overall, program-based UDFs are well-suited for concepts involving bounding box-like spatial relationships. In addition, other attributes and the `pixels` column can also be used to reason about existing concepts and perform statistical analysis of the pixels in a frame. A program-based UDF could also invoke a pretrained model over each frame, but this is in general expensive and slow. As discussed in Section 4.3.3, a distilled-model UDF is a more efficient approach for concepts that require visual understanding of videos.

Syntax verification

Prior works have proposed various approaches to improve the performance of LLM-generated programs. One line of work leverages unit tests to verify the functional correctness of generated programs [29, 88, 45, 38]. This approach is not suitable for VOCAL-UDF because users do not know in advance what UDFs will be generated, and thus cannot provide labeled data before issuing a query. Another line of work leverages LLMs to automatically generate test cases [34], evaluate the generated programs [41, 190], and select the best one [211]. However, the ambiguity of semantic concepts in video queries means that the correct program is not always unique and often many of the generated programs are reasonable. As a result, the best program cannot be easily identified without user feedback and a working dataset.

Additionally, the quality of an LLM is not guaranteed. To address this issue, VOCAL-UDF uses a two-step approach to verify and select the best program: syntax verification and semantic verification. In the first step, VOCAL-UDF focuses solely on the syntax correctness of the programs. VOCAL-UDF executes each candidate program on a small sample of data from the database I and checks whether: (i) the number and types of inputs and outputs

are correct, (ii) the program can be executed with the data samples, and (iii) Θ , if any, can be parsed successfully. If the verification fails, VOCAL-UDF appends the error message to the context and prompt the LLM to make another attempt. VOCAL-UDF draws samples by constructing tuples that contain attributes and values that respectively correspond to one or two objects in the database. If the program still fails after a few trials, VOCAL-UDF discards the program. In our prototype, we empirically set the number of trials to five. Recent research has demonstrated the potential for formally verifying the correctness of generated UDFs based on a given semantic interpretation [40], which could be incorporated into VOCAL-UDF to further enhance the reliability of its generated Python programs. After generating multiple candidate programs, VOCAL-UDF next utilizes user labels to select the one that best aligns with user intent, which we discuss in Section 4.3.4.

4.3.3 *Distilled-model UDF generation*

While program-based UDFs are powerful and flexible for predicates that reason about existing concepts, bounding box coordinates, or perform a simple statistical analysis of pixel values, they struggle with tasks that require understanding the visual contents of frames. Even though pretrained models like VLMs can be used in a program-based UDF to classify relationships and attributes in a zero-shot manner, running such models over the entire video dataset is expensive. For instance, applying GPT-4o to every frame of the 10,000 five-second videos in the CLEVRER dataset [224] to evaluate a predicate (e.g., “color-red” or some more complex predicate) would cost approximately \$1,413. This cost remains high even when applying techniques such as downsampling or predicate push-down: There are about 1.28M video frames and 5.44M objects to consider. Even if 99% of the image patches corresponding to these objects were filtered, the cost of evaluating a predicate on the remaining 1% of objects would still be around \$14—far higher than generating one UDF (about \$0.15; see Section 4.5.1). VOCAL-UDF’s goal is to generate cheaper UDFs with visual understanding capabilities. However, training a lightweight image classifier from scratch would

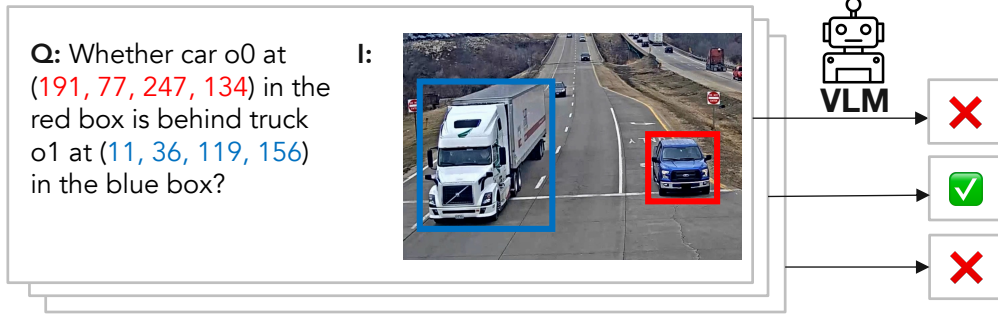


Figure 4.4: Data labeling by a VLM, using **behind** as an example.

require the user to spend a lot of time labeling for just one concept.

Model distillation is a common technique in machine learning to transfer knowledge from a large model to a smaller, more efficient model [85, 196, 28, 36, 83]. Modeling Collaborator [196] is a newly proposed framework that leverages foundation models to train image classifiers for visual concepts using minimal user effort. To do so, given a target concept and description, the system (i) mines relevant images from the public domain, (ii) uses foundation models to annotate sampled images, (iii) trains a lightweight classifier using features extracted from a pretrained model (e.g., CLIP) and labels annotated by the foundation models, and (iv) performs multiple rounds of active learning to further improve its performance.

VOCAL-UDF adopts a similar approach to automatically construct lightweight image classifiers for new concepts without requiring user labeling but with the modifications needed to resolve three unique challenges presented in our compositional query setting. First, random sampling of the user dataset might not give enough positive samples for training, especially for rare concepts. Second, VOCAL-UDF generates UDFs for relationships and attributes, which differ from the concepts in [196]. An image usually includes multiple objects, and VOCAL-UDF needs a different prompting strategy to guide a VLM in classifying specific objects or pairs of objects*. Finally, for the same reason, merely extracting features from the entire image is insufficient to train a good classifier.

Image sampling

To generate a distilled-model UDF, VOCAL-UDF initially randomly samples frames from the user’s video dataset for annotation. However, when the target visual concept is infrequent in the dataset, random sampling does not effectively collect enough positive samples for training. For instance, only 0.83% human-object pairs have an “eating” relationship in the Charades [92] dataset. Our solution is to apply *object-aware sampling* to bootstrap the sampling process. Since all objects of interest are already detected and tracked in the dataset, VOCAL-UDF can filter out irrelevant objects and sample only those likely to be involved in the target concept. For example, when labeling the “eating” relationship, object classes like “food” and “person” are more relevant than “car” and “window”. To do this, VOCAL-UDF first asks an LLM for relevant object classes, and then only samples objects belonging to these classes. Figure A.4 shows an example prompt of object-aware sampling.

Data labeling

VOCAL-UDF uses VLMs to automatically label sampled video frames as positive or negative based on a UDF description. A VLM takes as input an image-text pair and outputs a textual response. However, using a video frame and the UDF description as a direct query is ineffective, since the concepts we are interested in target specific objects or pairs of objects, rather than the frame as a whole. Thus, VOCAL-UDF applies the following prompting strategy, with the goal of encouraging the VLM to focus on particular objects or interactions between two objects. For attribute concepts, we use the UDF description proposed in Section 4.3.1 as the text input, and create the image input by sampling an object from the **Objects** relation and cropping the video frame to include only the object. For relationship concepts, VOCAL-UDF augments the text input with the class names and bounding box coordinates of the relevant objects in the video frame to provide more context to the VLM. It then generates an image patch cropped from the video frame that includes a pair of objects in the same frame from the **Objects** relation. VOCAL-UDF further augments the image

patch by overlaying a red box around the subject and a blue box around the target, thereby providing the VLM with directional information about the relationship. Figure 4.4 shows an example prompt for labeling the **behind** relationship.

Model training

Similar to [193, 196, 53], VOCAL-UDF leverages a pretrained vision model (e.g., CLIP) as the feature extractor and uses the feature-label pairs to train a multi-layer perceptron (MLP). However, for relationship classification, directly extracting features from the image patch containing two objects would perform poorly, as the feature extractor is not aware of the object locations and cannot capture directional relationships (e.g., o_1 holding o_2 vs. o_2 holding o_1). To address this, VOCAL-UDF concatenates features from three versions of an image patch: the original one containing both objects, one where everything except the subject is masked, and one where everything except the target is masked. For attribute UDFs, VOCAL-UDF simply extracts features from the image patch containing the object. In addition, VOCAL-UDF incorporates text features of object class names, derived from the **Objects** relation, to enhance the MLP’s performance. After training an initial model, VOCAL-UDF uses an active learning approach similar to [193, 196] to iteratively improve the model. During each iteration, the trained MLP is run over the unlabeled dataset, VOCAL-UDF selects a batch of samples with the highest uncertainty for labeling by the VLM. VOCAL-UDF then retrains the MLP with the updated labeled dataset.

VOCAL-UDF uses VLMs to automatically label data and the cost is bounded by the training data size, which is at most 500 frames in our evaluation. This is only 0.04% of the 1.28M frames in the entire database for CLEVRER, 0.8% of the 65K frames for CityFlow-NL, and 0.2% of the 289K frames for Charades (see Section 4.5), which correspond to extremely low sampling rates. This cost is thus far lower than directly applying an LLM to the entire video dataset, even with downsampling. Although VOCAL-UDF’s distilled model process may occasionally deliver unsatisfactory performance, its UDF Selector component, described next, mitigates this by selecting the best candidate UDFs and filtering low-performing mod-

els.

4.3.4 UDF selection

For a given UDF, VOCAL-UDF can generate both a distilled-model and a set of program-based implementations. When there are multiple implementation candidates, VOCAL-UDF needs to select the best one. VOCAL-UDF does not have any initial data to validate semantic correctness, only the user NL query. If VOCAL-UDF were to collect labeled data for validation before query execution, the user would need to manually go through the video dataset to find positive and negative examples. Instead, VOCAL-UDF utilizes active learning at query time to strategically request binary labels on carefully selected samples that are most likely to resolve disagreements among UDF candidates. Additionally, VOCAL-UDF leverages LLMs to generate the UDF Python code on behalf of the user (Section 4.3.2) to reduce user effort. Manually writing code snippets for a UDF would require knowledge of Python and could be a tedious trial-and-error process for the user, especially when a UDF requires hyperparameters, operates on pixel values, or leverages existing UDFs as building blocks. We now describe the UDF selection process in detail.

UDF selection and active learning

To best align with the user’s intent, VOCAL-UDF strives to select the candidate UDF that yields the best F1 score for a given set of user labels. As summarized in Algorithm 2, VOCAL-UDF uses active learning [150, 235, 104] to reduce the number of labeled examples needed from the user. Key hyperparameters include: a labeling budget b , thresholds t_p and t_n giving the number of positive and negative samples needed to initiate active learning, and the number of tuples n_s sampled per iteration. Section 4.5 details the values used in our experiments.

During each iteration, the algorithm randomly samples n_s tuples from the database (Line 4). It operates in two phases: *bootstrapping* and *active learning*. The bootstrap-

Algorithm 2: UDF selection using active learning.

Input : Set of unlabeled data U , set of UDF candidates C , and hyperparameters b , n_s , t_p , t_n

Output: Selected UDF with highest score

```

1  $L_p \leftarrow \{\}, L_n \leftarrow \{\}$ 
2  $W \leftarrow \{w_i \mid w_i = 1/|C|, i = 1, 2, \dots, |C|\}$ 
3 for  $i = 1$  to  $b$  do
4    $U_s \leftarrow \text{SAMPLESUBSET}(U, n_s)$ 
   /* Phase 1: Bootstrapping */
5   if  $|L_p| < t_p$  then
6      $L'_p, L'_n \leftarrow \text{PICKPOSITIVE}(U_s, C, W)$ 
7   else if  $|L_n| < t_n$  then
8      $L'_p, L'_n \leftarrow \text{PICKNEGATIVE}(U_s, C, W)$ 
   /* Phase 2: Active learning */
9   else if  $|L_p| < |L_n|$  then
10     $L'_p, L'_n \leftarrow \text{PICKPOSITIVE}(U_s, C, W)$ 
11  else
12     $L'_p, L'_n \leftarrow \text{PICKDISAGREED}(U_s, C, W)$ 
13   $L_p \leftarrow L_p \cup L'_p, L_n \leftarrow L_n \cup L'_n, U \leftarrow U - (L'_p \cup L'_n)$ 
14   $W \leftarrow \text{COMPUTESCORE}(L_p, L_n, C)$ 

```

ping phase collects at least t_p positive and t_n negative samples. While the number of labeled positives ($|L_p|$) or negatives ($|L_n|$) remains below its respective threshold, VOCAL-UDF selects samples most likely to be positive or negative (Lines 5 to 8). Once enough initial labeled data has been collected, VOCAL-UDF transitions to active learning, adopting the margin and positive strategy from [150]. Specifically, if $|L_p| < |L_n|$, it asks the user to label samples most likely to be positive; otherwise, it picks the sample with the greatest disagreement among UDF candidates to differentiate them [235] (Lines 9 to 12).

In $\text{PICKPOSITIVE}(U_s, C, W)$ and $\text{PICKNEGATIVE}(U_s, C, W)$, the algorithm prioritizes VLM annotations obtained in Section 4.3.3 over the UDF candidates' majority vote, as VLM labels are empirically more reliable. In $\text{PICKDISAGREED}(U_s, C, W)$, it computes a disagreement score for each sample in U_s based on UDF candidates C , selecting the one with the highest disagreement [104, 235]. Each sample's disagreement score is the weighted dis-

agreement among UDF candidates, with candidate weights initialized to $1/|C|$ (Line 2) and updated to its performance (F1 score in our prototype) over L_p and L_n after each iteration (Line 14). At the end of each iteration, L_p , L_n and U are updated with the new user labels (Line 13).

Dummy UDFs

Several factors can contribute to the unsatisfactory performance of generated UDFs, including the complexity of the target concept, the low quality of VLM labels, and an insufficient number of positive training samples for rare events. When no UDF candidate performs well, VOCAL-UDF should not apply a UDF that can hurt the performance. To prevent this, VOCAL-UDF appends a *dummy UDF* to its list of UDF candidates. A dummy UDF is a constant-valued function that produces **True** and is equivalent to omitting the predicate. When all other candidates perform poorly, VOCAL-UDF opts for the dummy UDF during its UDF selection process.

UDF generation strategy

Program-based UDFs and distilled-model UDFs offer different trade-offs in terms of performance, interpretability, generation cost, and inference throughput. VOCAL-UDF provides four strategies depending on the user’s preference and the system’s performance requirements: **program** only generates program-based UDFs, **model** only generates distilled-model UDFs, **11m** asks the LLM to decide whether to generate program-based or distilled-model UDFs, and **both** generates both classes of UDFs. Figure A.5 shows an example prompt of using the **11m** UDF generation strategy. Depending on the user specification, one or more program-based UDFs may be generated for each proposed UDF, and one distilled-model UDF is generated for each proposed UDF. By default, VOCAL-UDF uses the **both** strategy to maximize query performance. In Section 4.5.4, we also evaluate the performance of using the **11m** strategy to automatically choose UDF types.

4.3.5 Comparing with direct LLM methods

As an alternative to VOCAL-UDF, a multimodal LLM can directly answer user queries. However, as shown in prior work [137, 206] and as we confirm in Section 4.5.1, LLMs still struggle with *compositional* queries, and directly providing the LLM with the video frames and the user query in a single prompt (the “**vanilla LLM**” approach in Section 4.5.1) leads to low F1 scores. Furthermore, the cost scales with the dataset size, making it expensive for large video collections today, as we discussed in Section 4.3.3 and evaluate in Table 4.2.

A higher performance LLM baseline is to parse the query into our DSL, then use a combination of existing cheap models for common concepts and an LLM for application-specific concepts (i.e., VOCAL-UDF without UDF generation, or “**LLM for concepts**” in Table 4.2). While it may yield good F1 scores, this approach is only practical for short videos and a small number of queries. For larger workloads, the cost and latency of this method are even greater than “vanilla LLM” due to repeated LLM calls to identify concepts for each object or object pair. As Table 4.2 shows, executing just one query with three missing UDFs over the CLEVRER dataset using GPT-4o costs about \$1,504 USD. Reducing costs through sampling is also non-trivial because low sampling rates can degrade query performance, especially for compositional events involving temporal dimensions. As we show in Figure 4.6b, even a small downsampling factor of 8 can greatly reduce the upper bound of achievable F1 scores from 1.0 to 0.51.

VOCAL-UDF overcomes these issues by using the multimodal LLM to learn each new application-specific concept once, over a fixed-size sample of the data, and generate cheaper, domain-specific models (either program-based or distilled-model UDFs) that can then be applied to the remaining videos and to all future queries, avoiding repeated reliance on the LLM and reducing overall costs.

4.4 Implementation

VOCAL-UDF is implemented in Python using the AutoGen framework [213]. The Query Executor converts queries in DSL notation into SQL and uses a relational engine (DuckDB [165] in our prototype) to execute them. We apply the same query translation algorithm as in [235] to optimize query execution. To reduce system latency, we parallelize API calls to the LLM.

VOCAL-UDF requires a preprocessing stage per dataset before queries can be issued. The user provides a video dataset and optionally a set of UDFs. If these UDFs include object detection and tracking models, VOCAL-UDF uses them to populate the **Objects** relation. Otherwise, VOCAL-UDF uses a predefined object detection and tracking model to identify common objects. VOCAL-UDF pre-extracts features into Parquet files for use by distilled models at query time (Section 4.3.3), and we utilize NVIDIA DALI [5] to accelerate this process. VOCAL-UDF also executes the initial UDFs ahead of time to populate the **Relationships** and **Attributes** relations and compute the active domains for **oname**, **rname**, and **aname**. These operations happen as soon as VOCAL-UDF receives the videos and UDFs and before the user issues any queries. If no initial UDFs are supplied, the **Relationships** and **Attributes** relations will be empty. Similarly, when a new UDF is later created, our prototype runs it over the entire dataset and materializes the results in the database. It then converts the UDF into a value-lookup UDF so that subsequent queries can reuse the materialized results. All UDFs are implemented in Python.

There is preprocessing overhead in the initial setup. The cost of running object detection, object tracking, and feature extraction is relatively modest compared to that of LLMs and VLMs, and they can be run locally. The cost of executing a UDF depends on its implementation. In this dissertation, we materialize all results in the video dataset before issuing new queries. Several works have focused on optimizing query execution with UDFs [97, 216], which could be integrated into our system.

The source code for VOCAL-UDF is available at <https://github.com/uwdb/VOCAL-UDF>.

4.5 Evaluation

Baselines. We compare VOCAL-UDF against VisProg [75] and EQUI-VOCAL [235], both of which are limited to predefined UDFs when answering queries. VisProg uses a different set of modules for different tasks, so we manually write new modules for the system to be able to answer compositional queries. These include both logical (e.g., **Eval**, **Event**, **Before**) and conceptual (e.g., **Red**, **Holding**) modules. VisProg also requires in-context examples, which we create separately from the evaluation queries. While EQUI-VOCAL does not need such examples, it does not accept NL queries as input. To address this, we adapt VOCAL-UDF’s approach of converting NL into the EQUI-VOCAL DSL using an LLM. We further compare VOCAL-UDF with two LLM baselines as described in Section 4.3.5.

Metrics. We evaluate query answering performance using F1 scores, precision, and recall. We propose and generate UDFs using training data and report query performance over the test set. We evaluate each dataset using 30 queries, and each query is run three times. We additionally evaluate the system efficiency in terms of latency and cost. For latency, we report the wall-clock time as perceived by the user from the moment they submit a query to the moment when VOCAL-UDF returns the query results. For cost, we measure both the LLM API invocation cost using the OpenAI pricing model [6] and the resource cost using the AWS pricing model [2, 1].

Datasets: We evaluate VOCAL-UDF on three datasets: CLEVRER [224], CityFlow-NL [64] and Charades [92]. CLEVRER is a benchmark that facilitates testing queries with varying complexities. CityFlow-NL and Charades represent real-world applications—traffic monitoring and human activities—typical for video data management systems. UDFs in CityFlow-NL include fine-grained vehicle types, while Charades comes with complex human interactions. The evaluation queries are based on concepts from these datasets, which are

labeled by the original annotators and reflect events from those domains.

CLEVRER: The CLEVRER [224] dataset consists of 10,000, 5-second synthetic videos of moving objects. To determine ground-truth information, following [224], we use a Mask R-CNN [78] to locate objects and predict their colors, shapes, and materials. We write rule-based functions to extract spatial relationships and attributes. For our experiments, we use the same Mask R-CNN and rule-based functions as UDFs, for a total of six relationships and 17 attributes. Among them, we select two relationships and nine attributes as the *base UDFs*, which are available to all systems during query evaluation. We use templates to automatically generate 30 target queries with seven predicates, up to three variables (i.e., three distinct objects), up to three region graphs, and duration constraints with three possible values. We ensure there are at least 5% positive examples in the dataset for each target query (we ensure the same for all datasets). We then rewrite the DSL queries as NL queries using GPT-4 and manually verify the correctness.

CityFlow-NL: The CityFlow-NL [64] dataset contains traffic videos captured from multiple cameras and NL descriptions for vehicle tracks. Following [120], we extract vehicle colors and types from the NL descriptions. Since only sampled vehicle tracks are annotated, we only consider them in our evaluation. We extract six spatial relationships using rule-based predictors and create 1,473 non-overlapping 50-frame video segments from the original dataset. To create UDFs for attributes, we train binary image classifiers. For relationships, we use the same rule-based functions as UDFs. Then, we select three relationships and four attributes as *base UDFs*. We automatically generate 30 queries with up to four predicates, up to three variables, up to three region graphs, and duration constraints with three possible values. An example query is "a vehicle o0 with type sedan is to the right of another vehicle o1 for at least 15 frames. Then, o0 is to the left of o1 for at least 15 frames", which could be used to find sedans changing their positions with other vehicles.

Charades: The Charades [92] dataset contains 30-second indoor activity videos. We focus exclusively on frames with scene graph annotations provided by Action Genome [92], which only provide human-object relationships but do not cover attributes or object-object rela-

tionships. Thus, our evaluation queries focus on human interactions with objects. At query time, only human-object pairs are sampled to collect more positive examples. Incomplete annotations is a common issue in real-world datasets [35] like Charades. To address this, we augment the dataset by automatically generating dense spatial relationship annotations using rule-based predictors to replace the existing ones. For semantic relationships, we train binary image classifiers. In total, we have 20 relationships (five spatial, 15 semantic) and 35 object classes. Among them, we select two spatial and seven semantic relationships as the base UDFs. We automatically generate 30 target queries with up to four predicates (one is always `object(oi, 'person')`), up to three variables, and up to two region graphs, but without duration constraints. An example query is "a person is eating something, and then the same person is holding another object while being in front of it", which could retrieve videos including a person taking a bite from an apple, and then picking up a cookbook in front of them to read a recipe.

For each dataset, we use half of the videos as training data and the rest as test data. To generate ground truth labels, we run each target DSL query on the dataset.

Evaluation setup. We conduct all experiments on a compute cluster. For each experiment, we request one node with eight Intel Xeon Gold 6230R CPUs at 2.10GHz, 200GB of RAM, and one NVIDIA A40 GPU. We use the GPT-4o model (`gpt-4o-2024-08-06`) as the LLM and VLM across all systems. We configure VOCAL-UDF as follows. For the CLEVRER dataset, we use a labeling budget of 20 for each UDF during selection. We generate 10 candidate programs, allowing numeric parameters and frame pixels as inputs. During UDF selection, we bind each numeric parameter to its default value as well as five random values. When generating distilled-model UDFs, we ask the VLM to annotate 100 sampled frames per UDF. Given their increased complexity, for the CityFlow-NL and Charades datasets, we set the labeling budget per UDF to 50 and the number of VLM-annotated frames per UDF to 500. 100 frames are annotated for the initial training, and then active learning is performed to select 100 additional frames in each round. Frame pixels are disallowed in program-based UDF generation due to their limited impact on performance. To focus our evaluation on

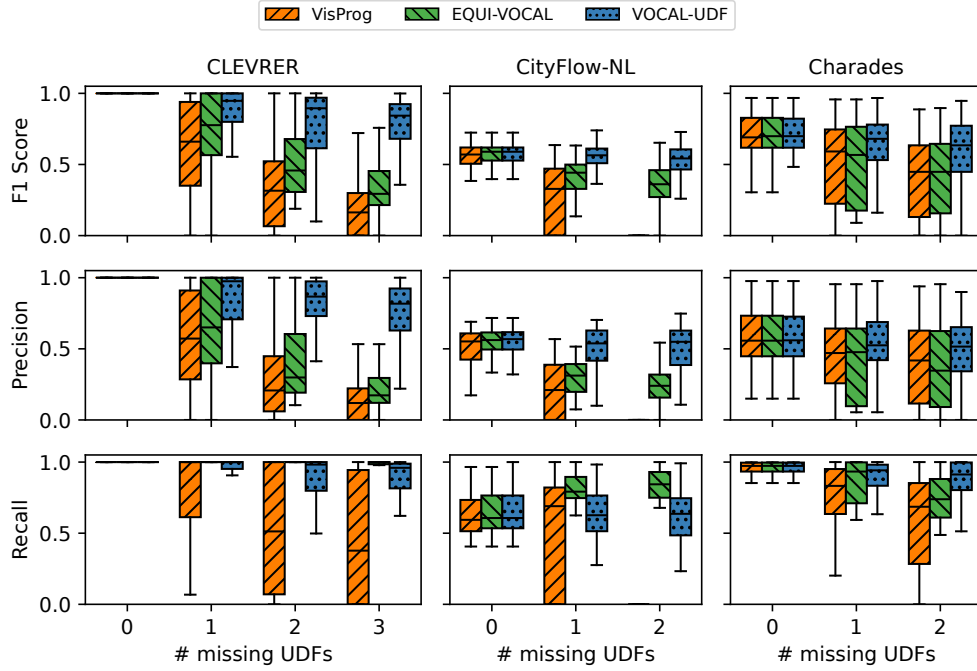


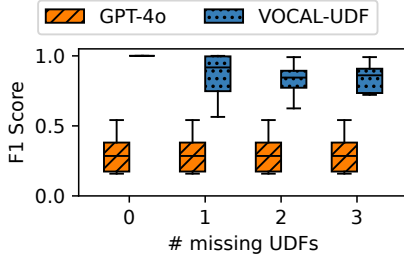
Figure 4.5: End-to-end performance of generated queries with various number of missing UDFs.

relationships and attributes, we assume objects are pre-detected and tracked.

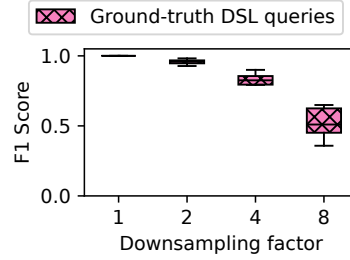
4.5.1 End-to-end performance

F1 score, precision, and recall.

We first evaluate the end-to-end performance of VOCAL-UDF and baseline systems by varying the number of missing UDFs from zero (i.e., all required UDFs are available) to three. In each experiment, all base UDFs are available, but X supplemental UDFs are randomly removed. We execute thirty distinct queries three times. Figure 4.5 presents the F1 scores, precision, and recall. **VOCAL-UDF maintains high performance even when UDFs are missing.** When all required UDFs are available, all systems perform similarly. For



(a) Vanilla LLM vs. VOCAL-UDF.



(b) Downsampling reduces F1 scores.

Figure 4.6: (a) F1 scores of the vanilla LLM approach and VOCAL-UDF on CLEVRER under a simplified setting. (b) F1 scores of running ground-truth DSL queries over sampled videos.

CLEVRER, F1 scores with no missing UDFs reach 1.0 since predefined UDFs are also used to generate ground-truth scene graph annotations. However, CityFlow-NL and Charades contain less accurate ML models in their UDFs, leading to lower F1 scores. When UDFs are missing, VisProg and EQUI-VOCAL experience significant F1 score drops, as they cannot generate new UDFs, with VisProg performing worse due to its higher likelihood of including unavailable UDFs. VOCAL-UDF mitigates F1 score degradation by generating new UDFs as needed, though performance drops more on Charades as its missing UDFs involve complex, harder-to-generate semantic relationships. Interestingly, while EQUI-VOCAL achieves higher recall than VOCAL-UDF on CLEVRER and CityFlow-NL when UDFs are missing, it does so at the expense of substantially lower precision, for an overall lower F1 score.

Comparing with direct LLM methods

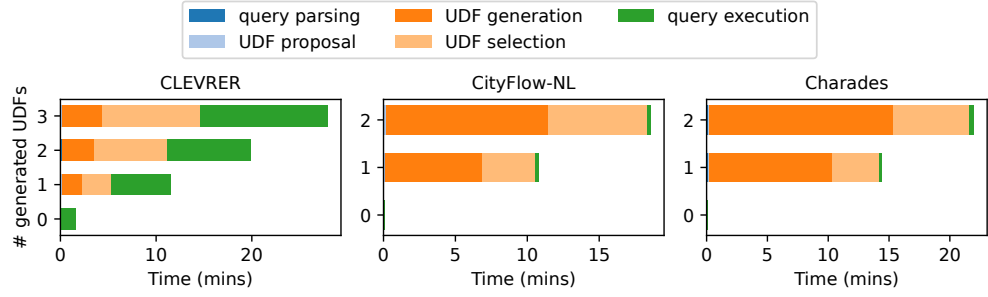
To understand the performance gap between VOCAL-UDF and using an LLM directly, we evaluate both approaches on CLEVRER. We use the same queries from the end-to-end experiments, but under a simplified setting due to the high expense of GPT-4o. We select 500 videos and downsample each by 75%, retaining every fourth frame. We use the first

10 queries as evaluated in Section 4.5.1 but remove any duration constraints due to the downsampling. We compare VOCAL-UDF to a vanilla LLM method in which we provide GPT-4o with a sequence of video frames and ask with the prompt: “Examine the sequence of frames from a video and determine if the following event occurs? `{user_query}`. Answer with ‘yes’ or ‘no’.” We use OpenAI’s Batch API that offers 50% lower cost. Figure 4.6a shows that **VOCAL-UDF achieves significantly higher F1 scores than the vanilla LLM approach**. The cost of the vanilla LLM method scales with the dataset size, averaging \$5.17 USD per query under this simplified setting. If extended to the full 5,000 videos in the test set of CLEVRER, the estimated cost would be \$207 USD per query.

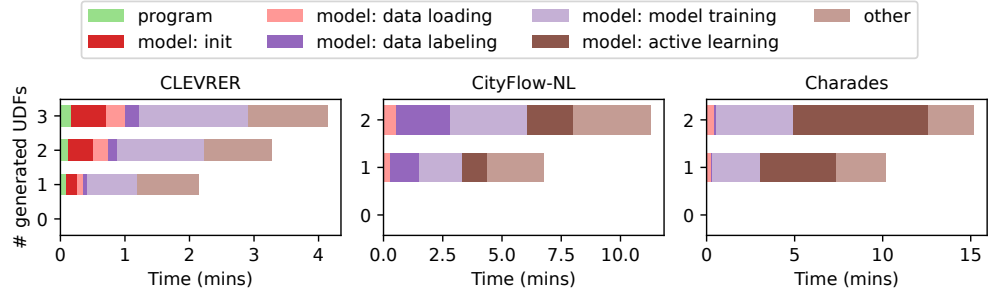
As an alternative baseline, one can use an LLM to classify new concepts instead of generating UDFs. However, as discussed later in Section 4.5.1, the cost of doing so without downsampling is prohibitively high. While downsampling reduces LLM costs, it can also degrade query performance. Figure 4.6b shows the F1 scores of executing ground-truth DSL queries (i.e., assuming perfect LLM predictions) over the simplified CLEVRER with various downsampling factors. **Downsampling greatly reduces the upper bound on achievable F1 scores, with performance dropping to 0.51 at a downsampling factor of just 8.**

Execution time.

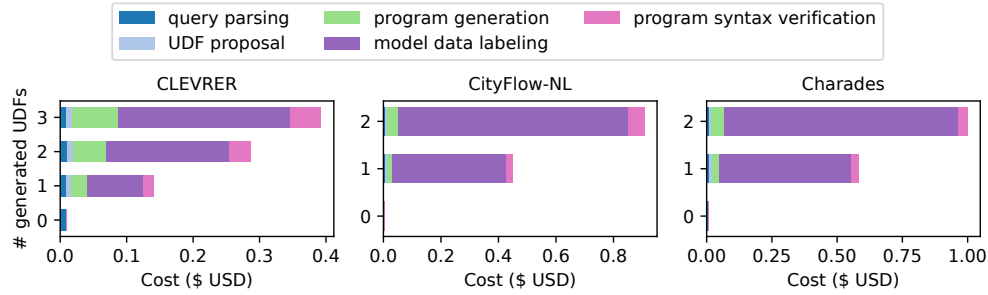
To analyze VOCAL-UDF’s efficiency, we perform an end-to-end evaluation of execution time. Figure 4.7a breaks down the wall-clock time for each system component, with each stacked bar showing the mean execution time for queries with different numbers of UDFs that VOCAL-UDF *decides to generate*. VOCAL-UDF may occasionally decide to generate extra, unnecessary UDFs. For clarity, we omit CLEVRER with four generated UDFs and CityFlow-NL with three generated UDFs from Figure 4.7, as each has only one data point—arising from the LLM’s inaccurate decision—and does not reflect an average execution time for that number of generated UDFs. **The average execution time is under 100 seconds when all required UDFs are available and increases proportionally with**



(a) Total time.



(b) UDF generation time.



(c) Monetary cost of OpenAI API invocation.

Figure 4.7: Detailed breakdown of efficiency and cost.

the number of generated UDFs. In CLEVRER, query execution takes longer when new UDFs need to be created, requiring several minutes to process the full 14-hour video dataset as VOCAL-UDF materializes new UDF outputs. Subsequent queries using the same UDFs, however, are as fast as the case with no missing UDFs. CityFlow-NL and Charades are faster

due to smaller datasets and the exclusion of frame pixels in program-based UDF generation, which eliminates the need for video loading and reduces the program execution cost. For CityFlow-NL and Charades, UDF generation dominates the execution time, as the number of VLM-annotated frames per UDF increases to 500, with active learning applied after labeling every 100 samples. UDF selection requires non-negligible time across all datasets, involving tasks such as frame loading, feature extraction, program execution, and model prediction.

Figure 4.7b provides a detailed breakdown of wall-clock time for the UDF generation process. Since VOCAL-UDF asynchronously parallelizes calls to the LLM, we measure the time breakdown as follows. All categories except for “other” capture the synchronous portions of each sub-step (i.e., excluding LLM calls), while “other” represents the remaining observed time, which includes the time spent explicitly waiting on asynchronous LLM calls. Program-based UDF generation is labeled as “program,” while distilled-model UDF generation is further divided into sub-steps (“init”, “data loading”, etc.). **Distilled-model UDF generation takes significantly longer than program-based UDFs, but no single sub-step consistently dominates across all cases.** Specifically, training distilled models takes the longest in CLEVRER and CityFlow-NL, whereas active learning takes the longest in Charades due to a larger dataset. The active learning step could be accelerated by performing it on a sample of the unlabeled dataset. The results also suggest that **LLM calls do not dominate UDF generation time**, thanks to asynchronous execution. For example, during data labeling, VOCAL-UDF submits batches of 100 asynchronous tasks to maximize parallelization.

Monetary cost.

To understand the query execution cost, we consider both the LLM API invocation cost and the compute resource cost. Figure 4.7c shows the mean monetary cost of OpenAI LLM calls per query. **VOCAL-UDF costs less than \$1.0 USD per query, with data labeling for distilled-model UDFs being the primary cost factor.** Moreover, GPT-4o can be replaced by open-source models. For instance, Amazon Bedrock offers Llama 3.1 70B for a

Table 4.2: LLM cost estimation for executing one query over CLEVRER’s full test set, with three missing UDFs.

Method	GPT-4o	Llama 3.1 70B
VOCAL-UDF	\$0.4	\$0.03–0.11
Vanilla LLM	\$207	\$59
LLM for concepts	\$1504	\$430

Table 4.3: Natural language to DSL correctness.

Dataset	F1 = 1	F1 $\geq$ 0.98
CLEVRER	79%	100%
CITYFLOW-NL	92%	99%
CHARADES	83%	97%

price of \$0.72 USD per 1M tokens [1], which is $3.5\times$ cheaper for input and $14\times$ cheaper for output tokens than GPT-4o. Other open-source models and hosting options [4] are available to further reduce costs, with a potential trade-off in query F1 scores. Compute resource costs depend on deployment scenarios. For example, on a `g4dn.12xlarge` EC2 spot instance, the estimated average resource cost of running a query with the largest number of missing UDFs is \$0.7 USD for CLEVRER, \$0.4 for CityFlow-NL, and \$0.5 for Charades [2]. Thus, **the total per-query cost for each dataset remains under \$1.5.**

Table 4.2 compares the estimated LLM costs of VOCAL-UDF, “vanilla LLM”, and “LLM for concepts” (as described in Section 4.3.5) using both GPT-4o and Llama 3.1 70B. Due to the high expense of the baselines, we estimated costs rather than running full experiments. We estimate the cost of “LLM for concepts” as follows: Each CLEVRER video contains an average of 544 objects and 1,933 object pairs. Each LLM call requires 300 input tokens (255 for the image and 45 for the task) and one output token (“yes” or “no”). We first run the query with missing UDFs removed, then invoke the LLM only on the matching videos to minimize costs. As shown in Table 4.2, **VOCAL-UDF is much more cost-effective than direct LLM baselines.** While further optimizations—e.g., evaluating immutable attributes on a single frame and propagating them to other frames for the same object—could reduce the cost of “LLM for concepts”, the cost remains proportional to the dataset size, in contrast to VOCAL-UDF.

Table 4.4: Proposing UDFs. FPs are incorrectly proposed UDFs, and FNs are missed UDFs.

Dataset	# New UDFs	# proposed UDFs	# FP	# FN
CLEVRER	540	441	27	126
CITYFLOW-NL	270	261	4	14
CHARADES	270	229	19	60

4.5.2 UDF proposal

Translating NL queries to DSL is not a contribution of this dissertation, but we report it as it affects the end-to-end performance. For this experiment, all UDFs are available, and the LLM translates the same queries from the end-to-end experiment, running each three times. We measure the F1 score of the generated queries. As shown in Table 4.3, **VOCAL-UDF can effectively parse the complex textual queries into the DSL notation**, returning correct video segments at least 79% of the time, which increases to $\geq 97\%$ if we consider an F1 score ≥ 0.98 . Common translation errors include missing and redundant predicates, misuse of UDFs with similar names, and incorrect temporal ordering of predicates.

We evaluate VOCAL-UDF ’s UDF proposal using the same set-up as the end-to-end evaluation. A proposed UDF is considered correct if its name matches one of the target query’s ground truth UDFs, with a manual check for synonyms. As shown in Table 4.4, **VOCAL-UDF proposes fewer UDFs than expected, with more false negatives (FNs) than false positives (FPs)**. This occurs because some new UDFs are equivalent to available UDFs (e.g., “above(o1, o2)” vs. “beneath(o2, o1)”). However, this is beneficial as it avoids unnecessary UDF generation. Most other FNs are mainly because VOCAL-UDF decides to approximate using other available UDFs (e.g., “touching” for “holding”). **Most FPs arise from proposing compositional UDFs**, such as “behind_and_near” for “behind” and “near”.

Table 4.5: Program-based UDFs performance.

Dataset	best = “program”		best $\neq$ “program”	
	best UDFs	all UDFs	best UDFs	all UDFs
CLEVRER	0.995	0.203	0.664	0.024
CITYFLOW-NL	1.000	0.831	0.349	0.203
CHARADES	1.000	0.092	0.162	0.030

Table 4.6: Program-based UDF types.

Dataset	best = “program”				best $\neq$ “program”			
	all	reuse	param	pixel	all	reuse	param	pixel
CLEVRER	195	52	59	40	37	8	14	12
CITYFLOW-NL	83	25	14	—	91	90	16	—
CHARADES	70	32	15	—	75	33	28	—

4.5.3 UDF generation

We now evaluate the performance of program-based UDFs. We consider all correctly proposed program-based UDFs across the 270 experiments with the largest number of new UDFs (3 datasets $\times$ 30 queries $\times$ 3 runs). We classify UDFs into two categories, one where at least one of their best implementations is program-based, and another where the best is not program-based. Table 4.5 shows the median F1 score of the best generated program for each UDF (best UDFs) and all generated programs for each UDF (all UDFs). **When the best UDF generation is program-based, VOCAL-UDF consistently produces high-quality programs.** However, not all generated UDF candidates are of high quality; for instance, the median F1 score of all program-based UDFs for Charades is only 0.092. Thus, VOCAL-UDF must carefully select the best candidate during the UDF selection phase. When the best UDF is not program-based, F1 scores decrease significantly, indicating that certain relationships or attributes may not be well-suited to program-based UDFs.

As described in Section 4.3.2, VOCAL-UDF supports program generation with various

Table 4.7: Distilled-model UDFs performance.

Dataset	best = “model”		best $\neq$ “model”	
	model	dummy	model	dummy
CLEVRER	0.889	0.519	0.556	0.662
CITYFLOW-NL	0.687	0.289	0.735	0.664
CHARADES	0.724	0.207	0.302	0.728

enhanced features. We evaluate how often VOCAL-UDF uses these features by categorizing program-based UDFs into four types: “all” (represents all generated programs), “reuse” (utilizes results from existing UDFs), “param” (takes numeric parameters as inputs), and “pixel” (accepts frame pixels as inputs). Table 4.6 shows the number of best generated programs for each UDF in each category. The results indicate that the best programs span various types, **highlighting the importance of enhanced features in generating high-quality program-based UDFs.**

Using the same method as above, we evaluate the performance of distilled-model UDFs. UDFs are classified into two categories, one where at least one of their best implementations is distilled-model, and another where none are distilled-model. Table 4.7 shows the median F1 score for both distilled-model generation and dummy generation (as the baseline) for each UDF. **When the best UDF generated is distilled model, VOCAL-UDF shows a significant improvement in F1 scores over the baseline.** When the best UDF is not distilled model, the F1 scores of distilled models are comparable or worse than the baseline, indicating that certain relationships or attributes may not be suitable for distilled-model UDFs. For CityFlow-NL, F1 scores are higher for UDFs whose best implementations are not distilled models, likely due to the complexity of semantic attributes that are more challenging to classify.

To access VOCAL-UDF’s ability to handle ambiguous intent in program-based UDFs, we test three concepts from CLEVRER—“behind”, “far”, and “location.bottom”. For each concept, we execute the same query with one predicate but three different interpretations. For

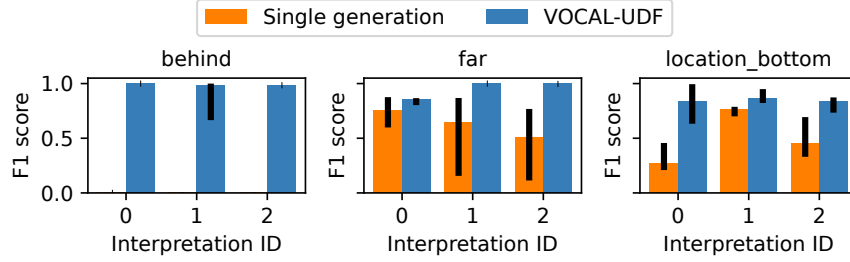


Figure 4.8: Median F1 scores of VOCAL-UDF vs. a single-generation baseline on queries with one predicate under various interpretations, each run 20 times. Error bars show the IQR.

instance, “behind” is interpreted as: (1) o1’s center above o2’s center, (2) o1’s center below o2’s center, and (3) o1’s bottom edge above o2’s bottom edge. We compare VOCAL-UDF to a baseline that uses an LLM to generate a single program without numeric hyperparameters or UDF reuse. Figure 4.8 shows the F1 scores for each concept across varying interpretations, showing that **VOCAL-UDF can more robustly adapt to different definitions of the same concept**. The baseline struggles with “behind” since it uses an interpretation based on bounding box overlap that mismatches all three ground truths. While it works well for one interpretation of the other concepts, its single-generation method fails to accommodate varying interpretations, leading to significant performance drops. One limitation of VOCAL-UDF is that a concept becomes fixed once a UDF is generated. The UDF would have to be explicitly deleted or disabled if a user wanted a different interpretation for an already-existing concept. Additionally, VOCAL-UDF currently does not handle intent ambiguity for distilled-model UDFs, but this can be extended by asking LLMs to generate diverse descriptions of the target concept and distilling multiple models.

We further examine the VLM’s labeling quality during distilled-model UDF generation on the CityFlow-NL and Charades datasets. We evaluate nine attributes in CityFlow-NL and 18 relationships in Charades. For each attribute or relationship, we use GPT-4o to label a randomly selected, balanced set of 500 samples. Table 4.8 shows the average F1 score of

Table 4.8: VLM labeling quality.

CityFlow-NL		Charades			
suv	0.843	holding	0.776	sitting on	0.897
white	0.906	standing on	0.846	covered by	0.749
grey	0.797	carrying	0.761	eating	0.737
van	0.956	wiping	0.685	touching	0.632
sedan	0.903	leaning on	0.816	wearing	0.854
black	0.819	drinking from	0.916	lying on	0.857
red	0.973	writing on	0.894	above	0.683
blue	0.904	in front of	0.644	beneath	0.561
pickup truck	0.956	behind	0.599	in	0.444

Table 4.9: UDF selection performance and selected UDF distribution, using “both” strategy.

Dataset	No.	best	80% of best	program	model	dummy
CLEVRER	232	163 (70%)	218 (94%)	183 (79%)	37 (16%)	12 (5%)
CITYFLOW-NL	174	144 (83%)	154 (89%)	94 (54%)	76 (44%)	4 (2%)
CHARADES	145	93 (64%)	123 (85%)	81 (56%)	23 (16%)	41 (28%)

each concept over three runs. **GPT-4o generally achieves high labeling quality for a wide range of concepts:** seven out of the 27 concepts achieve F1 scores of at least 0.9, and 15 attain F1 scores of at least 0.8. However, GPT-4o struggles to label the five spatial relationships, suggesting that **VLMs like GPT-4o are still limited in spatial reasoning.**

4.5.4 UDF selection

Table 4.9 reports the UDF selection results. The “best” column shows the number of UDFs selected by VOCAL-UDF that achieve the highest F1 score among all UDF candidates, while the “80% of best” column shows how many attain at least 80% of the best F1 score. **VOCAL-UDF effectively selects better-performing UDFs from candidates, even with a labeling budget as low as 20.** Although VOCAL-UDF does not always pick

Table 4.10: Number of correctly selected UDF types. “No.” represents the number of proposed UDF instances.

Dataset	best $\neq$ “dummy”				best = “dummy”	
	No.	both	llm (gpt-4o)	llm (gpt-4-turbo)	No.	both
CLEVRER	225	203 (90%)	208 (92%)	145 (64%)	7	3 (43%)
CITYFLOW-NL	174	154 (89%)	170 (98%)	102 (59%)	0	0 (—)
CHARADES	125	90 (72%)	104 (83%)	57 (46%)	20	18 (90%)

the best UDF due to similar scores for many candidate UDFs, it can still select a good UDF implementation at least 85% of the time (with an F1 score of at least 80% of the best implementation). Table 4.9 also breaks down the types of UDFs that VOCAL-UDF selects. **To handle diverse semantic concepts, VOCAL-UDF generates and selects UDFs of various types**, underscoring the necessity for VOCAL-UDF to support two different types of UDFs. Interestingly, VOCAL-UDF selects more dummy UDFs (41 instances, 28%) on the Charades dataset than the other datasets. Notably, 20 instances are “holding”, which is difficult to distinguish under the current labeling budget. Another 17 cases are “in”, which our rule-based predictor identifies by checking bounding box overlap. This method classifies most object pairs as having this relationship, thereby allowing even the dummy UDFs to perform exceptionally well.

VOCAL-UDF supports four UDF generation strategies, as described in Section 4.3.4. We now compare the “both” and “llm” strategies in selecting the correct UDF type. When multiple candidates of different types share the highest F1 score, any of these types is considered correct. Using the same end-to-end experiment setting with the largest number of missing UDFs, Table 4.10 shows the number of correctly selected UDF types. The results are divided into two categories: one where the best UDF type is not “dummy” and one where it is “dummy.” When the best UDF type is not “dummy,” **VOCAL-UDF correctly selects the UDF type at least 72% of the time with the “both” strategy**, reaching up to 90% on CLEVRER. Interestingly, **using “llm” with GPT-4o yields a higher accuracy than “both”**. However, this strategy is highly sensitive to the model used; switching to

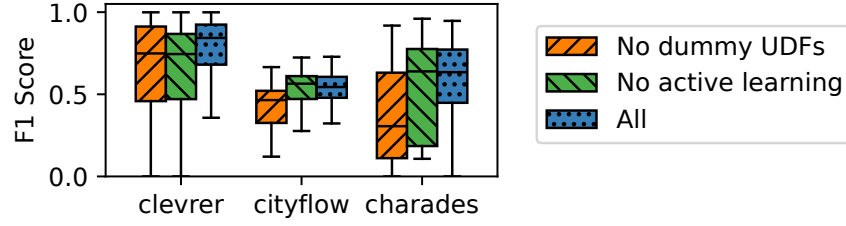


Figure 4.9: F1 scores of VOCAL-UDF without active learning and without dummy UDFs in UDF selection.

an earlier GPT-4 Turbo model reduces the accuracy to 46–64%. Despite this, the “11m” strategy demonstrates potential to further improve performance while reducing latency and cost. When the best UDF type is “dummy,” VOCAL-UDF can also select the correct UDF type 43% to 90% of the time using the “both” strategy.

We study the impact of active learning and dummy UDFs to the UDF selection process, using the same end-to-end experiment setting with the largest number of new UDFs. Figure 4.9 compares F1 scores across three system variants: the complete system with both active learning and dummy UDFs (All), the system with random sampling and dummy UDFs (No active learning), and the system with active learning but without dummy UDFs (No dummy UDFs). **Both active learning and dummy UDFs help achieve higher F1 scores.** For CityFlow-NL, the best UDF implementations are easily distinguishable from candidates, so active learning does not improve F1 scores but also does not degrade system performance.

VOCAL-UDF reduces user effort by using active learning to request labels at query time rather than asking the user to provide all examples upfront. Figure 4.10 compares the number of samples needed under two approaches—active learning during UDF selection and random sampling before query execution—to obtain at least ten positive examples for a given UDF. We consider all generated UDFs from the end-to-end experiment and classify them as “easy” (spatial relationships and attributes with relatively balanced labels) or “hard” (other concepts with greater class imbalance). **Active learning allows VOCAL-UDF**

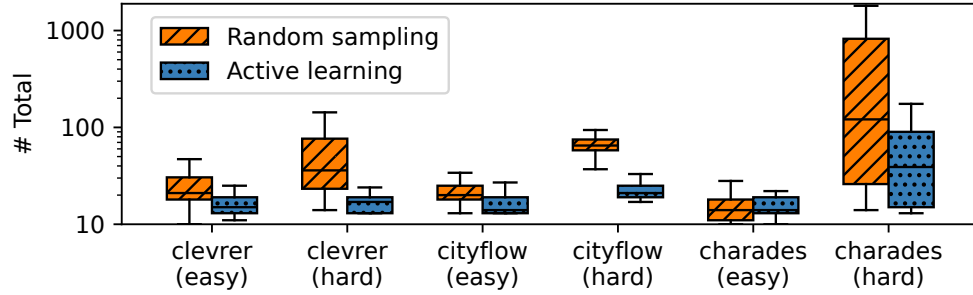


Figure 4.10: Number of samples required to get at least 10 positive examples (y-axis in log scale; lower is better).

to collect positive samples with reduced labeling effort, particularly for “hard” concepts. Since most object pairs exhibit “in” relationships in Charades, random sampling requires labeling only 11.5 samples on average to yield 10 positives. As a result, the overall labeling effort for “easy” concepts in Charades is similar between random sampling and active learning.

4.6 Limitations

VOCAL-UDF does not currently support relationship and attribute UDFs involving state changes over time, such as identifying a car “moving close” to a truck, which requires distance comparisons at different timestamps. As an approximation, multiple frame-level UDFs can represent each state separately; for instance, detecting a “far” relationship followed by a “near” one.

Materializing and reusing UDF results can be problematic when the definition of an attribute or relationship is not objective (e.g., “near”) and varies from query to query. To address this, VOCAL-UDF could allow users to access and manage generated UDF descriptions, implementations, and results, enabling them to dynamically enable or disable specific UDFs before issuing queries.

VOCAL-UDF’s performance is affected by its ability to semantically understand the user

queries and available UDFs. We assume that predefined UDFs are relevant to the target domain and that an LLM is capable of disambiguating each UDF effectively. The performance of VOCAL-UDF thus heavily depends on the underlying LLM, which is why we used the latest LLMs (e.g., GPT-4o) to ensure high-quality semantic understanding. Furthermore, VOCAL-UDF currently supports queries with detailed and explicit descriptions. Extending support to vague queries is left for future work.

VOCAL-UDF leverages LLMs in various system components, making the quality of query results dependent on the underlying LLM performance. To enhance the reliability of LLM-generated answers, VOCAL-UDF employs syntax verification, generates multiple candidate UDFs, and leverages user annotations via active learning to select the best implementation. Introducing dummy UDFs further ensures that newly generated UDFs do not negatively impact query results. While users already provide annotations to guide UDF selection, VOCAL-UDF could be extended to incorporate additional human interventions, allowing users to access intermediate results for manual examination [220, 234, 47]. Our experiments empirically demonstrate the promising potential of our LLM-based approach, and we posit that as LLMs advance, VOCAL-UDF performance will correspondingly improve.

4.7 Summary

In this chapter, we presented VOCAL-SIFT, a new system that supports compositional video queries with the capability to generate new UDFs. VOCAL-SIFT utilizes LLMs to parse NL queries and automatically determine the need for new UDFs. It supports both program-based and distilled-model UDF generations and improves UDF quality through syntax verification and semantic verification.

Chapter 5

VOCAL-SIFT: OPTIMIZING HIGH-LEVEL VIDEO QUERIES VIA NECESSARY-PREDICATE FILTERING

In this chapter, we present VOCAL-SIFT, our video query processing system that enables efficient high-level NL querying over large video datasets.

The previous two chapters introduced systems that target *compositional* video queries. EQUI-VOCAL (Chapter 3) lowers the bar for formulating compositional queries by synthesizing a declarative query from a handful of positive and negative examples, freeing users from writing precise spatio-temporal specifications. VOCAL-UDF (Chapter 4) extends this support to NL by automatically generating any missing predicates as new UDFs, allowing users to articulate queries directly in NL. Together, these systems answer queries whose semantics can be fully expressed by composing visual atoms—objects, attributes, relationships, and temporal relations. Figure 5.1 (a) illustrates this approach. Other video querying systems [67, 216, 143, 75] follow the same approach: each visual subtask is delegated to a module, either predefined or supplied as a UDF, that solves it in isolation, and the query engine composes the results.

Many real-world video queries, however, are **high-level**: the user specifies a concise NL event description rather than an explicit composition of visual primitives. For example, a user might ask, “Show me instances of car accidents.” Although some such events could in principle be expressed as large disjunctions of predicate compositions, this representation is not provided by the user and is often impractical to enumerate. Such queries can arise because the event in question involves implicit domain knowledge, substantial visual variability, or subjective interpretation. For instance, evaluating “a Biellmann spin in figure

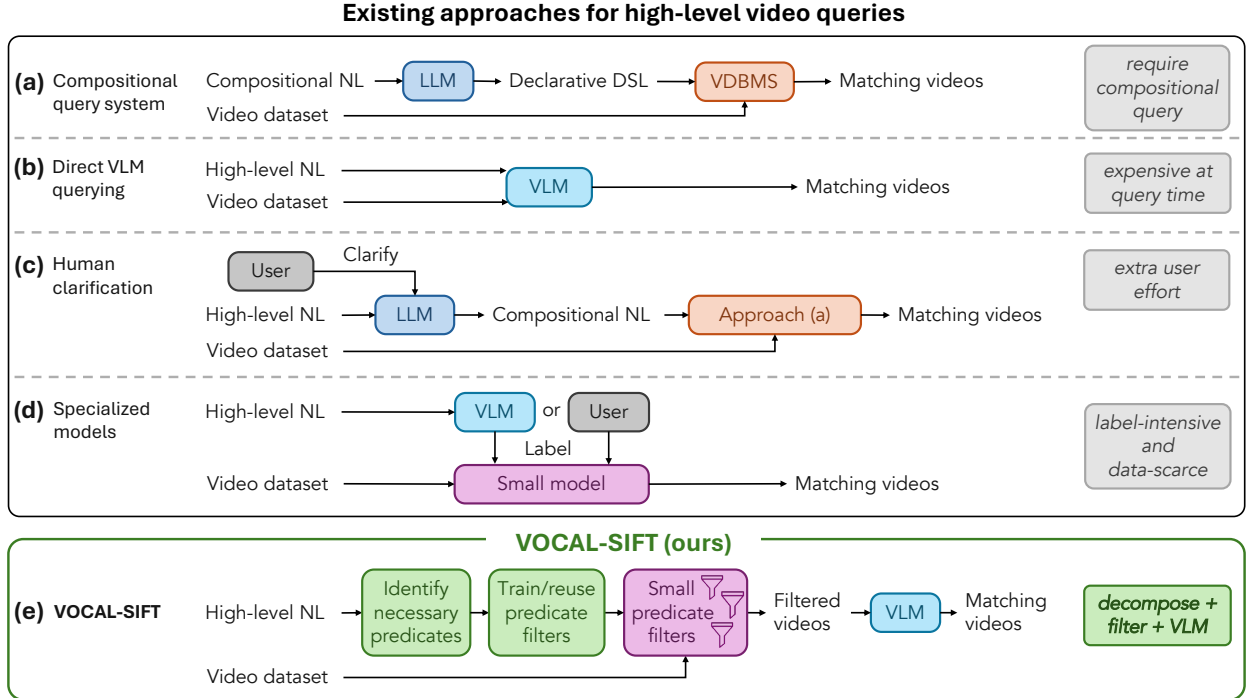


Figure 5.1: Comparison of approaches for high-level video query processing. Existing approaches either require users to provide compositional queries, invoke expensive VLMs over the full dataset, rely on user clarification, or train a single specialized model. VOCAL-SIFT decomposes high-level queries into necessary predicates, uses lightweight predicate filters to prune the video corpus, and invokes the VLM only over the filtered videos.

skating” requires implicit domain knowledge to map the named technique to its underlying visual structure, such as a skater holding a raised skate behind their head while rotating. Conversely, an event like a “goal celebration in soccer” exhibits substantial visual variability: the action might manifest as a knee slide, a backflip, or a group embrace, varying significantly across positive examples. Finally, distinguishing “aggressive behavior” from “playful wrestling” relies heavily on subjective interpretation; while the explicit visual primitives (e.g., grabbing, falling) may be identical, accurately classifying the event requires inferring underlying intent and emotional state.

Consequently, answering a high-level query requires case-by-case semantic reasoning that goes beyond evaluating a query consisting of a specific set of predicates. Recent VLMs

provide a natural mechanism for this form of semantic reasoning because they can condition on an open-ended NL event description and combine heterogeneous visual evidence with commonsense and domain knowledge. Rather than matching a query to a fixed list of predicates, a VLM can weigh multiple partial cues when deciding whether the event occurred. For example, for “alley-oop in basketball”, a VLM may jointly consider the players, ball, basket, pass trajectory, airborne receiver, and finish near the rim. These cues provide useful evidence, but they are not individually necessary, sufficient, nor do they constitute a complete definition of the event. Prompting techniques [110, 148] can further enhance such reasoning. However, directly applying VLMs to every video in a large corpus remains time-consuming and computationally expensive, with cost scaling linearly in the number of sampled frames. For instance, directly invoking GPT-4o over a 7-hour video can cost hundreds to thousands of US dollars for answering a single query [236]. Direct VLM evaluation therefore preserves high-level semantics, but applies the expensive model to every video (Figure 5.1 (b)). Our evaluation shows that direct VLM evaluation is up to $13.6\times$ more expensive than VOCAL-SIFT (Table 5.4).

An alternative is to leverage interactive clarification. Recent work on LLMs has explored asking users clarifying questions to resolve task ambiguity [114, 121]. Adapting this approach to video querying could allow the system to ask follow-up questions that prompt the user to describe the visual logic of the event more explicitly. These answers can be converted into a more explicit compositional query and passed to existing video query systems. However, this approach presumes an always-available user and repeated back-and-forth introduces additional latency and user effort (Figure 5.1 (c)), which we demonstrate in our user study in Figure 5.6.

Another option is to train a smaller, task-specific model through either direct supervision or model distillation [28, 36, 83, 196] to approximate the high-level query (Figure 5.1 (d)). However, supervised training demands substantial user labeling effort, often requiring hours of manual annotation for a single query, despite recent efforts to streamline the process [53, 150]. Model distillation aims to transfer knowledge from a large reference model to a smaller,

task-specific one by using the reference model to annotate training data automatically, but naively generating sufficient training data for rare or complex events remains expensive, and distilled models can struggle to capture intricate or highly variable query semantics at practical model sizes (see Figure 5.3 and Table 5.3).

Given these limitations of existing approaches, we propose VOCAL-SIFT, a video query processing system that enables efficient high-level NL querying over large video datasets. VOCAL-SIFT follows the “decompose, filter, then evaluate” path in Figure 5.1 (e). VOCAL-SIFT does not try to replace the final VLM judgment with a complete compositional definition. Instead, it automatically identifies **necessary predicates**, which are visually detectable conditions that should hold whenever the high-level event occurs, even though they may not be sufficient to prove that the event occurred. For selected necessary predicates, VOCAL-SIFT reuses or trains lightweight predicate filters, applies them to prune non-matches, and invokes the expensive VLM only on the remaining candidate videos.

This design builds on a rich line of VDBMS research focused on avoiding expensive predicate evaluations. Prior systems have introduced probabilistic predicates, proxy models, and multi-level classifier cascades that use cheap approximate tests to filter data before invoking an expensive oracle [136, 100, 12, 99, 102]. These approaches, however, assume that the target predicate is already known and explicitly specified by the user, such as an object predicate (e.g., “bus”), an attribute predicate (e.g., “red color” or “speed > 60”), or a cardinality predicate over detected objects (e.g., “contains at least five cars”). VOCAL-SIFT generalizes these proxy model principles from explicitly defined predicates to high-level queries where the necessary visual conditions must be dynamically inferred. By automatically discovering these necessary predicates for a high-level event and realizing them as lightweight filters, VOCAL-SIFT extends the proven efficiency of proxy models to the domain of open-ended, high-level semantic querying.

Realizing this design raises three key challenges. First, the system must automatically decompose a high-level query into necessary predicates. This is difficult because high-level queries are underspecified and may admit many valid visual instantiations. For example, an

“alley-oop in basketball” may be executed as a one-handed dunk, a two-handed layup, or from different passing angles. The system must infer predicates that are both necessary and visually detectable (e.g., “player”, “ball”, “basket”, and “airborne receiver near the rim”) without adding non-essential conditions (e.g., “dunk” or “crowd celebration”). This difficulty is compounded by the variability of LLM outputs: naively prompting an LLM may yield inconsistent decompositions across runs, omit critical predicates, or introduce spurious ones. We empirically demonstrate the limitations of naive LLM prompting in our evaluation (Section 5.5.3). VOCAL-SIFT addresses these issues with an LLM-based, multi-step semantic decomposition process. Inspired by prior work that improves LLM reliability by sampling and reconciling multiple reasoning paths [211, 60], VOCAL-SIFT performs multiple independent decompositions and then reconciles recurring predicates into a consistent set of necessary predicates. In the alley-oop example, this design allows VOCAL-SIFT to preserve consistently extracted predicates (e.g., “airborne receiver”), merge equivalent descriptions (e.g., “near the rim” and “near the basket”), and discard spurious predicates (e.g., “fast break”).

Second, once candidate predicates are identified, the system must decide which ones should be instantiated as filters and which ones can be omitted. Instantiating a filter for every predicate is costly and can be wasteful, because predicates within the same query are frequently correlated and add little additional pruning power. For example, in figure skating, a “spinning” predicate usually implies that the skater is already balanced on one leg, so training both “spinning” and “balancing on one leg” filters is largely redundant. Yet this decision is difficult because the selectivity of each predicate and its marginal contribution to pruning are not known a priori. Reusing filters across queries introduces a complementary challenge: enabling reuse without biasing decomposition. Reuse is essential for amortizing training cost, but exposing the library of existing predicates too early can cause the decomposition process to force-fit newly encountered semantics to available functions even when the match is only approximate. For example, if the library already contains “touching”, the system may be tempted to map a query meaning “holding” to that predicate, even though

holding implies a much more specific interaction than mere contact. The system must therefore separate semantic reasoning from reuse, and align new predicates to existing filters only after decomposition has been completed. VOCAL-SIFT addresses the reuse problem by separating decomposition from library alignment. It first lets the LLM produce necessary predicates without constraining them to the existing filter library, and only afterward aligns the completed predicate set against the library. This allows VOCAL-SIFT to reuse filters when the semantics match exactly, while avoiding approximate substitutions that could distort the query. For predicates that do not have reusable filters, VOCAL-SIFT then applies filter selection. It uses an LLM to estimate whether or not each missing predicate is largely implied by the filters already available for the query, and trains a new filter only when the predicate is expected to provide enough additional pruning benefit to justify its construction cost.

Finally, the system must instantiate selected predicates as reliable filters. A training-free baseline is to use zero-shot embedding filters: a pretrained vision-language encoder maps both video frames and predicate text into the same vector space, and the filter retains videos whose frame vectors are close to the text vector. However, embedding similarity produces continuous scores, so filtering still requires a rule for turning those scores into video-level filter decisions. As our evaluation shows (Section 5.5.2), effective rules vary across predicates and datasets, and zero-shot embedding filters do not consistently retain true matches while pruning enough non-matches. VOCAL-SIFT instead instantiates selected predicates as distilled predicate filters, using VLM labels to train compact classifiers over precomputed frame representations. This approach learns predicate-specific decision boundaries that can adapt to the target dataset and be reused across later queries. Relationship and attribute predicates, such as “near” or “silver”, raise additional challenges. For example, a relationship predicate “holding” may refer to a player holding a basketball, an audience member holding a water bottle, or a coach holding a tablet. Although one could in principle annotate and train filters for the predicate alone, without specifying its participating objects, doing so is difficult in practice: the VLM may not label all instances consistently, and the positive examples may

vary substantially in their participating objects, visual appearance, and image location. As a result, the distilled model must learn from a more varied set of positive examples, which may be harder to train from frame-level features alone. VOCAL-SIFT addresses this issue by *grounding* each such predicate in its participating objects during annotation and inference, producing more specific filters such as “a player holding a basketball”. Our experiments show that grounding improves both label quality and filter selectivity (Section 5.5.5).

In summary, VOCAL-SIFT makes the following contributions:

- We introduce necessary-predicate filtering for high-level video queries. VOCAL-SIFT automatically identifies essential visual conditions to prune the video corpus before applying an expensive VLM oracle, avoiding the need for users to write complex compositional queries that explicitly describe the full visual logic for a video event or scene of interest.
- We design a multi-step semantic decomposition process to derive reusable necessary predicates by independently generating candidates, aligning them against an existing library, and reconciling LLM outputs into a robust predicate set (Section 5.2).
- We develop a filter selection method that avoids training redundant predicate filters by estimating semantic dependency, which reduces the filter construction cost while preserving filtering quality (Section 5.3).
- We instantiate necessary predicates as object-grounded distilled predicate filters (Section 5.4). Experiments with our prototype implementation and three datasets show that VOCAL-SIFT maintains high recall while substantially reducing expensive VLM invocations, and a user study confirms that our automatic decomposition reduces task completion time compared to human clarification, without a significant degradation in filtering quality or user experience (Section 5.5).

Overall, we show in this chapter how VOCAL-SIFT enables users to seamlessly express

Algorithm 3: VOCAL-SIFT overview.

Input: Video corpus V , user’s NL query t , library of existing atomic predicates P ,
partial map from predicates to instantiated filters M

/ Stage 1: Parsing */*

- 1 $q \leftarrow \text{PARSE}(t, P)$
- 2 $P_A, P_H \leftarrow \text{CLASSIFYPREDICATES}(q)$

/ Stage 2: Semantic Decomposition */*

- 3 $P_q \leftarrow P_A$
- 4 **foreach** $p_h \in P_H$ **do**
- 5 $P_q \leftarrow P_q \cup \text{SEMANTICALLYDECOMPOSE}(p_h, t, P)$
- 6 $P \leftarrow P \cup P_q$

/ Stage 3: Filtering */*

- 7 $P_{reuse}, P_{inst}, P_{skip} \leftarrow \text{SELECTFILTERS}(P_q, M)$
- 8 **foreach** $p \in P_{inst}$ **do**
- 9 $M[p] \leftarrow \text{INSTANTIATEFILTER}(p, V)$
- 10 $F_q \leftarrow \{M[p] : p \in P_{reuse} \cup P_{inst}\}$
- 11 $V_{cand} \leftarrow \text{APPLYFILTERS}(V, F_q)$

/ Stage 4: Evaluation */*

- 12 $V_{ans} \leftarrow \text{EVALUATECANDIDATES}(V_{cand}, t)$
- 13 **return** V_{ans}, P, M

an additional category of queries over video datasets, significantly extending the practical application of video data management systems.

5.1 VOCAL-SIFT overview

Answering video queries requires bridging user intent with executable operations over large video corpora while balancing expressiveness and efficiency. VOCAL-SIFT achieves this goal by taking as input a NL query that contains a combination of *atomic* and *high-level* predicates. Atomic predicates name low-level object, attribute, or relationship conditions that existing visual operators can evaluate directly, while high-level predicates name semantically rich events whose visual structure is implicit. A query must contain at least one predicate of either kind. VOCAL-SIFT identifies the high-level predicates in the query and extracts a set of atomic predicates from each of them. For example, given the query “alley-oop

in basketball”, VOCAL-SIFT recognizes “alley-oop” as a high-level predicate and extracts atomic predicates such as the presence of a player, a ball, and a basket, together with an airborne receiver near the rim. VOCAL-SIFT then effectively determines the overall subset of predicates that are worth evaluating to filter away video snippets that do not satisfy the query. Finally, VOCAL-SIFT invokes an expensive VLM on the remaining video snippets to confirm whether they satisfy the query.

More specifically, VOCAL-SIFT processes each query in four stages: *parsing*, *semantic decomposition*, *filtering*, and *evaluation*. Parsing converts the NL query into a structured representation and identifies which predicates are already atomic and which are high-level. Semantic decomposition rewrites each high-level predicate into a set of necessary predicates that are concrete enough to support filtering. Filtering then decides which existing filters can be reused and which new ones should be instantiated, and applies them to prune the corpus to a candidate set. Finally, evaluation invokes an *oracle*—a strong but expensive model such as a VLM, whose judgment we treat as the ground truth for the query—only on the remaining candidates. Algorithm 3 summarizes this end-to-end pipeline.

Algorithm 3 begins by parsing the user query t into a structured query q expressed in the same DSL used by EQUI-VOCAL and VOCAL-UDF (Section 3.1.2). The parser then partitions the predicates in q into atomic predicates P_A and high-level predicates P_H (lines 1-2). Both steps are LLM-driven. Given t and the existing predicate library P , the LLM produces a DSL expression and labels each predicate as *atomic* if it describes an explicit visual condition over objects (e.g., “basket”), attributes (e.g., “a player being airborne”), or relationships (e.g., “a player near the basket”). Predicates that instead name a broader event or activity whose visual meaning is not yet explicit in this form are labeled as *high-level* (e.g., “alley-oop in basketball”). For each $p_h \in P_H$, VOCAL-SIFT invokes SEMANTICALLYDECOMPOSE to rewrite it into a set of necessary predicates represented as atomic predicates (line 5 and Section 5.2). These predicates are merged with P_A to form the query-specific predicate set P_q , which is then added to the atomic predicate library P so that future queries can reuse any newly discovered predicates.

The filtering stage is then realized in two steps. First, `SELECTFILTERS` examines P_q together with the predicate-to-filter map M and classifies each predicate in P_q into one of three groups: those whose filters can be reused (P_{reuse}), missing predicates worth instantiating as new filters (P_{inst}), and missing predicates whose filters would add little pruning benefit and are therefore skipped (P_{skip}) (Section 5.3). Second, for each $p \in P_{inst}$, `VOCAL-SIFT` invokes `INSTANTIATEFILTER` and records the new filter in M , making it available to future queries as well (Section 5.4). The active filter set F_q for the current query is then assembled from M over $P_{reuse} \cup P_{inst}$, and applying it to V yields the candidate set V_{cand} (line 11).

Finally, `VOCAL-SIFT` invokes `EVALUATECANDIDATES` on V_{cand} rather than on the full corpus, preserving the expressiveness of oracle-based evaluation while avoiding unnecessary oracle calls (line 12). Subsequent queries can benefit from the predicates discovered during decomposition and the filters trained during instantiation.

The following sections elaborate on the three core components that make this pipeline effective. Section 5.2 describes how `VOCAL-SIFT` transforms high-level predicates into necessary predicates, Section 5.3 explains how it decides whether a missing predicate is worth instantiating, and Section 5.4 describes how the selected predicates are realized as concrete filters.

5.2 Semantic decomposition

Given a high-level predicate p_h from the parsed query, semantic decomposition identifies simpler visual conditions that should hold whenever p_h occurs. `VOCAL-SIFT` represents these conditions as a query-specific set of *necessary predicates*: atomic object, attribute, and relationship predicates that are implied by the high-level event and concrete enough to support efficient filtering before oracle evaluation. For our running query “alley-oop in basketball”, decomposition yields necessary predicates such as the presence of a “player”, a “ball”, and a “basket”, together with the relationship “player near the basket” and the attribute “airborne player”. These predicates do not define the event completely, but each is

a visually grounded condition that should hold in positive videos and can therefore be used to discard videos where the condition is absent.

This problem is difficult because high-level predicates are both underspecified and visually variable. A query such as “alley-oop in basketball” names the technique, but it leaves implicit the visual evidence to recognize it, such as the players, the ball, the basket, the pass, and the airborne receiver. At the same time, valid instances may differ in pose, camera view, timing, or execution style, so a decomposition that fixes one canonical scenario can incorrectly exclude positives. The decomposition must therefore identify predicates that remain necessary across this variability while still being visually detectable.

Existing proxy model and cascade systems show the value of filtering negatives before invoking an expensive oracle (Section 2.4.1). These systems assume that the target predicate to approximate is already explicit, such as “detecting buses”, “identifying red cars”, or “checking whether a frame contains at least five cars”. VOCAL-SIFT adapts the same proxy model principle, but must first reason about the *semantics* of an underspecified, high-level predicate to decide which visual conditions can be mapped to inexpensive filters that will also help to effectively eliminate negatives. Semantic decomposition closes this gap by translating an open-ended event description into explicit necessary predicates.

An alternate approach to answering queries that include high-level predicates is to ask the user for clarifications [114, 121, 237, 107]. The system can ask the user for the details of the objects, relationships, and attributes they would expect to see in their event. For “alley-oop in basketball,” the system might ask the user to list the participating objects (e.g., player, ball, basket), the defining relationships (e.g., a player jumping near the basket while receiving a pass), and any required attributes (e.g., that the receiver is airborne). Human clarification can expose useful domain knowledge [231, 167, 10], and we evaluate it as a baseline in Section 5.5.7. However, it adds interaction latency, requires an always-available user, and assumes that users can articulate the relevant visual logic, which can be difficult for domain-specialized events.

To avoid burdening and slowing down the user, VOCAL-SIFT performs this reason-

ing automatically with an LLM. As described in Section 5.1, the parser translates an NL query into a structured representation that follows the DSL used by the previous systems. Semantic decomposition operates over the predicate portion of this representation. Each necessary predicate is a frame-level condition over a scene graph: object predicates name object categories, such as `player`; attribute predicates attach a property to an object, such as `airborne(player)`; and relationship predicates connect two participating objects, such as `near(player, basket)`. We use this function-style notation to make predicate arguments explicit. This schema constrains the LLM’s output to structured predicates rather than free-form descriptions such as “a player jumps to catch the ball above the rim”. It also provides the interface required by `SELECTFILTERS` and `INSTANTIATEFILTER` in later stages, and supports cross-query reuse because semantically identical predicates can share the same library entry and filter.

Concretely, VOCAL-SIFT processes the NL query t in two LLM-driven stages. The first stage, parsing (Section 5.1), translates t into the schema and labels each predicate as *atomic* if it describes an explicit visual condition over objects, attributes, or relationships, and as *high-level* otherwise; for “alley-oop in basketball”, the entire query is labeled as a single high-level predicate p_h . The second stage, semantic decomposition, operates on one high-level predicate at a time. For each p_h , VOCAL-SIFT prompts the LLM with the full query t (for context) and p_h (the target), asking which objects, relationships, and attributes must be present whenever the event occurs.

Since learned filters are imperfect and necessary predicates may be weakly visible or transient in positive videos, VOCAL-SIFT combines the decisions of the predicate filters using a *fractional aggregation policy* rather than requiring every filter to accept every true positive: a video is retained whenever a sufficient fraction $\alpha \in (0, 1]$ of the filters for the selected predicates accept it. We describe in more detail and evaluate this policy in Section 5.5.1. Fractional aggregation preserves the semantic role of necessary predicates while making filtering robust to occasional filter errors.

Reliable and reusable predicate generation

Directly prompting an LLM to generate the final predicate set in one step is insufficient in practice for three reasons. First, if the LLM has access to the existing predicate library P during generation, it may try to express the high-level event using existing predicates even if the latter are not an accurate match. For example, if P contains `touching(player, ball)` but not `holding(player, ball)`, the LLM may settle on `touching(player, ball)` for “alley-oop” even though touching is satisfied by far more situations than the brief catch at the rim. Second, LLM outputs are stochastic, so repeated generations can produce inconsistent decompositions. For “alley-oop”, one run might propose `near(player, rim)` while another proposes `near(player, basket)`; both refer to the same condition, but a single run captures only one of them. Third, the LLM may introduce spurious predicates that are not truly necessary for the event. For “alley-oop”, one-shot generation may include predicates such as `crowd_celebration` or `fast_break`, which are typical of, but not required by, the play; filtering on such predicates would eliminate valid positives.

VOCAL-SIFT addresses these challenges with a multi-step semantic decomposition process. It first generates candidate predicates without access to the existing predicate library, then aligns the candidates to reusable library entries when their meanings match, and finally reconciles repeated LLM generations into a single predicate set.

The first step, *query decomposition*, produces candidate necessary predicates for a high-level predicate p_h in the context of the full query t . To avoid bias from existing system state, this step runs *without access* to the predicate library P : the LLM is asked to identify objects, attributes, and relationships that must be present whenever the high-level event occurs, using its own vocabulary rather than choosing from a fixed list. Because a single LLM generation may omit relevant predicates or introduce noisy ones, VOCAL-SIFT invokes this step multiple times independently. Each invocation returns one candidate decomposition, and the reconciliation step described below aggregates them into a single predicate set for p_h .

The second step, *query alignment*, compares each candidate predicate against the existing library P . A candidate predicate is mapped to a library predicate only when the two have the same meaning; otherwise, it is retained as a new predicate. Running alignment as a separate step allows VOCAL-SIFT to reuse existing filters without letting the library distort the initial semantic reasoning. For example, `rim` is aligned with the library predicate `basket` because the rim and the basket refer to the same visual referent in this domain, whereas `holding(player, ball)` is not approximated by `touching(player, ball)`.

The final step, *query reconciliation*, aggregates the independent, aligned decompositions into a single predicate set for p_h , following prior work that improves LLM reliability by sampling and reconciling multiple reasoning paths [211, 60]: each query decomposition invocation provides one independent sample, and reconciliation combines them. Reconciliation proceeds in three steps. First, VOCAL-SIFT uses the LLM to *canonicalize* new candidate predicates so that surface variants across runs (e.g., `hoop` and `rim`) collapse onto a single predicate name. Second, it *groups equivalent candidates* across runs—those aligned to the same library predicate or sharing a canonical name—so that `near(player, rim)` and `near(player, basket)` are merged while `jumping(player)` and `airborne(player)` remain distinct. Third, it retains a candidate as *sufficiently supported* only if a majority of decompositions produce it, on the rationale that necessary predicates should recur across samples while spurious ones such as `fast_break` typically appear in only one or two. The retained predicates are added to P_q in Algorithm 3 for downstream filter selection.

We compare this multi-step decomposition process against a direct decomposition baseline in Table 5.5, showing that it achieves better end performance empirically.

5.3 Filter selection

After semantic decomposition, a high-level query is rewritten into a query-specific predicate set P_q (Algorithm 3). Some predicates in P_q already have filters available in the predicate-to-filter map M , while others require new instantiation. Instantiating every missing

Table 5.1: Notation used in filter selection.

Symbol	Meaning
V	Video corpus.
q	Current query.
P	Library of atomic predicates discovered across queries.
M	Map from predicates to instantiated filters.
P_q	Predicate set produced by semantic decomposition for query q .
P_q^+	Predicates in P_q whose filters are already available in M .
P_q^-	Predicates in P_q that do not yet have instantiated filters.
S	Predicate subset used for filtering.
$V_{cand}(S)$	Candidate videos retained after applying the filters for predicates in S .
C_I	Cost of invoking the oracle once on one candidate video.
C_T	One-time construction cost for a new predicate filter.
$\Delta_q(p)$	Oracle evaluation savings for query q from applying a filter for p .
$P(p S)$	Probability that a video passes p given that it passes the filters for S .
$\tau_q(p)$	Threshold on $P(p P_q^+)$ for instantiating p .

predicate can be wasteful because predicates generated for the same query are often correlated. For example, if filters for **player**, **ball**, and **basket** already retain mostly basketball clips, a separate filter for **near(player, basket)** may remove few additional videos, and the one-time cost of training it may not be justified by the savings. The goal of SELECTFILTERS is therefore to decide which missing predicates are worth instantiating.

Problem formulation

We formulate filter selection as a trade-off between the one-time cost of instantiating a new filter and the future savings in oracle calls that such a filter enables. Table 5.1 summarizes the notation used in this section. For a query q , let

$$P_q^+ = \{p \in P_q : p \in \text{dom}(M)\} \quad \text{and} \quad P_q^- = P_q \setminus P_q^+,$$

where $\text{dom}(M)$ is the set of predicates for which M already stores an instantiated filter. Thus, P_q^+ contains predicates whose filters are already available, and P_q^- contains predicates

that require new instantiation. For any predicate subset $S \subseteq P_q$, let $V_{cand}(S)$ be the videos that pass the filters associated with S .

Consider a missing predicate $p \in P_q^-$. If VOCAL-SIFT does not instantiate a filter for p , the oracle is applied to all videos in $V_{cand}(P_q^+)$. If it instantiates a filter for p , the oracle is applied only to $V_{cand}(P_q^+ \cup \{p\})$. Let C_I be the oracle evaluation cost per video. The reduction in oracle evaluation cost for query q is therefore

$$\Delta_q(p) = C_I \cdot \left(|V_{cand}(P_q^+)| - |V_{cand}(P_q^+ \cup \{p\})| \right). \quad (5.1)$$

Now define

$$P(p \mid P_q^+) = \frac{|V_{cand}(P_q^+ \cup \{p\})|}{|V_{cand}(P_q^+)|},$$

i.e., the probability that a video satisfies predicate p given that it already satisfies all predicates in P_q^+ . Substituting this relation gives

$$\Delta_q(p) = C_I \cdot |V_{cand}(P_q^+)| \cdot (1 - P(p \mid P_q^+)).$$

Thus, the instantiation decision is essentially a correlation decision: the higher $P(p \mid P_q^+)$ is, the less additional pruning benefit a separate filter for p can provide.

If predicate p recurs across multiple queries in a workload Q , the total benefit of instantiating it is the sum of the per-query savings:

$$\Delta(p) = \sum_{q \in Q: p \in P_q} \Delta_q(p).$$

Let C_T denote the one-time cost of generating a new filter. An ideal workload-level policy would instantiate p whenever $\Delta(p) > C_T$. Because VOCAL-SIFT does not know the future workload at query time, it approximates this decision using the current query.

Quantifying the threshold

We next quantify how large $P(p \mid P_q^+)$ must be before instantiation becomes unnecessary for a single query. For a query q and a missing predicate $p \in P_q^-$, considering only the immediate benefit on that query gives

$$\Delta_q(p) > C_T \iff P(p \mid P_q^+) < 1 - \frac{C_T}{C_I |V_{cand}(P_q^+)|}. \quad (5.2)$$

Define the single-query threshold

$$\tau_q(p) = 1 - \frac{C_T}{C_I |V_{cand}(P_q^+)|}.$$

Then instantiation is worthwhile only when $P(p \mid P_q^+) < \tau_q(p)$. Because $\tau_q(p)$ depends on the current candidate count, it can vary substantially across queries and datasets.

Empirical characterization of predicate correlations

Existing techniques estimate correlations from annotated concept datasets [233, 39] or from linguistic and commonsense knowledge [82, 226, 132]. In our setting, however, VOCAL-SIFT must reason about previously unseen predicate combinations without maintaining a labeled co-occurrence dataset or training a separate probabilistic model.

Before describing how VOCAL-SIFT estimates $P(p \mid P_q^+)$, we first conduct an empirical analysis to examine whether decomposed predicates are correlated in practice. The analysis measures the two quantities that determine whether a missing predicate is worth training: how strongly decomposed predicates imply one another, and how many candidate videos remain after existing filters have already been applied. We use VideoNet [218] as a representative corpus of domain-specialized action videos. Section 5.5.1 describes the dataset and benchmark construction in detail. Here, we sample 50 action names as high-level queries. We run semantic decomposition on each sampled query, instantiate filters for the resulting

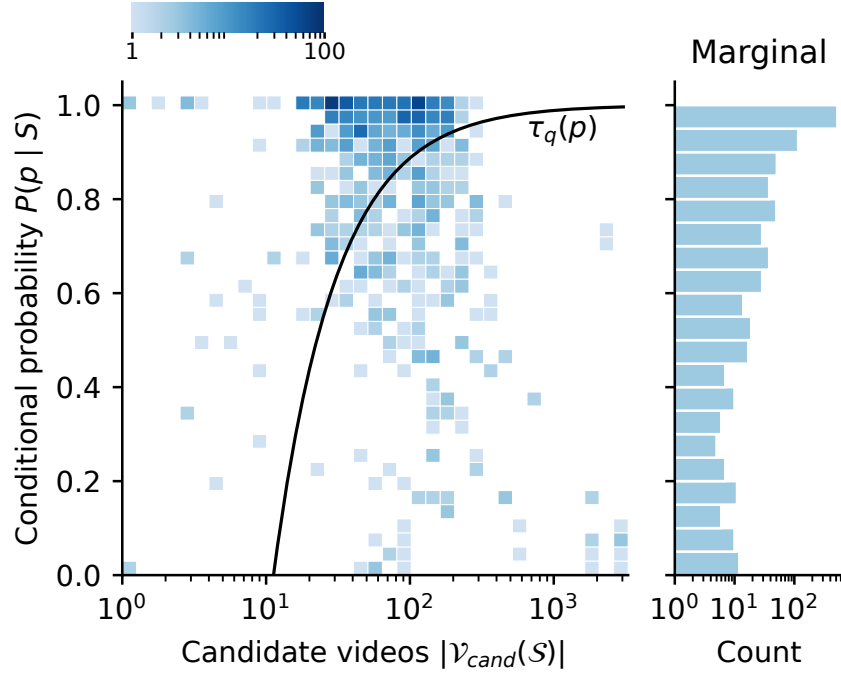


Figure 5.2: Empirical conditional probability $P(p \mid S)$ versus candidate count $|V_{cand}(S)|$ over 1,000 sampled decision contexts. The main panel shows a 2D histogram, with color indicating the number of contexts in each bin; the right panel shows the marginal distribution of $P(p \mid S)$, and the black curve shows the single-query threshold from Equation (5.2). Contexts below the curve are those for which instantiating a new filter is worthwhile under the single-query cost model.

predicates (Section 5.4), and retain filters whose validation F_1 is at least 0.7 so that noisy predicate outputs do not dominate the measured correlations. This leads to 74 object, 84 relationship, and 73 attribute predicates.

We then construct 1000 sampled *decision contexts* (q, p, S) , where q is a sampled query, p is the predicate being considered for instantiation, and $S \subseteq P_q \setminus \{p\}$ plays the role of P_q^+ . For each context, we apply the retained filters over the full corpus to compute $P(p \mid S)$ and the candidate count $|V_{cand}(S)|$. We compute $\tau_q(p)$ using Equation (5.2), with $C_T \approx \$0.45$ for labeling 500 images with **gpt-4.1** and $C_I \approx \$0.04$ for one oracle evaluation with **gpt-4.1** on 32 sampled frames per video.

Figure 5.2 plots $P(p \mid S)$ against $|V_{cand}(S)|$ over the sampled decision contexts. The main panel is a 2D histogram in which the color of each bin encodes the number of decision contexts falling into that bin, and the right panel is the marginal histogram of $P(p \mid S)$. The black curve shows the single-query threshold from Equation (5.2). We find that 72.7% of contexts lie above the threshold, indicating that instantiating a new filter is often not worthwhile under the single-query cost model. In addition, the marginal distribution of $P(p \mid S)$ is heavily skewed toward 1: 52.5% of contexts have $P(p \mid S) \geq 0.95$, and 33.6% exceed 0.99. These results motivate selecting which missing predicates to instantiate rather than training a filter for every predicate.

LLM-based filter selection

The empirical analysis above suggests that many predicates generated for the same query are highly correlated. VOCAL-SIFT therefore asks the LLM to judge whether a missing predicate is sufficiently implied by the predicates whose filters are already available, rather than to produce a calibrated probability estimate.

For each target predicate $p \in P_q^-$, VOCAL-SIFT provides the LLM with P_q^+ and considers two prompting variants.

Redundancy-based selection. The first variant asks whether p is almost always implied by P_q^+ , i.e., whether $P(p \mid P_q^+) \approx 1$. If so, VOCAL-SIFT skips instantiation; otherwise, it instantiates a filter for p (Section 5.4). This variant is simple and can be applied before constructing new filters, but it only skips predicates that are almost fully implied by P_q^+ and does not use $\tau_q(p)$.

Threshold-based selection. The second variant uses the cost model directly. For each missing predicate p , VOCAL-SIFT computes $\tau_q(p)$. If $\tau_q(p) \leq 0$, VOCAL-SIFT skips instantiation immediately because even a maximally selective filter would not recover its construction cost. Otherwise, it asks the LLM whether $P(p \mid P_q^+)$ is likely to exceed this threshold, skipping p if the answer is yes and instantiating it otherwise. This variant can skip predicates that are not fully redundant but still provide too little expected pruning

benefit. Its drawback is that computing $\tau_q(p)$ requires the filters for all predicates in P_q^+ to be instantiated and applied first, so VOCAL-SIFT cannot select and instantiate filters for different missing predicates fully in parallel. It also asks the LLM for a finer comparison than the redundancy-based policy.

We evaluate both variants in Section 5.5.4. For any predicate selected for instantiation, VOCAL-SIFT then constructs a lightweight filter using the procedure described in Section 5.4.

5.4 Filter instantiation

Once SELECTFILTERS decides that a predicate should be instantiated, VOCAL-SIFT realizes it as a *distilled predicate filter*: a lightweight classifier trained on frame embeddings using oracle VLM labels. This section describes the distillation procedure, a zero-shot embedding baseline used for comparison, and the object-grounding step used to represent relationship and attribute predicates.

Distilled predicate filters

Model distillation transfers the judgment of a strong oracle into a much cheaper student model [28, 36, 83, 196]. VOCAL-SIFT uses the oracle VLM as the teacher and trains a small multi-layer perceptron (MLP) on top of frozen frame embeddings as the student. Unlike VOCAL-UDF (Chapter 4), VOCAL-SIFT does not execute a DSL query or maintain object detections and tracks, so each filter operates over embeddings of entire frames.

The distillation procedure is fully automated. Given a predicate, VOCAL-SIFT first uses a pretrained vision-language encoder to embed both the predicate description and video frames into the same vector space. It retrieves frames with high text-image similarity as likely positives and frames with low similarity as likely negatives, then asks the oracle VLM to label each sampled frame as satisfying or not satisfying the predicate. These oracle-labeled frames form the training set for an MLP classifier, whose input is the corresponding

frame embedding. After training an initial classifier, VOCAL-SIFT refines it with active learning: it applies the classifier to additional frames, selects uncertain examples near the decision boundary, asks the oracle to label them, and retrain the classifier with the expanded labeled set.

Distilled predicate filters have two main advantages. First, once trained, inference is cheap because the filter is a compact discriminative model applied over precomputed frame embeddings. Second, the MLP learns a non-linear, predicate-specific decision boundary from oracle-labeled examples, allowing the filter to capture a richer decision boundary than a single text-image similarity score. The downside is that each new filter incurs a construction cost: sampling, oracle labeling, training, and validation. This cost is why VOCAL-SIFT uses `SELECTFILTERS` to instantiate only predicates that are expected to provide enough additional pruning benefit.

Baseline: zero-shot embedding filters

As a baseline, we also consider using the embedding space itself as the filter. A zero-shot embedding filter precomputes frame embeddings offline, embeds the predicate text into the same vector space, and scores each frame by its similarity to the predicate. The resulting scores rank frames by how well they match the predicate.

The main benefit of this baseline is its low setup cost. A new predicate can be evaluated without oracle labeling or model fitting. The challenge is that embedding similarity only produces a continuous score for each frame, but it does not by itself determine which frames should pass the filter. The system must choose a cutoff rule for these scores, and effective rules can vary across predicates and datasets. In our evaluation, we compare several such rules for zero-shot embedding filters (Section 5.5.2). This baseline therefore tests whether decomposition plus an off-the-shelf embedding space is enough, or whether VOCAL-SIFT’s learned predicate-specific filters are needed.

Both distilled predicate filters and zero-shot embedding filters require a textual representation of each predicate. For relationship and attribute predicates, an additional design

question is how the predicate is *grounded in* its participating objects during labeling and inference.

Object grounding for relationships and attributes

Relationship and attribute predicates require explicit referents to be used as filters. Without them, predicates such as **holding** or **red** are ambiguous because multiple object instances in the same frame may satisfy the description. VOCAL-SIFT therefore attaches each such predicate to its participating objects, turning an underspecified predicate into an object-anchored expression such as “player holding basketball” or “red jersey”.

Unlike VOCAL-UDF, which conditions distilled UDFs on cropped object patches and overlaid bounding boxes (Chapter 4), VOCAL-SIFT realizes grounding *purely through text*. This keeps filter instantiation lightweight—VOCAL-SIFT never needs to detect or track objects in the corpus—while still resolving the referent ambiguity of ungrounded predicates such as **holding** or **red**. For distilled predicate filters, grounded text makes the oracle labeling prompt more specific and therefore supports a more targeted classifier. For zero-shot embedding filters, it provides a more precise textual query for similarity search.

VOCAL-SIFT stores the resulting distilled filters in M , allowing future queries to reuse previous work and avoid repeated filter construction.

5.5 Evaluation

We implement a VOCAL-SIFT prototype in Python and configure it to use SigLIP2 as its visual encoder and GPT-4.1 as the oracle VLM for both decomposition prompting and distillation supervision. Per-predicate distilled filters are trained as MLPs over frozen SigLIP2 frame embeddings. Each candidate predicate filter is retained only if its validation F_1 exceeds a threshold ρ ; we report $\rho = 0.7$ unless otherwise specified. At evaluation time, learned filters and zero-shot embedding filters are applied at the frame level on the evaluation frame set. A video passes a filter if any of its frames passes that filter. Systems with multiple

Table 5.2: Datasets. “Classes” is the number of action classes in the corpus, “Queries” is the number of evaluated queries, and “TPs” is the total number of true positives across all evaluated queries.

Dataset	Classes	Queries	Train videos	Eval videos	TPs
ACTIONATLAS	581	50	448	447	71
VIDEONET	1087	50	3873	3070	142
K20-SKEW	20	20	504	494	494

filters combine these per-filter video decisions using the aggregation policy described below. The source code for VOCAL-SIFT is available at <https://github.com/uwdb/VOCAL-SIFT>.

We conduct an experimental evaluation of the VOCAL-SIFT prototype. First, we show that VOCAL-SIFT achieves a strong recall–cost trade-off on three video datasets, reducing oracle invocations by up to $13.6\times$ compared with directly applying the oracle VLM to every video (Section 5.5.2). Second, ablation studies show that multi-step decomposition, filter selection, and object-grounded filter instantiation each contribute to filtering quality or cost reduction (Section 5.5.3–Section 5.5.5). Third, we demonstrate that VOCAL-SIFT is robust to its tunable hyperparameters, while zero-shot embedding baselines depend more heavily on similarity decision rules (Section 5.5.6). Finally, we conduct a user study and show that VOCAL-SIFT’s automatic semantic decomposition reduces task completion time compared with an LLM-based human clarification baseline, without measurably degrading retrieval quality or task success (Section 5.5.7). We report all numbers for the filtering stage itself so that differences reflect filter design rather than downstream oracle quality.

5.5.1 Experimental setup

Dataset. We evaluate VOCAL-SIFT on three datasets shown in Table 5.2. We split videos in each dataset into training and evaluation sets. Training videos are used for filter instantiation, while evaluation videos are used for the evaluation. We further split the training

videos into an 80/20 train/validation split, where the validation split is used to compute validation F_1 for retaining candidate filters. Training is performed in an unsupervised manner, so we do not reveal the labels through the training process.

ACTIONATLAS [179] is a benchmark of fine-grained, domain-specialized action queries over sports videos. We use the action names as queries (e.g., “dirt jumping in mountain biking”), which are representative of the high-level queries targeted by VOCAL-SIFT: they involve implicit domain knowledge, exhibit substantial visual variability, and occur rarely in the corpus. The dataset contains 895 videos with 581 distinct actions, and one to five positive videos per action. For actions with at least two positive videos, we split the videos of each action evenly between the training and evaluation sets. For the remaining actions, which have only one positive video each, we pool their videos and split the pooled set evenly between the training and evaluation sets. We randomly sample 50 evaluation queries from actions with at least two positive videos in the training split.

VIDEONET [218] is a large-scale dataset of domain-specialized action queries collected from real-world videos. It spans a diverse set of domains and action categories, capturing substantial variation in how actions are visually instantiated across contexts. The action names provide natural high-level queries, requiring VOCAL-SIFT to derive filtering predicates from query text rather than receiving an explicit predicate decomposition. Compared to ACTIONATLAS, VIDEONET covers a broader range of domains beyond sports, such as crafts, food, and medical care. It contains 6943 videos with 1087 distinct actions, and four to 14 positive videos per action. For each action class, we use a 50/50 split of its videos for training and evaluation; videos from other action classes serve as non-matches for that query. We randomly sample 50 queries for evaluation.

We also evaluate on subsets of Kinetics700 [106], which comprises 700 human action classes. To create K20-SKEW, we sample 20 classes and introduce class imbalance following [53], generating a Zipfian distribution with parameter $s = 2$. Some human actions appear frequently in the resulting dataset and some very rarely. Videos are split evenly into training and evaluation sets. We use the 20 classes as queries. K20-SKEW stresses whether filtering

remains effective under highly imbalanced data distributions.

Metrics. Since VOCAL-SIFT performs filtering before oracle evaluation, we assess the videos retained by each system using true positive rate (TPR), true negative rate (TNR), and video retention rate. TPR measures the fraction of ground-truth positives retained after filtering and upper-bounds the end-to-end recall achievable by any downstream oracle. The video retention rate measures the fraction of videos that must be processed by the oracle and is therefore the main proxy for evaluation cost. We report TNR for completeness, but focus our analysis on the recall–cost trade-off between TPR and video retention rate. A desirable filtering therefore maximizes TPR while minimizing video retention rate.

Filter aggregation policy. Since semantic decomposition yields multiple filters per query, we must aggregate the per-filter video decisions to determine whether to retain a video. In our evaluation, we explore a relaxed intersection policy parameterized by a threshold $\alpha \in (0, 1]$: for a query q with filter set F_q , a video v is retained if and only if the fraction of filters that pass it satisfies $\frac{1}{|F_q|} \sum_{f \in F_q} \mathbf{1}[f(v) = 1] \geq \alpha$. Larger α retains fewer videos but is sensitive to any single filter with low recall, whereas smaller α improves robustness to individual filter errors by allowing a small fraction of filters to fail on a true positive. We use $\alpha = 0.8$ as the default because learned filters are imperfect and some necessary predicates may be weakly visible or transient in positive videos. Section 5.5.6 evaluates sensitivity to this choice.

Baselines. We compare VOCAL-SIFT against three baseline families of filtering methods. All baselines use the same frame-to-video rule as VOCAL-SIFT: a video passes a filter if any sampled frame passes it. This keeps the comparison focused on filter construction rather than video-level pooling.

Full-query distillation. Instead of decomposing a query, this baseline trains a single lightweight classifier on top of frozen SigLIP2 frame features to approximate the entire high-level query, using GPT-4.1 labels as supervision. This monolithic distillation baseline tests whether a single distilled filter can replace decomposition. We report the results at $\rho = 0.0$ (retaining all classifiers) and $\rho = 0.7$ (retaining classifiers with validation $F_1 \geq 0.7$).

Query embedding. This zero-shot embedding family embeds the full NL query and the video frames into the same vector space, then filters videos using text-image similarity. Because similarity produces a continuous score for each frame, the system must choose a rule for converting these scores into filter decisions. We evaluate three similarity decision rules. *Percentile* keeps frames in the top $p\%$ of the query’s frame-similarity distribution, with $p \in \{1\%, 5\%, 10\%\}$. *Binary* constructs a positive prompt stating that the frame contains the queried event and a negative prompt stating that it does not, then keeps frames that are closer to the positive prompt. *Kneedle* sorts frames by similarity score and uses the Kneedle algorithm [180] to choose the cutoff at the maximum curvature of the score curve.

Predicate embedding. This zero-shot embedding family uses the same decomposed predicates as VOCAL-SIFT, but replaces learned distilled filters with embedding-similarity filters applied independently to each predicate, as described in Section 5.4. For each predicate, frame-level similarity scores are converted into filter decisions using the same three rules as above. Predicate-level decisions are aggregated using the same aggregation policy as in VOCAL-SIFT. This baseline evaluates whether distilled predicate filters improve over zero-shot embedding filters when the predicate decomposition is held fixed.

5.5.2 End-to-end filtering performance

Filtering effectiveness

We first evaluate the overall effectiveness of VOCAL-SIFT’s filtering approach by executing each query and measuring the TPR, TNR, and video retention rate of the retained videos. A desirable filter lies in the upper-left of the recall-cost plane: high TPR with low video retention rate. We compare VOCAL-SIFT against the baselines described above: full-query distillation (FQD), zero-shot query embedding (QE), and zero-shot predicate embedding (PE). For QE and PE, the PERCENTILE, BINARY, and KNEEDLE variants differ only in how they convert frame-level similarity scores into filter decisions.

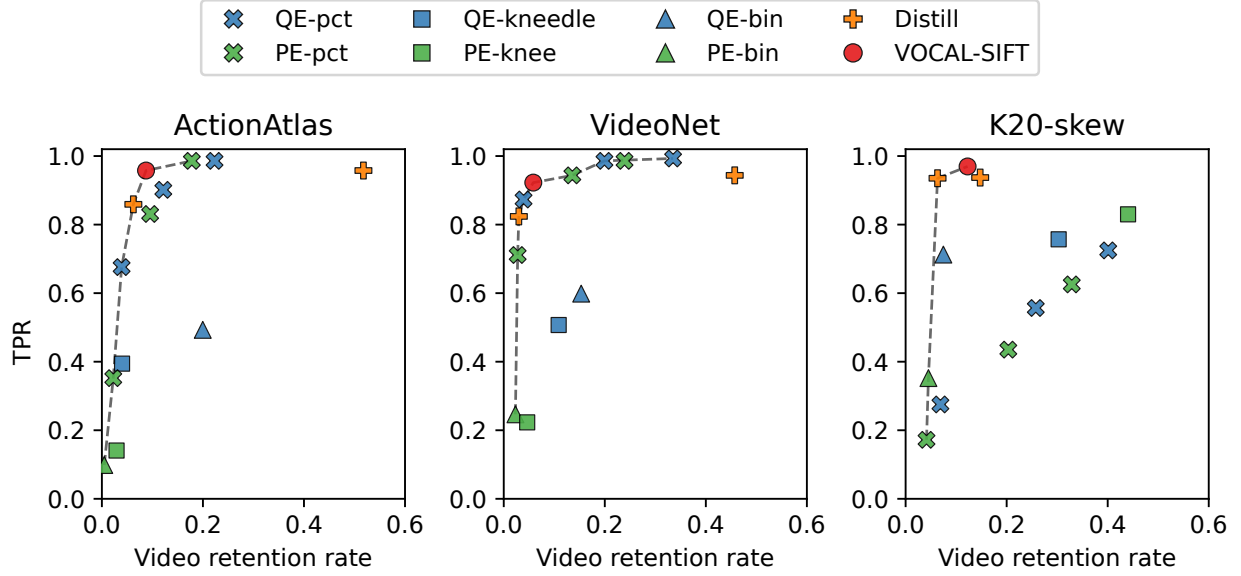


Figure 5.3: End-to-end filtering performance across datasets. Higher TPR and lower video retention rate are better.

Table 5.3: TPR, TNR, and video retention rate across datasets and methods. We include methods that achieve $\text{TPR} \geq 0.9$ on at least one dataset. For each method–dataset pair, entries are grayed out when the method fails to reach $\text{TPR} \geq 0.9$ on that dataset. FQD: full-query distillation; QE: query embedding; PE: predicate embedding.

Family	Method	ACTIONATLAS			VIDEONET			K20-SKEW		
		TPR	TNR	Retention rate	TPR	TNR	Retention rate	TPR	TNR	Retention rate
<i>FQD</i>	$\rho = 0.7$	0.96	0.48	51.8%	0.94	0.54	45.7%	0.94	0.89	14.8%
	$\rho = 0.0$	0.86	0.94	6.2%	0.82	0.97	3.0%	0.94	0.98	6.3%
<i>QE</i>	percentile, $p = 10\%$	0.99	0.78	22.3%	0.99	0.67	33.5%	0.72	0.62	40.1%
	percentile, $p = 5\%$	0.90	0.88	12.2%	0.99	0.80	19.9%	0.56	0.76	25.8%
<i>PE</i>	percentile, $p = 10\%$	0.99	0.82	17.8%	0.99	0.76	23.9%	0.63	0.69	32.9%
	percentile, $p = 5\%$	0.83	0.91	9.5%	0.94	0.86	13.6%	0.44	0.81	20.3%
Ours	with inst., $\rho = 0.7$	0.96	0.92	8.7%	0.92	0.94	5.8%	0.97	0.92	12.3%

Figure 5.3 summarizes end-to-end filtering quality across the three evaluation datasets

by plotting TPR against video retention rate, while Table 5.3 reports the same metrics together with TNR. Each point in the figure corresponds to one configuration of a method. **VOCAL-SIFT achieves a strong recall–cost trade-off across datasets.** In Figure 5.3, VOCAL-SIFT lies on the Pareto frontier for all three datasets, maintaining high TPR while substantially reducing the fraction of videos passed to the oracle. The table shows the same pattern: VOCAL-SIFT reaches TPRs of 0.96, 0.92, and 0.97 on ACTIONATLAS, VIDEONET, and K20-SKEW, respectively, while retaining only 8.7%, 5.8%, and 12.3% of videos per query on average.

Full-query distillation can achieve high TPR only by becoming conservative. When all trained classifiers are retained ($\rho = 0.0$), the video retention rate is low, but TPR drops by 0.09, 0.03, and 0.10 relative to VOCAL-SIFT on ACTIONATLAS, VIDEONET, and K20-SKEW, respectively, because unreliable classifiers can filter out true positives. When only classifiers with validation $F_1 \geq 0.7$ are retained, FQD recovers high TPR, but must pass many more videos to the oracle: $5.9\times$, $7.8\times$, and $1.1\times$ more than VOCAL-SIFT on the three datasets. This trade-off reflects the weakness of monolithic distillation, where a single lightweight model may not reliably capture the visual variability of a high-level query.

Zero-shot embedding filters are sensitive to similarity decision rules and dataset distribution. Both QE and PE must convert frame-level similarity scores into filter decisions. BINARY and KNEEDLE generally retain too few true positives, yielding low TPR across datasets. PERCENTILE is more competitive, but its effectiveness depends strongly on the corpus distribution and on the chosen percentile. For example, QE-PERCENTILE performs competitively on VIDEONET, but degrades substantially on K20-SKEW, where class imbalance makes a fixed percentile value unreliable across queries. PE-PERCENTILE follows a similar pattern: it can achieve high TPR on ACTIONATLAS and VIDEONET, but often retains substantially more data than VOCAL-SIFT and degrades on K20-SKEW. No single percentile setting dominates across datasets. We analyze this sensitivity in more detail in Section 5.5.6. These results suggest that decomposition alone is insufficient because high-level query filtering also benefits from learned predicate-specific filters that adapt to the

Table 5.4: Estimated dollar cost for processing all queries in each dataset. Pre-evaluation cost includes query parsing, decomposition, filter selection, and filter instantiation. Evaluation cost assumes \$0.04 per retained video.

Dataset	Method	Pre-eval cost	Eval cost	Total cost	Total cost reduction
ACTIONATLAS	Direct VLM	—	\$894	\$894	1.0×
	Full distill	\$23	\$463	\$486	1.8×
	VOCAL-SIFT	\$76	\$78	\$154	5.8×
VIDEONET	Direct VLM	—	\$6140	\$6140	1.0×
	Full distill	\$23	\$2807	\$2830	2.2×
	VOCAL-SIFT	\$91	\$359	\$450	13.6×
K20-SKEW	Direct VLM	—	\$413	\$413	1.0×
	Full distill	\$8	\$61	\$69	6.0×
	VOCAL-SIFT	\$26	\$51	\$77	5.4×

visual distribution of each predicate.

Estimated dollar cost

Table 5.4 estimates the dollar cost of running all queries across datasets and methods. We compare three methods: direct VLM that evaluates every query on every evaluation video, full-query distillation with $\rho = 0.7$, and VOCAL-SIFT. We do not consider other methods due to their insufficient TPR on at least one dataset. The total cost comprises pre-evaluation cost and oracle VLM evaluation cost on the retained videos. Pre-evaluation cost is the cost paid before the final oracle pass, including query parsing, semantic decomposition, filter selection, and filter construction. We assume an oracle cost of $C_I \approx \$0.04$ per video, the same value used in our cost model in Section 5.3, which corresponds to one oracle evaluation with `gpt-4.1` on 32 sampled frames per video.

VOCAL-SIFT is up to 13.6× cheaper than the direct VLM approach. Direct VLM performs no filtering and therefore evaluates every query against every evaluation video. On ACTIONATLAS and VIDEONET, VOCAL-SIFT is also 3.2× and 6.3× cheaper than full-query distillation, respectively. Although VOCAL-SIFT incurs a higher pre-evaluation cost,

Table 5.5: Semantic decomposition variants.

Dataset	Decomposition	TPR	Retention rate
ACTIONATLAS	Direct	0.92	20.1%
	VOCAL-SIFT	0.96	8.7%
VIDEONET	Direct	0.70	32.5%
	VOCAL-SIFT	0.92	5.8%
K20-SKEW	Direct	0.98	17.8%
	VOCAL-SIFT	0.97	12.3%

primarily from label annotation during filter instantiation, it substantially lowers evaluation cost by invoking the expensive oracle VLM on far fewer videos. On K20-SKEW, VOCAL-SIFT remains $5.4\times$ cheaper than direct VLM, but is $1.1\times$ more expensive than full-query distillation; these queries are more coarse-grained, so full-query distillation already performs well, and semantic decomposition provides less additional benefit.

5.5.3 Semantic decomposition

Table 5.5 compares VOCAL-SIFT’s *multi-step decomposition* with a *direct decomposition* baseline. As described in Section 5.2, VOCAL-SIFT first generates candidate predicates without access to the predicate library, then aligns them with existing library entries when their meanings match, and finally reconciles repeated LLM generations into a single predicate set. The direct decomposition baseline collapses these steps into a single LLM call that is asked to produce the final predicate set for the query in one step. **VOCAL-SIFT’s multi-step decomposition retains fewer videos.** Compared with direct decomposition, VOCAL-SIFT reduces video retention by $2.3\times$ on ACTIONATLAS, $5.8\times$ on VIDEONET, and $1.5\times$ on K20-SKEW. TPR is also higher on ACTIONATLAS and VIDEONET; on K20-SKEW, direct decomposition has slightly higher TPR but retains substantially more videos. Direct decomposition performs worse for two reasons. First, when reuse is decided in the

Table 5.6: Filter selection policies.

Dataset	Policy	TPR	Retention rate	Trained filters	Total cost
ACTIONATLAS	All predicates	0.92	8.5%	206	\$171
	Redundancy	0.96	8.7%	163	\$154
	Threshold	0.96	9.0%	91	\$125
VIDEONET	All predicates	0.89	5.2%	254	\$440
	Redundancy	0.92	5.8%	192	\$450
	Threshold	0.92	5.4%	138	\$400
K20-SKEW	All predicates	0.97	12.2%	69	\$78
	Redundancy	0.97	12.3%	64	\$77
	Threshold	0.97	12.7%	33	\$67

same one-shot prompt as decomposition, the LLM may try to express the query using the closest available predicate even when the match is only approximate, or omit a predicate when the exact concept is absent from the library. For example, for “ladder escape in yo-yo”, direct decomposition reuses a **pen-spinning** predicate because **yo-yo** is absent from the library, whereas VOCAL-SIFT produces predicates for **yo-yo** and **string**. Second, a one-shot prompt has no independent decompositions to reconcile, so it is more likely to omit necessary constraints or keep spurious ones. For “shoot dance”, direct decomposition includes the spurious predicate **other_arm_bent**; VOCAL-SIFT drops it because it appears in only one candidate decomposition.

5.5.4 Filter selection

As discussed in Section 5.3, VOCAL-SIFT supports two filter selection policies: redundancy-based selection and threshold-based selection. We compare these policies with the *all predicates* baseline, which instantiates every missing predicate. Table 5.6 shows the results. **Both filter selection policies reduce instantiation cost while preserving filtering quality.** Compared with instantiating all predicates, redundancy-based selection

trains 21%, 24%, and 7% fewer filters on ACTIONATLAS, VIDEONET, and K20-SKEW, respectively, while retaining similar TPR and video retention rate. This suggests that many decomposed predicates add limited marginal pruning power once related predicates for the same query are already enforced. Skipping such predicates reduces filter generation cost and avoids compounding errors from redundant or overly strict learned filters. In estimated total dollar cost, redundancy-based selection improves over all predicates on ACTIONATLAS and K20-SKEW by \$17 and \$1, respectively. On VIDEONET, it is \$10 more expensive because it retains slightly more videos, but this comes with a TPR improvement from 0.89 to 0.92.

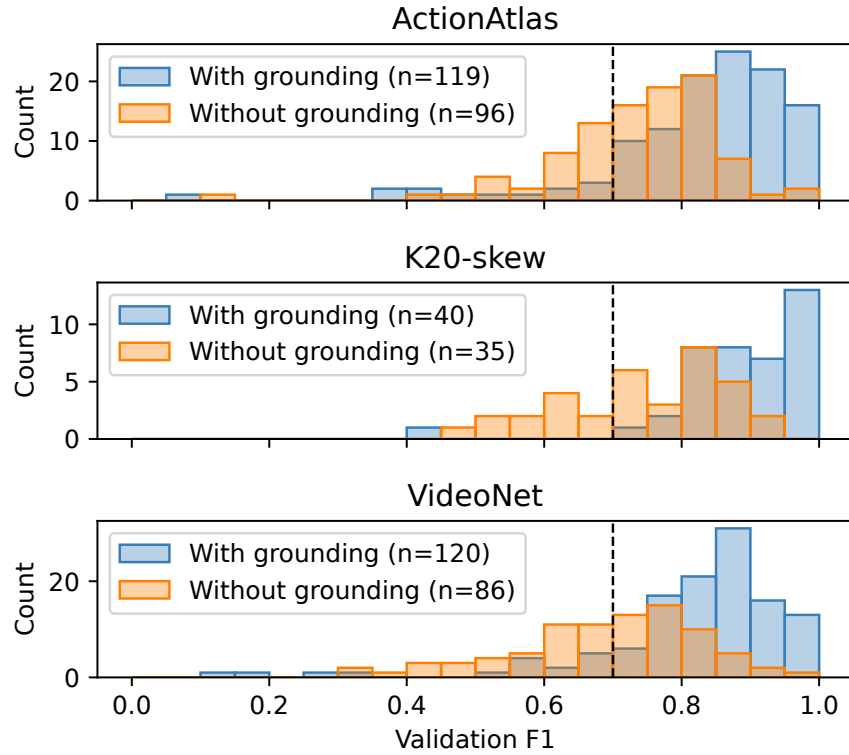
Threshold-based selection further reduces total cost. Relative to redundancy-based selection, threshold-based selection trains 44%, 28%, and 48% fewer filters on ACTIONATLAS, VIDEONET, and K20-SKEW, respectively, while preserving TPR and changing the video retention rate by at most 0.4%. This reduces total cost by 19%, 11%, and 13%, respectively. These savings support the cost model in Section 5.3: some predicates are not worth instantiating even when they are not fully redundant, because their expected pruning benefit does not outweigh the one-time filter generation cost. The drawback is that computing the threshold requires the filters in P_q^+ to be instantiated and applied first, so filter selection and filter instantiation cannot be fully parallelized across predicates. In practice, we use redundancy-based selection as the lower-latency default, and threshold-based selection when minimizing total dollar cost is more important than reducing query latency.

5.5.5 Filter instantiation

Table 5.7 compares three variants of predicate filter construction: *Objects only*, which drops relationship and attribute predicates; *Ungrounded*, which keeps relationship and attribute filters but trains them without object grounding; and *Grounded*, the full VOCAL-SIFT instantiation with object, relationship, and attribute filters. **Object filters alone retain many more videos.** Object-only filtering has slightly higher TPR than the full grounded variant, which is expected because it uses a strict subset of the predicates and

Table 5.7: Filter instantiation variants.

Dataset	Variant	TPR	Retention rate
ACTIONATLAS	Objects only	1.00	10.7%
	Ungrounded	0.99	11.9%
	Grounded	0.96	8.7%
VIDEONET	Objects only	0.99	9.4%
	Ungrounded	0.94	13.3%
	Grounded	0.92	5.8%
K20-SKEW	Objects only	0.99	30.2%
	Ungrounded	0.99	24.6%
	Grounded	0.97	12.3%

Figure 5.4: Validation F_1 distributions for relationship and attribute filters with and without object grounding.

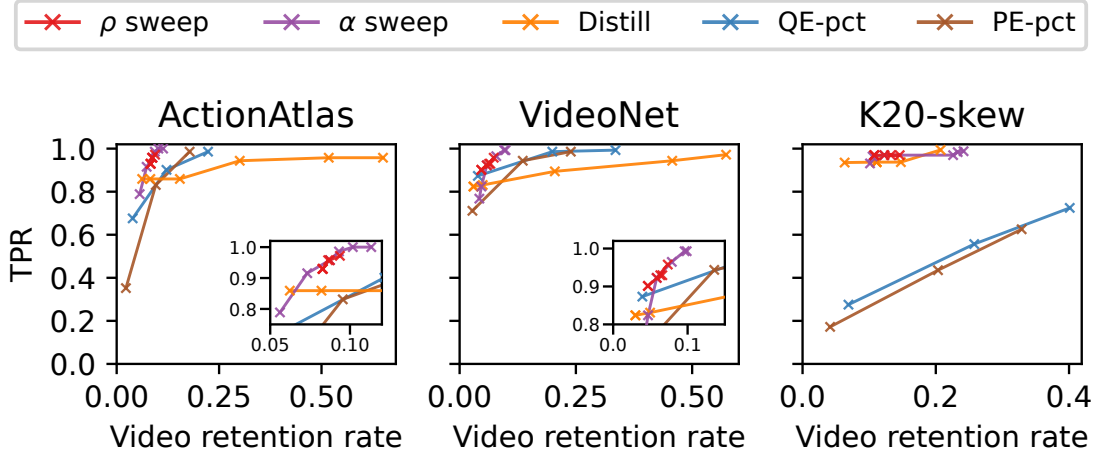


Figure 5.5: Threshold sensitivity across datasets.

therefore rejects fewer positives. However, it prunes substantially fewer videos: object-only filtering retains $1.2\times$, $1.6\times$, and $2.5\times$ more videos than the full grounded variant on ACTIONATLAS, VIDEONET, and K20-SKEW, respectively.

Grounding relationship and attribute filters improves both pruning and filter quality. Training relationship and attribute filters without object grounding retains more videos than the full grounded variant on all datasets: $1.4\times$ more on ACTIONATLAS, $2.3\times$ more on VIDEONET, and $2.0\times$ more on K20-SKEW. Figure 5.4 explains this gap at the model level: relationship and attribute filters trained without object grounding have worse validation F_1 distributions than their grounded counterparts. Thus, merely adding non-object predicates is not enough; grounding relationship and attribute filters in their participating objects produces higher-quality filters and filters out more non-matching videos while preserving recall.

5.5.6 Threshold sensitivity

VOCAL-SIFT exposes two tunable thresholds that govern its filtering behavior: the

validation F_1 threshold ρ , which determines which trained predicate filters are retained for use at evaluation time, and the filter aggregation threshold α , which determines the fraction of retained predicate filters that must pass a video for it to pass the filter. We evaluate both thresholds across a broad range. Figure 5.5 evaluates the sensitivity of filtering quality to threshold choices. For VOCAL-SIFT, we vary $\rho \in \{0.0, 0.1, 0.3, 0.5, 0.7, 0.8\}$ and $\alpha \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. For comparison, we include full-query distillation with the same ρ values, as well as percentile-thresholded QE and PE baselines with $p \in \{0.01, 0.05, 0.1\}$. Points closer to the upper-left indicate better recall–cost trade-offs.

VOCAL-SIFT achieves the best recall–cost trade-off across threshold settings.

Its results across ρ and α settings lie mostly on the Pareto frontier, showing that the learned predicate filters are robust to reasonable threshold choices. The main exception occurs when α approaches 1.0, which requires every necessary predicate filter to pass a video for it to be retained. This suggests that less strict aggregation is useful in practice, because even necessary predicates may be partially visible, transient, or missed by an imperfect visual filter in otherwise positive videos. Other methods exhibit less favorable trade-offs: full-query distillation either retains many more videos to recover high TPR or has lower TPR at more aggressive settings, while the percentile-thresholded zero-shot embedding baselines trace worse trade-off curves and fail completely on the imbalanced K20-SKEW setting. Overall, VOCAL-SIFT is less sensitive to its threshold settings than the zero-shot embedding baselines are to their similarity decision rules, suggesting that the default values used elsewhere in this section are not the result of careful tuning.

5.5.7 User study: VOCAL-SIFT vs. human clarification

Asking users clarification questions via an LLM is a common way to resolve query ambiguity [114, 121, 237, 107]. We compare VOCAL-SIFT’s automatic semantic decomposition (AUTO) against a *human clarification* baseline through a user study, measuring task completion time, retrieval quality, and subjective feedback.

We recruited 40 participants from the crowdsourcing platform Prolific [156] for a within-subjects study on four VIDEONET queries: “Arm roll in bartending” (BARTENDING), “Biellmann spin in figure skating” (SKATING), “Reverse vault in parkour” (PARKOUR), and “Alley-oop in basketball” (BASKETBALL). Each participant completed all four tasks, with each task assigned to one of two conditions. The first condition, *AUTO*, runs VOCAL-SIFT’s automatic semantic decomposition (Section 5.2) without any user interaction beyond the initial query. The second condition, *human clarification*, invokes an LLM that decides whether the initial query is ambiguous and, if so, asks the user one or more free-form questions to clarify the event’s visual logic before filters are constructed. Task order, condition order, and the task-condition assignment were *counterbalanced*: across participants we varied which task was assigned to which condition and the sequence in which the tasks were presented, in a Latin-square-like rotation.

Under the human-clarification condition, the LLM may or may not ask for clarifications, depending on how specific the user’s initial query is. For some participants the system asks several questions, while for others it accepts the initial query and proceeds directly to filter construction. We therefore report this condition in two ways: over all attempts assigned to that condition (HUMAN-ALL), and over the subset of attempts in which the system actually asked at least one clarification question (HUMAN-ASKED). The latter is a diagnostic subset that isolates outcomes when clarification occurs, rather than a separately randomized condition. For aggregate comparisons, we report significance using two-sided Wilcoxon signed-rank tests on participant-level paired aggregates, with one aggregate value per participant per condition. Per-task results are shown descriptively.

Participants first watched a short instructional video demonstrating the interface and task. For each task, they were shown the action name and three positive example videos, and were asked to find one additional matching video from the corpus. They could play the example videos and use external resources if needed to help them understand the action. Participants began by submitting an NL query. The system then parsed the query and derived predicates. In the human-clarification condition, an LLM asked free-form questions

to clarify the event’s visual logic when needed; in AUTO, the LLM decomposed the query into necessary predicates without user interaction (Section 5.2). Next, the system trained filters for the retained predicates, skipped redundant ones per Section 5.3, and returned a list of retrieved videos that passed the filters. Participants could watch videos, select an answer, toggle filters, and revise the query to start a new iteration. A task was *successful* if the selected video was a ground-truth positive. After each task, participants completed a questionnaire.

Figure 5.6a compares task completion time across conditions. **Automatic decomposition significantly reduces task completion time compared to human clarification.** Median completion time is 610s for AUTO, compared with 754s for HUMAN-ALL; the median paired reduction is 67.9s, which is significant under a two-sided Wilcoxon signed-rank test on participant-level paired aggregates ($p = 0.001$). The diagnostic HUMAN-ASKED subset has a higher median completion time of 780s, indicating that attempts involving clarification are slower. Per-task results show that the largest differences occur on SKATING and PARKOUR. Figure 5.6b breaks completion time down by pipeline stage. **The time savings come from both eliminating clarification and reducing video examination time.** In the human-clarification condition, clarification is requested in 64/80 attempts; when clarification is asked, the clarification step takes a mean of 147s. AUTO eliminates this step and replaces it with an automatic LLM decomposition stage. Because AUTO typically derives more predicates than users specify, it incurs higher *Filter training* time (226s per iteration, compared with 175s for HUMAN-ALL and 172s for HUMAN-ASKED). However, filter training is system-driven and parallelizable across predicates. Participants spend less time in *Video examination* with AUTO: 169s per iteration, compared with 206s for HUMAN-ALL.

Filtering quality and final task success are similar across conditions. Figure 5.7 reports the fraction of ground-truth positives among retrieved, viewed, and selected videos. AUTO produces a similar or slightly higher fraction of positives among retrieved videos (7.1% vs. 4.3%, $p = 0.103$) and viewed videos (7.1% vs. 5.5%, $p = 0.521$), but these differences are not statistically significant. Task success is numerically lower for AUTO than

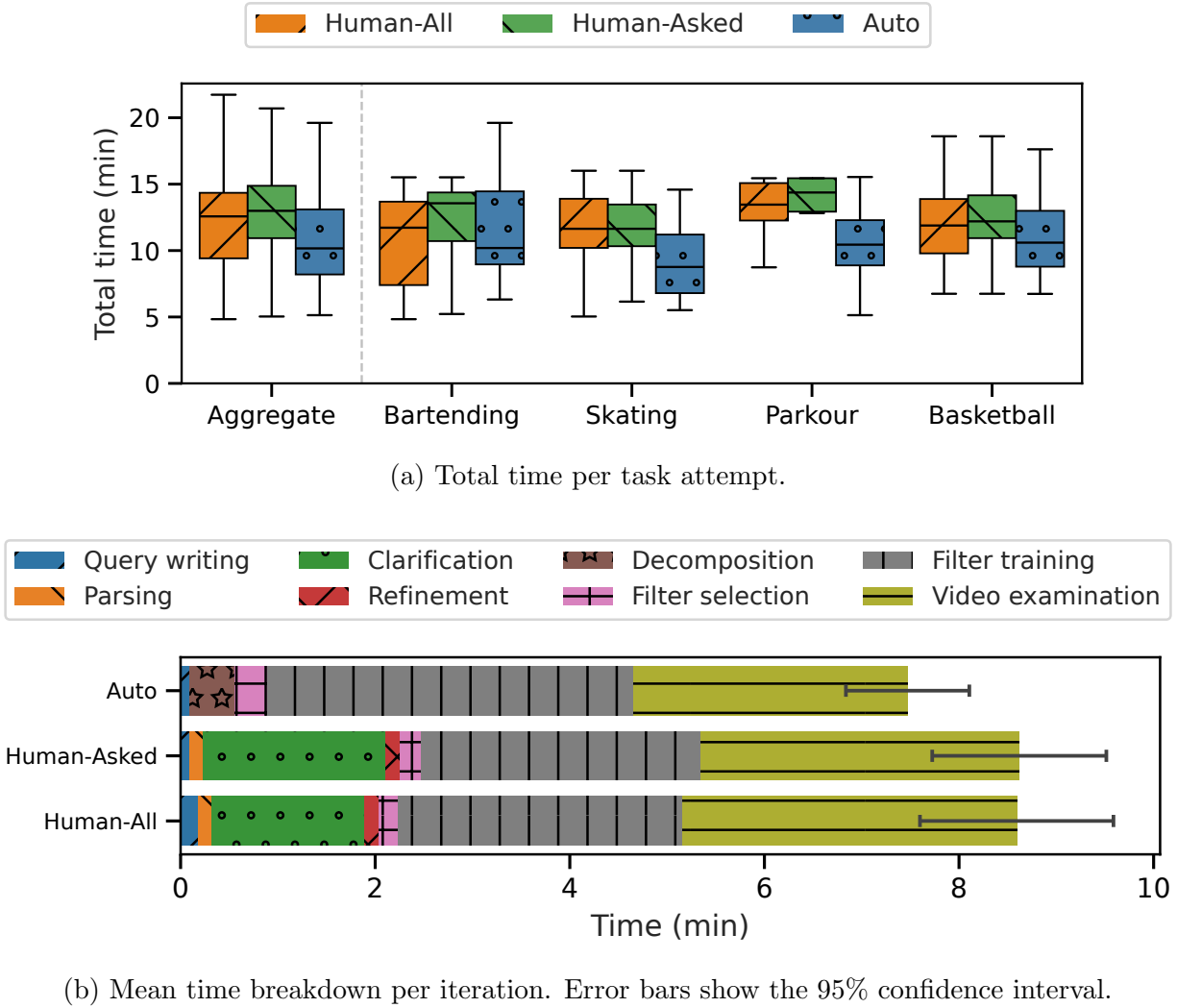


Figure 5.6: User study task completion time.

for HUMAN-ALL, but the difference is also not statistically significant (45.0% vs. 51.2%, $p = 0.317$), indicating that automatic decomposition reduces interaction time without measurably degrading retrieval quality or final task outcomes.

Subjective responses are similar across conditions. The questionnaire collects participants’ subjective ratings and NASA-TLX ratings (excluding physical demand due to irrelevance) on perceived workload, ease of use, and confidence, all on a 1–7 Likert scale. Fig-

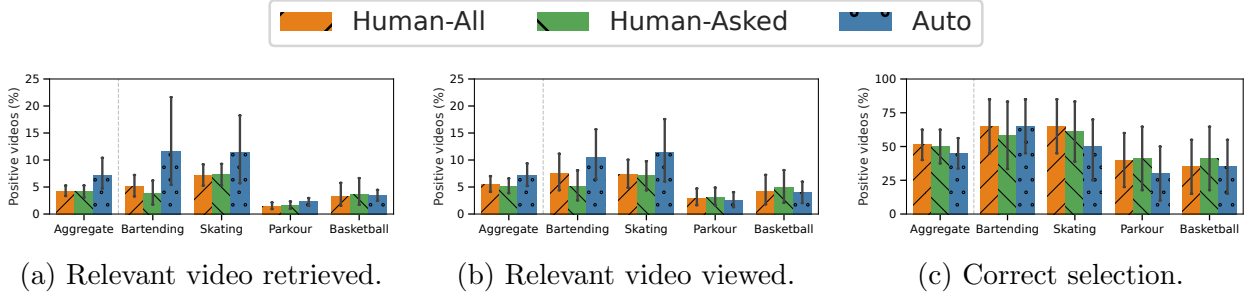


Figure 5.7: User study outcomes by condition. Human clarification and automatic decomposition produce statistically indistinguishable filtering quality and success rates. Error bars show the 95% confidence interval.

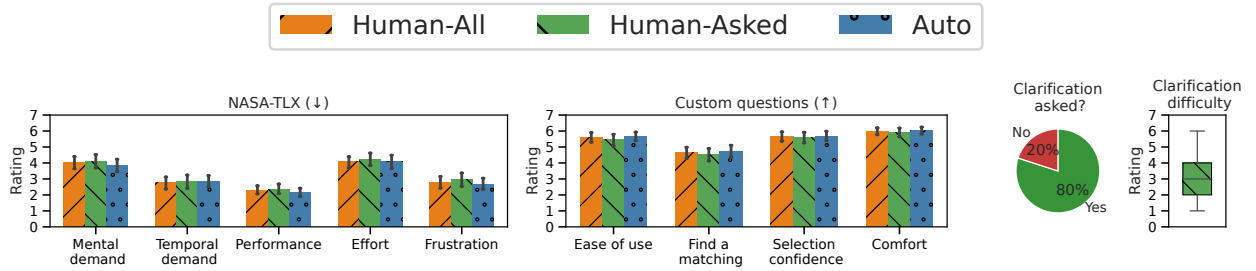


Figure 5.8: User study questionnaire responses. NASA-TLX and custom Likert responses are similar across conditions. Participants rate the difficulty of answering clarification questions as moderately difficult. Error bars show the 95% confidence interval.

Figure 5.8 shows that the conditions are subjectively similar: differences are at most 0.28 Likert points across items, and none are statistically significant under participant-level paired tests. For HUMAN-ASKED, participants rated the difficulty of answering clarification questions at a mean of 3.37 and a median of 3, indicating modest but non-negligible cognitive load.

5.6 Summary

In this chapter, we presented VOCAL-SIFT, a video query processing system that enables efficient high-level NL querying over large video corpora. VOCAL-SIFT extends the proxy-model principle from explicitly specified predicates to open-ended, high-level queries by au-

tomatically discovering *necessary predicates*—visually detectable conditions that must hold whenever the high-level event occurs—and realizing them as lightweight filters that prune the corpus before invoking an expensive oracle VLM. VOCAL-SIFT proposes a multi-step semantic decomposition process that independently generates candidate predicates, aligns them against a reusable predicate library, and reconciles LLM outputs; a filter selection method that uses LLM-based correlation estimation to skip instantiating predicate filters whose pruning benefit does not justify their construction cost; and a filter instantiation procedure that converts the resulting predicates into lightweight classifiers.

Across three benchmarks—ActionAtlas, VideoNet, and K20-SKEW—VOCAL-SIFT achieves a strong recall–cost trade-off and is up to $13.6\times$ cheaper than directly invoking the oracle VLM. A user study further shows that VOCAL-SIFT’s automatic decomposition reduces task completion time relative to interactive human clarification without degrading retrieval quality or task success.

Chapter 6

INTERFACES FOR AI-ASSISTED VIDEO QUERYING

In the preceding three chapters, we presented three systems that reduce the user effort required to query video data in complementary ways. EQUI-VOCAL reduces the burden of writing compositional queries by synthesizing queries from examples. VOCAL-UDF lets users issue compositional queries in NL without manually providing or implementing all required visual modules. VOCAL-SIFT supports high-level NL queries without requiring users to explicitly specify the visual logic needed to evaluate them.

This chapter presents the user interfaces implemented for these systems. These interfaces are research prototypes built to demonstrate each system’s query workflow. In each system, the interface is the gateway between user intent and query processing, determining what the user must provide, when the user is asked to intervene, and which intermediate outputs the user can inspect. The following sections describe each interface in detail.

The interfaces of the three systems are available at <https://github.com/uwdb/equi-vocal-demo>, <https://github.com/uwdb/VOCAL-UDF> and <https://github.com/uwdb/VOCAL-SIFT>.

6.1 *EQUI-VOCAL: query by example*

EQUI-VOCAL’s interface, shown in Figures 6.1 to 6.3, was developed for the EQUI-VOCAL demonstration paper [234]; we used a similar interface in the EQUI-VOCAL user study described in Section 3.4.4. The interface centers on query-by-example interaction. A user begins by configuring the dataset and available UDFs, selecting a video task, and providing a small number of initial positive and negative examples for the event of interest (Figure 6.1). The system then runs query synthesis and active learning, selecting additional video segments

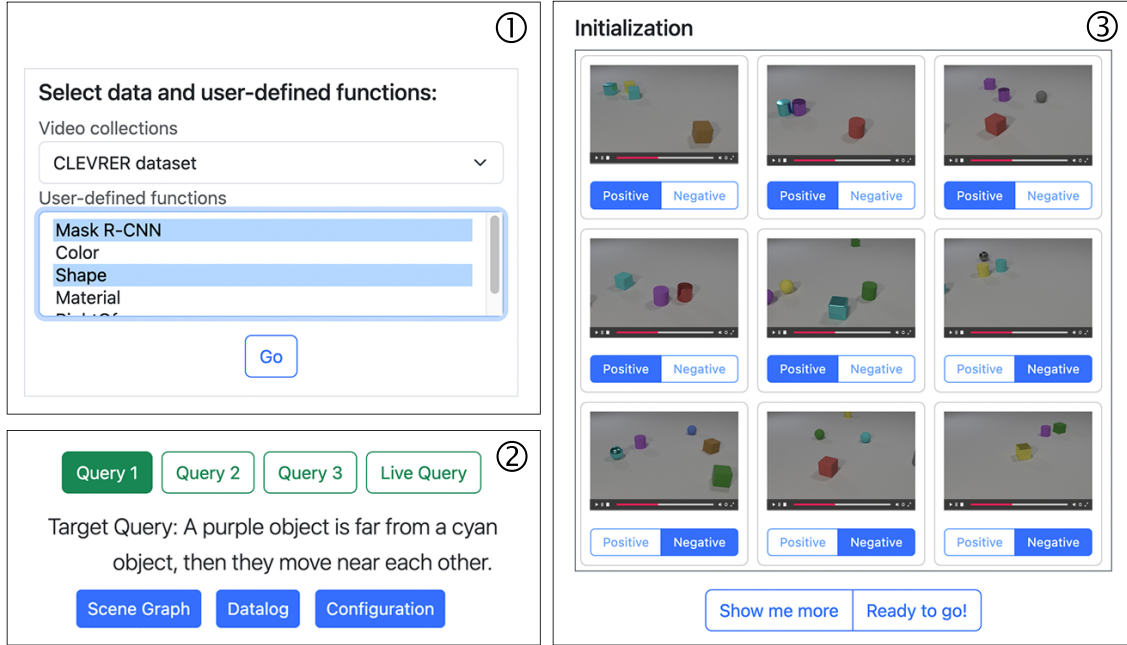


Figure 6.1: EQUI-VOCAL data preparation and query initialization interface. The user configures the dataset and available UDFs, selects a task, and provides initial positive and negative examples for query synthesis.

that are informative for disambiguating candidate queries and asking the user to label them (Figure 6.2). For demonstration purposes, the interface also shows the current best query’s predictions on a held-out test set, although such held-out labels would not normally be available to users in practice. The interface further includes scene graph and Datalog views of the target query for the demo task, illustrating the kind of query inspection panes that could also be shown for the current best query (Figure 6.3). This design matches the algorithmic structure of Chapter 3: the user does not write the compositional query, but their examples and labels guide the search through the query language.

6.2 VOCAL-UDF: querying with generated UDFs

For VOCAL-UDF, we implemented a lightweight command-line interface to demonstrate how the system handles queries whose required visual modules are not all available. The

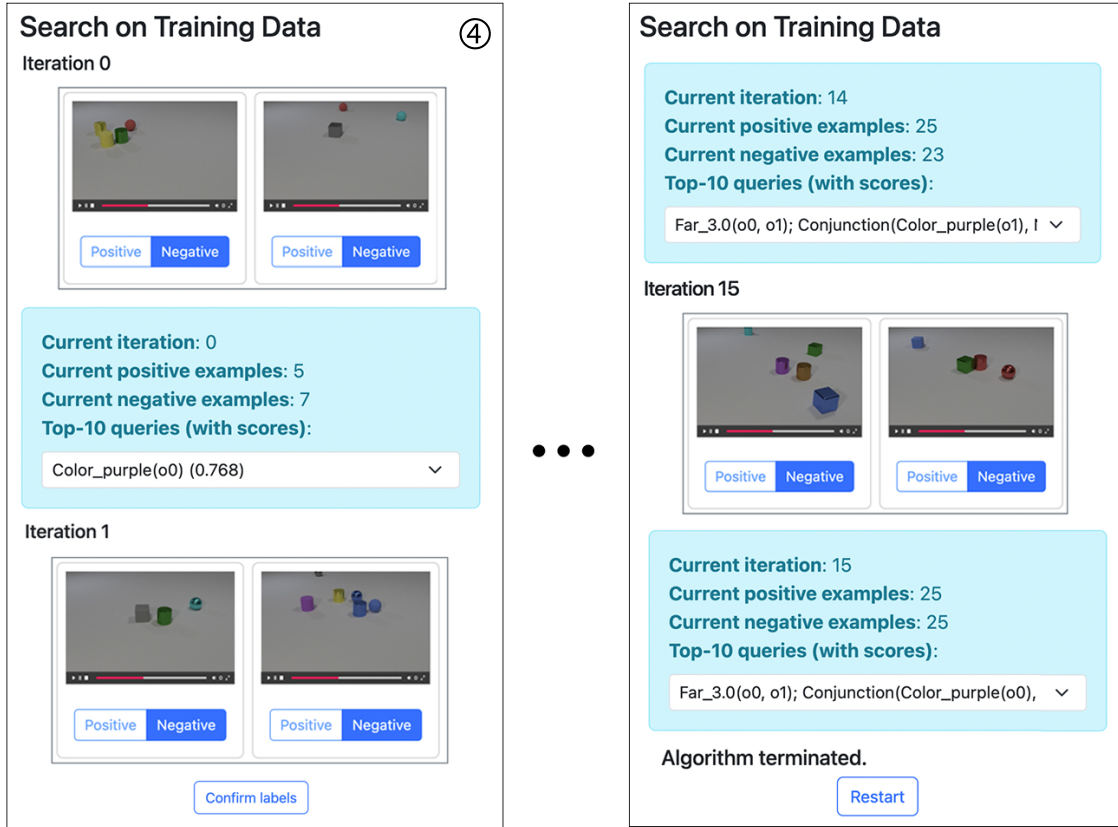


Figure 6.2: EQUI-VOCAL labeling interface with history. The system asks the user to label informative video segments selected during active learning, while the interface keeps a history of previous labels.

interaction begins with the user entering an NL query and selecting which registered UDFs are available for the query (Figure 6.4). The system first attempts to parse the query into the DSL introduced in Chapter 4. If the query refers to predicates that the available UDF catalog cannot support, VOCAL-UDF proposes new UDFs and generates candidate implementations (Figure 6.5).

The example shown in the interface requires two missing predicates, `color_red` and `near`. VOCAL-UDF generates multiple candidates for each missing UDF, including program-based UDFs produced by an LLM and distilled-model UDFs distilled from strong pretrained models. It then enters a selection loop in which the user provides binary labels for system-

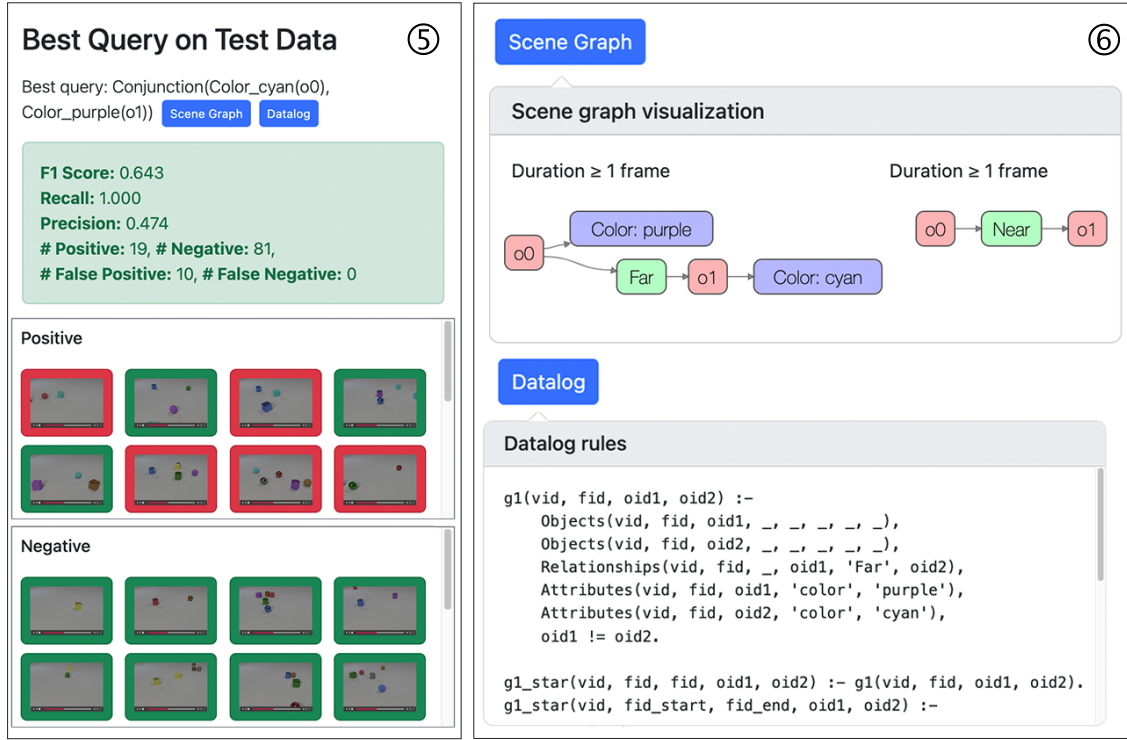


Figure 6.3: EQUI-VOCAL query inspection interface. The interface shows the current best query’s predictions and, for the demo task, scene graph and Datalog views of the target query to help users inspect query components.

selected examples (Figure 6.6): for `color_red`, the user judges whether a highlighted object is red; for `near`, the user judges whether the object highlighted in red is near the object highlighted in blue (not shown in the figure). These labels allow VOCAL-UDF to select the candidate implementation whose behavior is best aligned with the user’s intent.

This interface exposes a different kind of interaction than EQUI-VOCAL. The user is not labeling whole video events to synthesize a query; instead, they are helping the system choose the meaning and implementation of a missing predicate. Once the selected UDFs are added to the catalog, VOCAL-UDF reparses the original query, compiles it to SQL, executes it, and returns matching videos (Figure 6.7).

Beyond the initial NL query, the main user effort in VOCAL-UDF is this predicate-level labeling during UDF selection. A fully automatic variant could remove this effort by asking

Welcome to VOCAL-UDF CLI!

Expressing the query...

Enter your query in natural language: A red car is initially in front of another car, then drives to be behind and near the car

Choosing registered UDFs...

Registered UDFs

#	UDF Name	UDF Description
0	above	Whether o0 is above o1.
1	beneath	Whether o0 is beneath o1.
2	behind	Whether o0 is behind o1.
3	suv	Whether the type of o0 is an SUV.
4	white	Whether the color of o0 is white.
5	grey	Whether the color of o0 is grey.
6	van	Whether the type of o0 is a van.
7	to_the_left_of	Whether o0 is to the left of o1.
8	to_the_right_of	Whether o0 is to the right of o1.
9	in_front_of	Whether o0 is in front of o1.
10	sedan	Whether the type of o0 is a sedan.
11	black	Whether the color of o0 is black.
12	red	Whether the color of o0 is red.
13	blue	Whether the color of o0 is blue.
14	pickup_truck	Whether the type of o0 is a pickup truck.

Indices of UDFs to include (comma-separated, object UDFs are always included): 2, 9

Parsing the query...

Figure 6.4: VOCAL-UDF query setup interface. The user enters an NL query and selects the registered UDFs that the system may use when parsing the query.

Proposed 2 new UDF(s): {'color_red(o0)': 'Whether the color of o0 is red.', 'near(o0, o1)': 'Whether o0 is near o1.'}

Generating new UDFs...

Generated 51 candidate(s) for near(o0, o1): UDFCandidate(id: 0_distance_threshold-50, function_implementation: def py_near(o0_rnames, height, width, **kwargs):
center_o0_x = (o0_x1 + o0_x2) / 2
center_o0_y = (o0_y1 + o0_y2) / 2
center_o1_x = (o1_x1 + o1_x2) / 2
center_o1_y = (o1_y1 + o1_y2) / 2
distance = ((center_o0_x - center_o1_x) ** 2 + (center_o0_y - center_o1_y) ** 2) ** 0.5
threshold = kwargs.get('distance_threshold', 50)
return distance < threshold, test_score: -1, score: 1, loss_t: 0)

UDFCandidate(id: 0_distance_threshold-548.8135039273247, function_implementation: def py_near(o0_ename, o0_x1, o0_y1, o1_x1, o1_y1, **kwargs):
center_o0_x = (o0_x1 + o0_x2) / 2
center_o0_y = (o0_y1 + o0_y2) / 2
center_o1_x = (o1_x1 + o1_x2) / 2
center_o1_y = (o1_y1 + o1_y2) / 2
distance = ((center_o0_x - center_o1_x) ** 2 + (center_o0_y - center_o1_y) ** 2) ** 0.5
threshold = kwargs.get('distance_threshold', 50)
return distance < threshold, test_score: -1, score: 1, loss_t: 0)

Figure 6.5: VOCAL-UDF UDF proposal and generation interface. When the query refers to predicates not covered by the available UDFs, the system proposes missing UDFs and generates candidate implementations, including program-based and distilled-model UDFs.

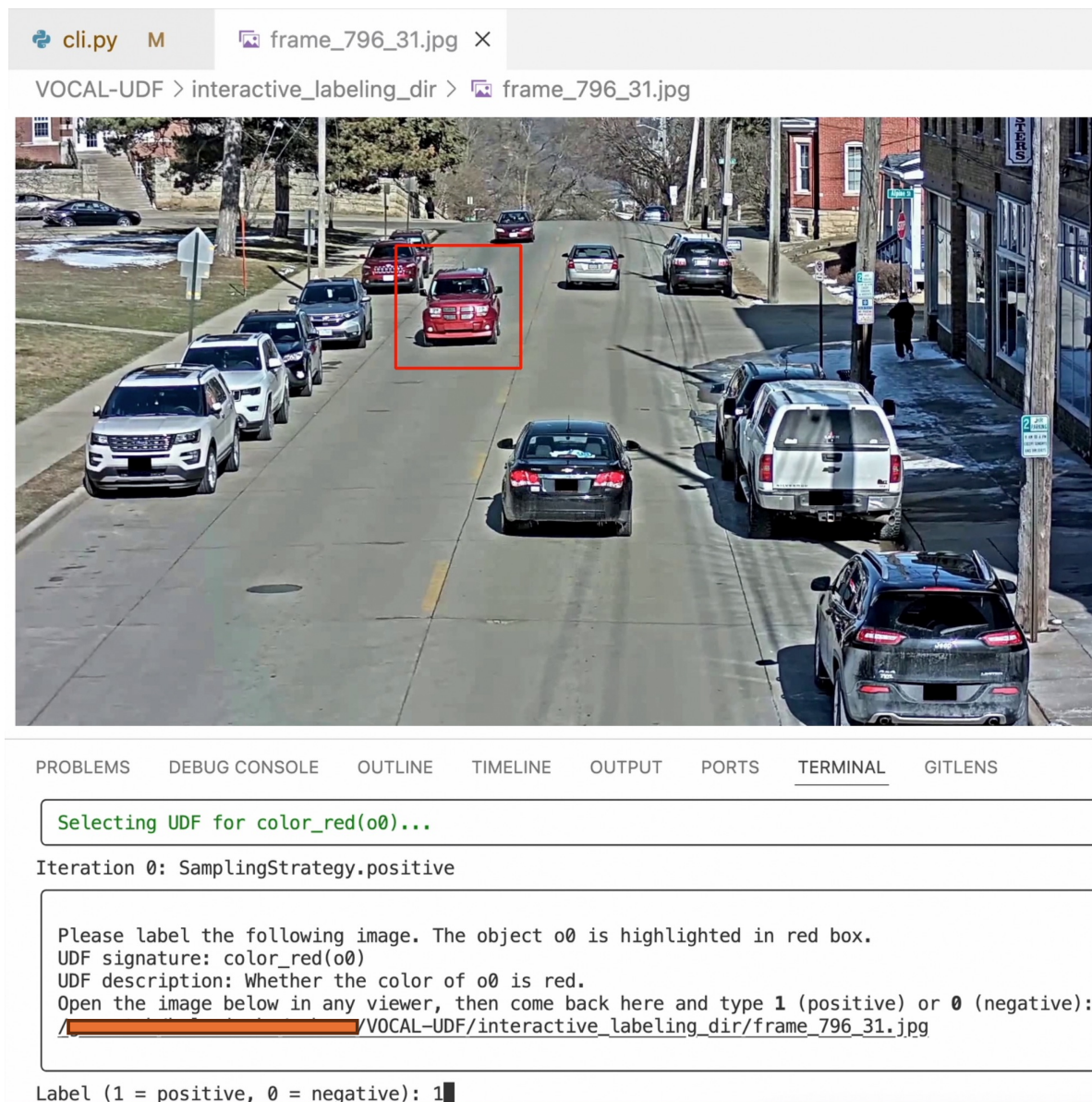


Figure 6.6: VOCAL-UDF labeling interface for UDF selection. The user labels targeted visual examples so the system can choose the candidate implementation whose behavior best matches the intended predicate.

the LLM to choose which type of UDF to generate for each missing predicate (Section 4.5.4) and, when generating a program-based UDF, producing only one implementation. However, real-world video tasks are often ambiguous, so keeping the user in the loop during UDF

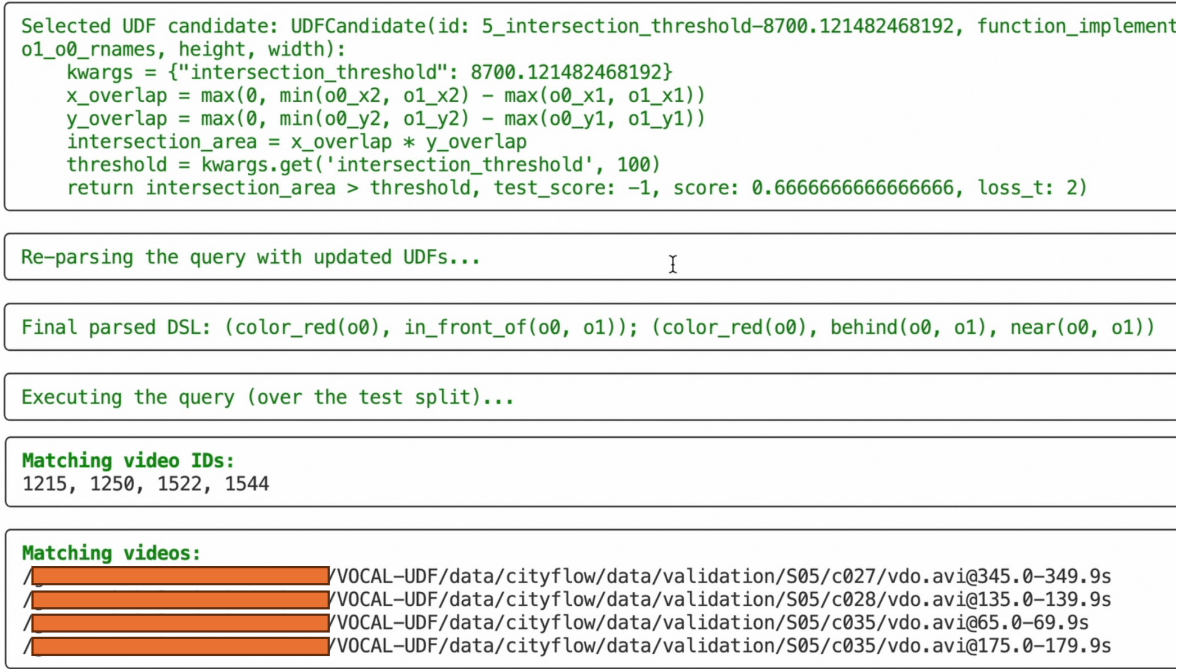


Figure 6.7: VOCAL-UDF result inspection interface. After selecting UDF implementations, the system adds them to the catalog, reparses and executes the original query, and returns matching videos.

selection helps the system choose implementations that reflect the user’s intent rather than only the LLM’s default interpretation.

6.3 VOCAL-SIFT: high-level search with predicate filters

VOCAL-SIFT supports high-level video search, where users can name an event directly in NL without specifying all the visual logic needed to recognize it. In the intended end-to-end workflow, the user provides only a high-level NL query; VOCAL-SIFT then decomposes the query, constructs filters, applies them to prune the corpus, and invokes the VLM on the retained videos.

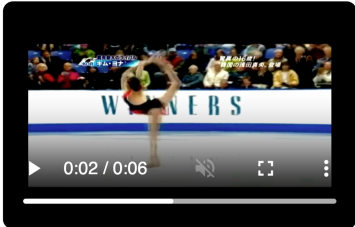
The interface shown in Figures 6.8 to 6.10 was designed specifically for the VOCAL-SIFT user study. The study compared VOCAL-SIFT’s automatic semantic decomposition with a user clarification baseline that asked follow-up questions before decomposition. Importantly,

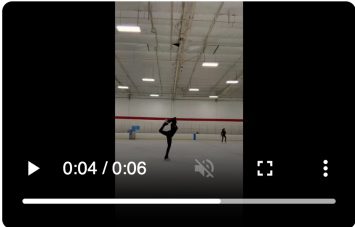
What to Find

Description: Biellmann Spin in Figure Skating

Your Task: Find **one additional video** that matches the description above. You can use the example videos below to understand what to look for, and you may use external resources (such as Google or ChatGPT) to help formulate your query.

Example Videos (3)





Step 1: Describe What You're Looking For

Type a description to help the system find one additional video that matches. The system will search for the videos based on your description. You can use the description above, or write it in your own words.

Your Search Description:

Biellmann Spin in Figure Skating

Step 2: Help Us Understand Your Search

We need a bit more detail about some parts of your search. Please tell us what you mean for each item below.

What specific visual actions or body positions would show that a figure skater is performing a Biellmann Spin?

the skater is spinning on one foot, with leg above the head.

Figure 6.8: VOCAL-SIFT query entry and optional clarification interface. The user reviews positive example videos, describes the desired high-level event in NL, and, in the user clarification baseline, answers follow-up questions about the event’s visual logic.

VOCAL-SIFT itself does not ask clarifying questions; the clarification step in Figure 6.8 appears only for the baseline comparison in the user study (Section 5.5.7). In the study workflow, participants were shown an action name and three positive example videos, then asked to find one additional matching video. They began by entering an NL query (Figure 6.8). In the user clarification baseline, the system asked follow-up questions about the event’s visual logic when needed; in the automatic decomposition condition, VOCAL-SIFT

Step 3: Finding Your Videos

Your Refined Search

spinning on one foot, with leg above the head in figure skating

Analyzing Your Search

We're checking which parts of your search need special filters to find the right videos...

Objects

✓ figure_skater
Creating a filter for this

Attributes

✓ spinning_on_one_foot_figure_skater
Creating a filter for this

✓ leg_above_head_figure_skater
Creating a filter for this

Creating Search Filters

We're creating filters to help find the videos you're looking for. This step can take a few minutes, and the page may briefly pause while filters are training—this is expected behavior. Once filters are ready, you can turn individual filters on or off below to update which videos appear in the results.

figure_skater (objects)
Ready (quality: 1.00)

spinning_on_one_foot_figure_skater (attributes)
Creating filter... 50%

leg_above_head_figure_skater (attributes)
Creating filter... 30%

VLM labeling (round 2/3)

Training MLP (round 1/3)

Figure 6.9: VOCAL-SIFT predicate and filter construction interface. The system presents the predicates derived from the user’s query, indicates which predicates are instantiated as filters, and shows filter construction progress and validation quality during filter training.

proceeded without asking these questions.

After decomposition, the interface showed the predicates that VOCAL-SIFT derived from the query and indicated which predicates were being instantiated as filters (Figure 6.9).

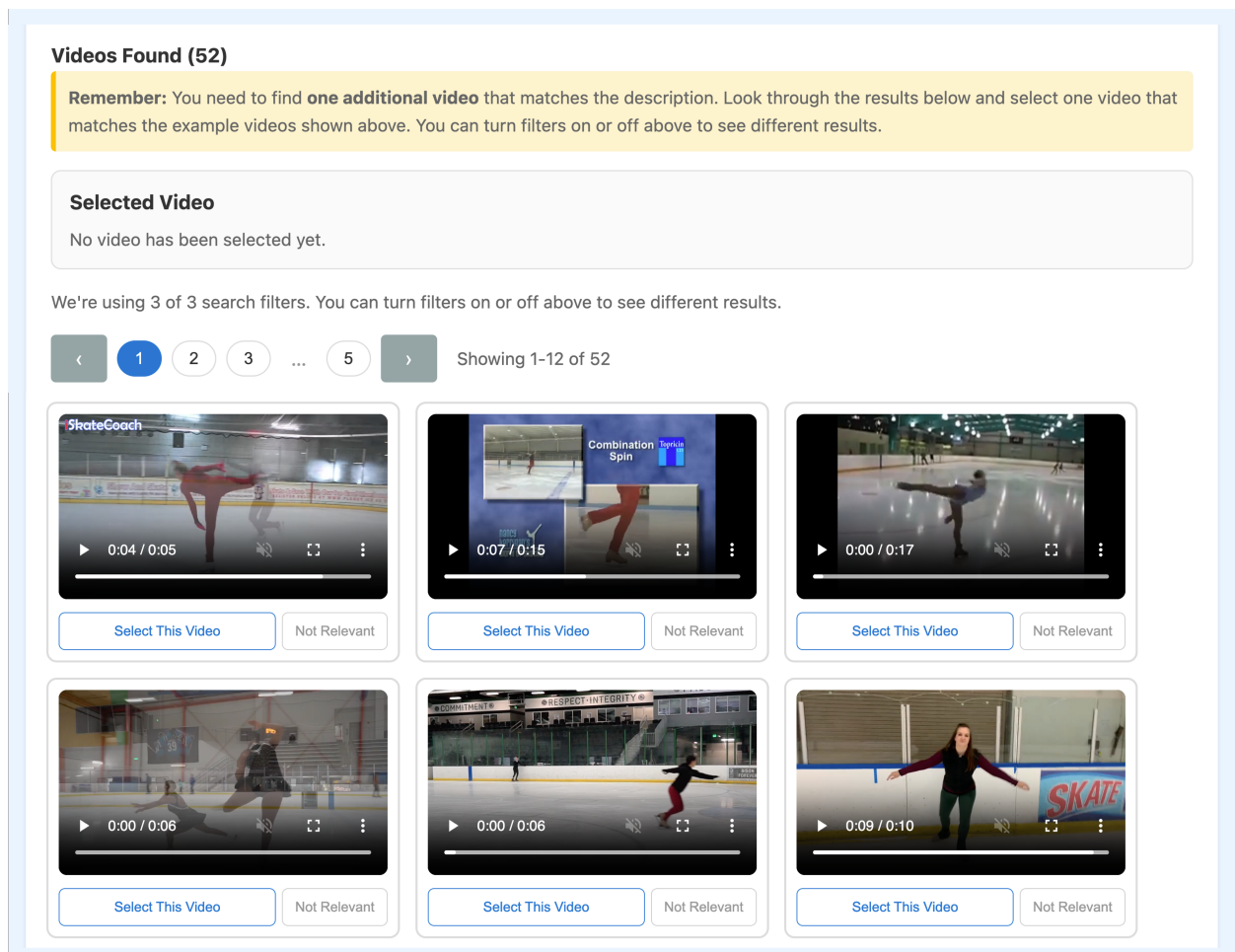


Figure 6.10: VOCAL-SIFT result inspection interface. The user reviews videos that pass the active filters, selects a matching video, marks non-relevant results, and toggles filters to inspect how predicate filtering changes the candidate set.

Showing these filters gave participants a lightweight explanation of how the search was being narrowed. The result interface then let participants examine retrieved videos and select a matching video. Participants could also mark non-relevant results and toggle filters to inspect how predicate filtering changed the candidate set (Figure 6.10).

6.4 Discussion

These prototypes suggest several implications for future end-to-end video query interfaces.

First, interfaces should support multiple channels for expressing intent. Different forms of user input communicate different parts of the user’s goal. NL queries are useful for describing the overall event, but they often leave implicit details such as temporal order, distance thresholds, or which visual evidence matters. Event-level labeled examples, such as positive and negative video segments, ground the full query in concrete instances and can guide query synthesis, but finding suitable initial examples from a large video corpus can be difficult for users. Asking users for predicate-level labels can be another way to capture their intent. Predicate-level labels operate at a lower level of the query hierarchy, specifying the intended meaning of a particular visual component rather than the full event. The three systems in this dissertation show that different stages of video querying benefit from different forms of input. Additionally, our POLY-VOCAL work [68], not described in this dissertation, explores a complementary direction by combining NL descriptions, previously synthesized queries, and user-labeled examples to bootstrap query synthesis more effectively than relying on examples alone.

Second, generated artifacts should be inspectable and persistent. Our three systems generate different artifacts during video querying. EQUI-VOCAL generates synthesized queries, VOCAL-UDF produces candidate UDF implementations and selected UDFs, and VOCAL-SIFT generates necessary predicates and filters. These artifacts should be inspectable because they make explicit how the system translated user intent into executable components. Making them visible can help users identify where an error may have occurred, decide whether an intermediate result matches their intent, and correct or audit the system’s behavior. These artifacts should also be persistent because they can remain useful beyond the query that produced them. VOCAL-UDF adds selected UDFs to the catalog so they can be reused by future queries, and VOCAL-SIFT’s learned filters could similarly be retained for recurring predicates or related high-level events. POLY-VOCAL further illustrates this direction by retrieving previously synthesized queries to initialize later query synthesis [68]. A future interface should treat these artifacts as a managed semantic layer, where users can inspect their provenance and quality, decide when they should be reused, and disable,

update, or version them as tasks and user intent evolve.

Finally, interaction should be targeted. User feedback is most useful when it resolves uncertainty that the system cannot handle reliably on its own. EQUI-VOCAL and VOCAL-UDF use active learning to request labels that are expected to be most informative for query synthesis or UDF selection, rather than asking users to label examples indiscriminately. The VOCAL-SIFT user study shows that AI assistance can reduce additional user effort without measurably degrading retrieval quality, task success, or subjective workload Section 5.5.7. This leaves an important open direction for future interfaces to decide when to involve the user, when to proceed automatically, and what form of interaction best supports the task without adding unnecessary latency.

Chapter 7

CONCLUSION AND FUTURE DIRECTIONS

Video is a content-rich and easy-to-capture data medium that has enabled emerging applications and systems across various fields. Advances in computer vision and artificial intelligence have greatly expanded the capabilities of video analytics, with pretrained vision models now available for tasks such as image classification, action recognition, and object detection. Yet pretrained models that can identify arbitrary, domain-specific events typically do not exist. More recently, large language models (LLMs) and vision language models (VLMs) have shown strong potential for visual question answering and other challenging visual tasks, but they still struggle with complex visual reasoning, and deploying them at scale remains computationally expensive. These trends make video database management an important research area.

However, current video database management systems (VDBMSs) face notable limitations in supporting complex video queries that ask how objects interact with each other spatially and temporally. First, writing declarative queries for complex events often demands significant database expertise. Second, existing VDBMSs assume all modules needed to answer a complex query already exist. In practice, this assumption frequently fails, because predefined vision models are unavailable for many domain-specific concepts. Lastly, many user queries are high-level: they name events directly, but do not specify the visual logic needed to recognize them. Large AI models make these tasks more approachable, but using them as black-box query evaluators over every video is too expensive for large-scale data management.

This dissertation presented three systems to address these challenges and advance the capabilities of compositional and high-level video querying. First, in Chapter 3 we introduced

EQUI-VOCAL, which addresses the challenge of query formulation for compositional events. It lets users specify an event through a small number of positive and negative video examples, and then synthesizes a declarative query on the user’s behalf. EQUI-VOCAL introduces an expressive data model and query language based on spatio-temporal scene graphs. It proposes a new query synthesis approach that explores the query search space efficiently using results of executing intermediate queries and beam search. Further, EQUI-VOCAL generates efficient SQL queries to reduce computational costs during query synthesis and leverages active learning to strategically select additional examples for labeling to reduce user effort. Our evaluation of EQUI-VOCAL shows that it outperforms two baseline systems—in terms of F1 score, synthesis time, and robustness to data noise—and can flexibly synthesize queries that the baselines do not support. We also conduct a user study to evaluate how real users perform with EQUI-VOCAL. EQUI-VOCAL and its demonstration appeared in PVLDB Volume 16 [235, 234], and the code is available at <https://github.com/uwdb/equi-vocal> and <https://github.com/uwdb/equi-vocal-demo>.

In Chapter 4 we introduced VOCAL-UDF, which addresses the challenge of missing modules. VOCAL-UDF is a self-enhancing system that enables users to issue compositional queries even when UDFs necessary to answer the queries are unavailable. Given an NL query, VOCAL-UDF uses LLMs and VLMs to identify missing modules, implement them as UDFs, and select the most effective ones for query execution. To handle a diverse range of semantic concepts, it supports both program-based and distilled-model UDF generation. To handle the inherent ambiguity in user intent and LLM errors, it generates multiple candidate UDFs with different semantic interpretations, implementations, and properties, and selects the implementation that best matches the user’s intent through active learning. Across synthetic and real-world datasets, VOCAL-UDF maintains strong query performance when predicates are missing and substantially outperforms two baseline systems and two direct LLM approaches. We also conduct a thorough analysis of VOCAL-UDF to understand its efficiency in terms of execution time and monetary cost. VOCAL-UDF appeared in SIGMOD 2025 [236], and the code is available at <https://github.com/uwdb/vocal-udf>.

In Chapter 5 we introduced VOCAL-SIFT, which addresses the semantic gap between high-level queries and compositional queries. VOCAL-SIFT efficiently processes high-level queries by automatically identifying necessary predicates that should hold whenever the event occurs, constructing lightweight filters for these necessary predicates, and invoking expensive VLM reasoning only on candidate videos that pass the filters. VOCAL-SIFT designs a multi-step semantic decomposition process to derive reusable necessary predicates by independently generating candidates, aligning them against an existing library, and reconciling LLM outputs into a robust predicate set. Next, we develop an LLM-based filter selection method that avoids training redundant predicate filters by estimating semantic dependency, which reduces filter construction cost while preserving filtering quality. Finally, we instantiate necessary predicates as object-grounded distilled predicate filters. Experiments with our prototype implementation and three datasets show that VOCAL-SIFT maintains high recall while substantially reducing expensive VLM invocations, and a user study confirms that our automatic decomposition reduces task completion time compared to human clarification, without a significant degradation in filtering quality or user experience.

The systems in this dissertation take steps toward making compositional and high-level video queries easier to specify and more efficient to execute, but they also point to several directions for future work.

First, future systems should develop richer interaction models for translating evolving user intent into executable video queries. As AI models make it possible to ask more abstract questions over video, users may not begin with a fully specified query; their intent often becomes clearer only after they see examples, intermediate predicates, generated UDFs, or retrieved results. Future interfaces should therefore bring together the three implications discussed in Chapter 6. They should let users express intent through NL, examples, edits, and labels; expose generated queries, UDFs, predicates, and filters as persistent artifacts that can be inspected and reused; and ask for feedback only at points where user judgment can meaningfully improve the system’s interpretation. Interaction should therefore be treated as part of query processing itself, helping the system refine the semantics of a video query over

time.

Second, future VDBMSs should combine structured queries, vector search, and model inference in a unified query processing framework. The systems in this dissertation use declarative query structure to make video search more precise, reusable, and efficient, while relying on LLMs, VLMs, and distilled models to handle semantic concepts that are difficult to specify or implement manually. This suggests a neurosymbolic direction for video data management in which symbolic query plans, learned predicates, embedding-based vector search, and direct model reasoning are optimized together. Such systems could use structured queries when the semantics are explicit and executable, vector search when approximate semantic retrieval is sufficient, and model inference when open-ended reasoning or disambiguation is required.

Finally, future work should explore agentic systems for video query optimization. The optimization decisions in this dissertation are still largely encoded within each individual system. EQUI-VOCAL uses explicit query execution optimizations to make synthesis efficient, VOCAL-UDF organizes query parsing, UDF proposal, candidate generation, and UDF selection into a staged workflow, and VOCAL-SIFT runs semantic decomposition and filter selection before invoking the VLM. These systems all assume that the main optimization choices have been specified before query execution. A more general VDBMS may need to adapt these choices to the query, dataset, available modules, budget, and accumulated workload. An agentic optimizer could use tools to inspect the dataset, estimate predicate selectivity, compare candidate decompositions or UDF implementations, decide when vector search or VLM inference is worth the cost, and revise a plan when intermediate results suggest that the current plan is unreliable. The main research challenge is to give such agents the flexibility to improve query execution while preserving the properties expected from a database system, including bounded cost, reproducibility, provenance, and predictable end-to-end quality.

In summary, this dissertation addressed important questions at the heart of making it easier and more efficient to ask and execute queries over video data. Many research problems

remain, and new ones will continue to arise as video data, and multimodal data more broadly, continues to grow in volume and diversity.

BIBLIOGRAPHY

- [1] Amazon bedrock pricing. <https://aws.amazon.com/bedrock/pricing>, 2024.
- [2] Amazon ec2 spot instances pricing. <https://aws.amazon.com/ec2/spot/pricing>, 2024.
- [3] Analysis: Gpt-4o vs gpt-4 turbo. <https://www.vellum.ai/blog/analysis-gpt-4o-vs-gpt-4-turbo>, 2024.
- [4] Llama 3 70b vs gpt-4: Comparison analysis. <https://www.vellum.ai/blog/llama-3-70b-vs-gpt-4-comparison-analysis>, 2024.
- [5] Nvidia dali. <https://developer.nvidia.com/dali>, 2024.
- [6] Openai api pricing. <https://openai.com/api/pricing>, 2024.
- [7] Sherif M. Abdou and Michael S. Scordilis. Beam search pruning in speech recognition using a posterior probability-based confidence measure. *Speech Commun.*, 42(3-4):409–428, 2004.
- [8] Neil Agarwal and Ravi Netravali. Boggart: Towards {General-Purpose} acceleration of retrospective video analytics. In *NSDI*, pages 933–951, 2023.
- [9] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: A visual language model for few-shot learning. In *NeurIPS*, 2022.
- [10] Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. Asking clarifying questions in open-domain information-seeking conversations. In *SIGIR*, pages 475–484, 2019.

- [11] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10):58–67, 2017.
- [12] Michael R. Anderson, Michael J. Cafarella, Germán Ros, and Thomas F. Wenisch. Physical representation-based predicate optimization for a visual analytics database. In *ICDE*, pages 1466–1477, 2019.
- [13] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, pages 39–48, 2016.
- [14] Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. Star-gate: Teaching language models to ask clarifying questions, 2024.
- [15] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models, 2021.
- [16] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond, 2023.
- [17] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond, 2023.
- [18] J. Bang, Pramod Chunduri, and Joy Arulraj. Eko: Adaptive sampling of compressed video data. *ArXiv*, abs/2104.01671, 2021.
- [19] Jaeho Bang, Gaurav Tarlok Kakkar, Pramod Chunduri, Subrata Mitra, and Joy Arulraj. SEIDEN: revisiting query processing in video database systems. *PVLDB*, 16(9):2289–2301, 2023.
- [20] Emmanouil N Barmpounakis, Eleni I Vlahogianni, and John C Golias. Unmanned aerial aircraft systems for transportation engineering: Current practice and future challenges. *IJTST*, 5(3):111–122, 2016.
- [21] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael J. Cafarella, Tim Kraska, and Sam Madden. MIRIS: fast object track queries in video. In *SIGMOD*, pages 1907–1921, 2020.

- [22] Favyen Bastani and Samuel Madden. OTIF: efficient tracker pre-processing over large video datasets. In *SIGMOD*, pages 2091–2104, 2022.
- [23] Favyen Bastani, Oscar R. Moll, and Samuel Madden. Vaas: Video analytics at scale. *PVLDB*, 13(12):2877–2880, 2020.
- [24] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.
- [25] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kudipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022.
- [26] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

- [27] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4, 2023.
- [28] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, pages 535–541, 2006.
- [29] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *ICLR*, 2024.
- [30] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *CVPR*, pages 4724–4733, 2017.
- [31] Shih-Fu Chang, William Chen, Horace J. Meng, Hari Sundaram, and Di Zhong. Videoq: An automated content based video search system using visual cues. In *MM*, pages 313–324, 1997.
- [32] Daren Chao, Yueting Chen, Nick Koudas, and Xiaohui Yu. Optimizing video queries with declarative clues. *PVLDB*, 17(11):3256–3268, 2024.
- [33] Daren Chao, Nick Koudas, and Ioannis Xarchakos. SVQ++: querying for object interactions in video streams. In *SIGMOD*, pages 2769–2772, 2020.
- [34] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *ICLR*, 2023.
- [35] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Dorsa Sadigh, Leonidas J. Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *CVPR*, pages 14455–14465, 2024.
- [36] Guobin Chen, Wongun Choi, Xiang Yu, Tony X. Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *NeurIPS*, pages 742–751, 2017.
- [37] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *TMLR*, 2024, 2024.

- [38] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.
- [39] Tianshui Chen, Weihao Yu, Riquan Chen, and Liang Lin. Knowledge-embedded routing network for scene graph generation. In *CVPR*, pages 6163–6171, 2019.
- [40] Tianyu Chen, Shuai Lu, Shan Lu, Yeyun Gong, Chenyuan Yang, Xuheng Li, Md Rakib Hossain Misu, Hao Yu, Nan Duan, Peng Cheng, Fan Yang, Shuvendu K. Lahiri, Tao Xie, and Lidong Zhou. Automated proof generation for rust code via self-evolution. In *ICLR*, 2025.
- [41] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *ICLR*, 2024.
- [42] Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. In *ICLR*, 2019.
- [43] Yueting Chen, Nick Koudas, Xiaohui Yu, and Ziqiang Yu. Spatial and temporal constrained ranked retrieval over videos. *PVLDB*, 15(11):3226–3239, 2022.
- [44] Yueting Chen, Xiaohui Yu, Nick Koudas, and Ziqiang Yu. Evaluating temporal queries over video feeds. In *SIGMOD*, pages 287–299, 2021.
- [45] Zhenfang Chen, Rui Sun, Wenjun Liu, Yining Hong, and Chuang Gan. GENOME: generative neuro-symbolic visual reasoning by growing and reusing modules. In *ICLR*, 2024.
- [46] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. Seed: Domain-specific data curation with large language models, 2024.

- [47] Jaemin Cho, Yushi Hu, Jason M. Baldridge, Roopal Garg, Peter Anderson, Ranjay Krishna, Mohit Bansal, Jordi Pont-Tuset, and Su Wang. Davidsonian scene graph: Improving reliability in fine-grained evaluation for text-to-image generation. In *ICLR*, 2024.
- [48] Rohan Choudhury, Koichiro Niinuma, Kris M. Kitani, and László A. Jeni. Zero-shot video question answering with procedural programs. *CoRR*, abs/2312.00937, 2023.
- [49] Pramod Chunduri, Jaeho Bang, Yao Lu, and Joy Arulraj. Zeus: Efficiently localizing actions in videos using reinforcement learning. In *SIGMOD*, pages 545–558, 2022.
- [50] Isabel F. Cruz and Wendy T. Lucas. Delaunay^{mm}: A visual framework for multimedia presentation. In *VL*, pages 216–223, 1997.
- [51] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Ze Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):5:1–5:60, 2008.
- [52] Maureen Daum, Enhao Zhang, Dong He, Magdalena Balazinska, Brandon Haynes, Ranjay Krishna, Apryle Craig, and Aaron Wirsing. VOCAL: video organization and interactive compositional analytics. In *CIDR*, 2022.
- [53] Maureen Daum, Enhao Zhang, Dong He, Stephen Mussmann, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. Vocalexplore: Pay-as-you-go video data exploration and model building. *PVLDB*, 16(13):4188–4201, 2023.
- [54] Justin Dellinger, Carolyn Shores, Apryle Craig, Shannon Kachel, Michael Heithaus, William Ripple, and Aaron Wirsing. Predators reduce niche overlap between sympatric prey. *Oikos*, 12 2021.
- [55] Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-sql: A survey of what we have and what we expect. In *COLING*, pages 2166–2187, 2022.
- [56] Zhongjun Ding, Yin Lin, Tianjing Zeng, Rong Zhu, Bolin Ding, and Jingren Zhou. Ambisql: Interactive ambiguity detection and resolution for text-to-sql, 2026.
- [57] Mehmet Emin Dönderler, Ediz Saykol, Umut Arslan, Özgür Ulusoy, and Ugur Güdükbay. Bilvideo: Design and implementation of a video database management system. *Multim. Tools Appl.*, 27(1):79–104, 2005.

- [58] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot text-to-sql with chatgpt, 2023.
- [59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.
- [60] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, pages 11733–11763, 2024.
- [61] Iman Elghandour and Ashraf Aboulnga. Restore: Reusing results of mapreduce jobs. *PVLDB*, 5(6):586–597, 2012.
- [62] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *ICCV*, pages 6804–6815, 2021.
- [63] Anna Fariha and Alexandra Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *PVLDB*, 12(11):1262–1275, 2019.
- [64] Qi Feng, Vitaly Ablavsky, and Stan Sclaroff. Cityflow-nl: Tracking and retrieval of vehicles at city scale by natural language descriptions, 2021.
- [65] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. NL2SQL is a solved problem... not! In *CIDR*, 2024.
- [66] Reinhard Friedl, Helmut Höppler, Karl Ecard, Wilfried Scholz, Andreas Hannekum, Wolfgang Öchsner, and Sylvia Stracke. Multimedia-driven teaching significantly improves students’ performance when compared with a print medium. *The Annals of Thoracic Surgery*, 81(5):1760–1766, 2006.
- [67] Daniel Y. Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. Recall: Specifying video events using compositions of spatiotemporal labels. *arXiv preprint arXiv:1910.02993*, 2019.

- [68] Manasi Ganti, Enhao Zhang, and Magdalena Balazinska. Bootstrapping compositional video query synthesis with natural language and previous queries from users. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–7, 2025.
- [69] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *PVLDB*, 17(5):1132–1145, 2024.
- [70] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *UIST*, pages 489–500, 2015.
- [71] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, pages 3354–3361, 2012.
- [72] Google Gemini Team. Gemini: A family of highly capable multimodal models, 2025.
- [73] Shilpa George, Junjue Wang, Mihir Bala, Thomas Eiszler, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards drone-sourced live video analytics for the construction industry. In *HotMobile*, pages 3–8, 2019.
- [74] Chunhui Gu, Chen Sun, David A. Ross, Carl Vondrick, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. In *CVPR*, pages 6047–6056, 2018.
- [75] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *CVPR*, pages 14953–14962, 2023.
- [76] Patrick Hammer, Tony Lofthouse, Enzo Fenoglio, Hugo Latapie, and Pei Wang. A reasoning based model for anomaly detection in the smart city domain. In *IntelliSys*, volume 1251 of *AISC*, pages 144–159, 2020.
- [77] Brandon Haynes, Amrita Mazumdar, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. Visual road: A video data management benchmark. In *SIGMOD*, pages 972–987, 2019.
- [78] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, 2017.

- [79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.
- [80] Wenjia He, Michael R. Anderson, Maxwell Strome, and Michael J. Cafarella. A method for optimizing opaque filter queries. In *SIGMOD*, pages 1257–1272, 2020.
- [81] Wenjia He and Michael J. Cafarella. Controlled intentional degradation in analytical video systems. In *SIGMOD*, pages 2105–2119, 2022.
- [82] Wenjia He, Ibrahim Sabek, Yuze Lou, and Michael J. Cafarella. Optimizing video selection LIMIT queries with commonsense knowledge. *PVLDB*, 17(7):1751–1764, 2024.
- [83] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.
- [84] Cheng-Yu Hsieh, Sibe Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander J. Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models, 2023.
- [85] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of ACL*, pages 8003–8017, 2023.
- [86] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, pages 269–286, 2018.
- [87] Bo Hu, Peizhen Guo, and Wenjun Hu. Video-zilla: An indexing layer for large-scale video analytics. In *SIGMOD*, pages 1905–1919, 2022.
- [88] Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models, 2023.
- [89] Jiani Huang, Calvin Smith, Osbert Bastani, Rishabh Singh, Aws Albarghouthi, and Mayur Naik. Generating programmatic referring expressions via program synthesis. In *ICML*, volume 119, pages 4495–4506, 2020.

- [90] Chan Hur, Jeong-Hun Hong, Dong-hun Lee, Dabin Kang, Semin Myeong, Sang-hyo Park, and Hyeyoung Park. Narrating the video: Boosting text-video retrieval via comprehensive utilization of frame-level captions. In *CVPR*, pages 24077–24086, 2025.
- [91] Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. Action genome: Actions as compositions of spatio-temporal scene graphs. In *CVPR*, pages 10236–10247, 2020.
- [92] Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. Action genome: Actions as compositions of spatio-temporal scene graphs. In *CVPR*, pages 10233–10244, 2020.
- [93] Ruyi Ji, Jingjing Liang, Yingfei Xiong, Lu Zhang, and Zhenjiang Hu. Question selection for interactive program synthesis. *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020.
- [94] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*, pages 253–266, 2018.
- [95] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. Inferring and executing programs for visual reasoning. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 3008–3017. IEEE Computer Society, 2017.
- [96] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *CVPR*, pages 3668–3678, 2015.
- [97] Gaurav Tarlok Kakkar, Jiashen Cao, Aubhro Sengupta, Joy Arulraj, and Hyesoon Kim. Aero: Adaptive query processing of ml queries. *Proc. ACM Manag. Data*, 3(3), 2025.
- [98] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.
- [99] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *PVLDB*, 13(4):533–546, 2019.

- [100] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing deep cnn-based queries over video streams at scale. *PVLDB*, 10(11):1586–1597, 2017.
- [101] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. Approximate selection with guarantees using proxies. *PVLDB*, 13(11):1990–2003, 2020.
- [102] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. Accelerating approximate aggregation queries with expensive predicates. *PVLDB*, 14(11):2341–2354, 2021.
- [103] Daniel Kang, John Guibas, Peter D. Bailis, Tatsunori Hashimoto, and Matei Zaharia. TASTI: semantic indexes for machine learning-based queries over unstructured data. In *SIGMOD*, pages 1934–1947, 2022.
- [104] Mohammad Reza Karimi, Nezihe Merve Gürel, Bojan Karlas, Johannes Rausch, Ce Zhang, and Andreas Krause. Online active model selection for pre-trained classifiers. In *AISTATS*, volume 130, pages 307–315, 2021.
- [105] George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *VLDB J.*, 32(4):905–936, 2023.
- [106] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.
- [107] Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sang-goo Lee, and Taek Kim. Aligning language models to explicitly handle ambiguity. In *EMNLP*, pages 1989–2007, 2024.
- [108] Chanwut Kittivorawong, Yongming Ge, Yousef Helal, and Alvin Cheung. Spatia-lyze: A geospatial video analytics system with spatial-aware optimizations. *PVLDB*, 17(9):2136–2148, 2024.
- [109] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwawata. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.
- [110] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwawata. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.

- [111] Nick Koudas, Raymond Li, and Ioannis Xarchakos. Video monitoring queries. In *ICDE*, pages 1285–1296, 2020.
- [112] Ranjay Krishna, Vincent S. Chen, Paroma Varma, Michael S. Bernstein, Christopher Ré, and Li Fei-Fei. Scene graph prediction with limited labels. In *ICCV*, pages 2580–2590, 2019.
- [113] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vis.*, 123(1):32–73, 2017.
- [114] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Clam: Selective clarification for ambiguous questions with generative language models, 2023.
- [115] Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Beam search algorithms for multilabel learning. *Mach. Learn.*, 92(1):65–89, 2013.
- [116] Tony C. T. Kuo and Arbee L. P. Chen. Content-based query processing for video databases. *TMM*, 2(1):1–13, 2000.
- [117] Christopher A Kurby and Jeffrey M Zacks. Segmentation in the perception and memory of events. *TiCS*, 12(2):72–79, 2008.
- [118] Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization, 2023.
- [119] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, pages 951–958, 2009.
- [120] Huy Dinh-Anh Le, Quang Qui-Vinh Nguyen, Duc Trung Luu, Truc Thi-Thanh Chau, Nhat Minh Chung, and Synh Viet-Uyen Ha. Tracked-vehicle retrieval by natural language descriptions with multi-contextual adaptive knowledge. In *CVPR Workshops*, pages 5511–5519, 2023.
- [121] Dongryeol Lee, Segwang Kim, Minwoo Lee, Hwanhee Lee, Joonsuk Park, Sang-Woo Lee, and Kyomin Jung. Asking clarification questions to handle ambiguity in open-domain QA. In *EMNLP (Findings)*, pages 11526–11544, 2023.

- [122] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *ICML*, 1994.
- [123] Fei Li and Hosagrahar V Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.
- [124] Hao Li, Chee-Yong Chan, and David Maier. Query from examples: An iterative, data-driven approach to query construction. *PVLDB*, 8(13):2158–2169, 2015.
- [125] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *NeurIPS*, 2023.
- [126] Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, pages 19730–19742, 2023.
- [127] Dahua Lin, Sanja Fidler, Chen Kong, and Raquel Urtasun. Visual semantic search: Retrieving videos via complex textual queries. In *CVPR*, pages 2657–2664, 2014.
- [128] Chunwei Liu, Matthew Russo, Michael J. Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael J. Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. Palimpsest: Optimizing ai-powered analytics with declarative query processing. In *CIDR*, 2025.
- [129] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.
- [130] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: marrying DINO with grounded pre-training for open-set object detection. In *ECCV*, volume 15105, pages 38–55, 2024.
- [131] Xiaochen Liu, Pradipta Ghosh, Oytun Ulutan, B. S. Manjunath, Kevin S. Chan, and Ramesh Govindan. Caesar: cross-camera complex activity recognition. In *Sensys*, pages 232–244, 2019.

- [132] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *ECCV*, pages 852–869, 2016.
- [133] Kuan Lu, Zhihui Yang, Sai Wu, Ruichen Xia, Dongxiang Zhang, and Gang Chen. Adda: Towards efficient in-database feature generation via llm-based agents. *Proc. ACM Manag. Data*, 3(3):125:1–125:27, 2025.
- [134] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, 2023.
- [135] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. Neurologic a*esque decoding: Constrained text generation with lookahead heuristics. In *NAACL*, pages 780–799, 2022.
- [136] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *SIGMOD*, pages 1493–1508, 2018.
- [137] Zixian Ma, Jerry Hong, Mustafa Omer Gul, Mona Gandhi, Irena Gao, and Ranjay Krishna. CREPE: can vision-language foundation models reason compositionally? In *CVPR*, pages 10910–10921, 2023.
- [138] Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m & m’s: A benchmark to evaluate tool-use for multi-step multi-modal tasks. In *ECCV*, volume 15068, pages 18–34, 2024.
- [139] Muhammad Maaz, Hanoona Abdul Rasheed, Salman Khan, and Fahad Khan. Video-chatgpt: Towards detailed video understanding via large vision and language models. In *ACL*, pages 12585–12602, 2024.
- [140] Rachael Mady, Peter Mason, Matt Strimas-Mackey, Miyoko Chu, Tina Phillips, David Bonter, Charles Eldermire, and Benjamin Walters. Bird cams lab biological data. <https://ecommons.cornell.edu/handle/1813/110264>, 2021.
- [141] Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. Egoschema: A diagnostic benchmark for very long-form video language understanding. In *NeurIPS*, 2023.

- [142] Clara Meister, Ryan Cotterell, and Tim Vieira. Best-first beam search. *Trans. Assoc. Comput. Linguistics*, 8:795–809, 2020.
- [143] Stephen Mell, Favyen Bastani, Steve Zdancewic, and Osbert Bastani. Synthesizing trajectory queries from examples. In *CAV*, pages 459–484, 2023.
- [144] Stephen Mell, Steve Zdancewic, and Osbert Bastani. Optimal program synthesis via abstract interpretation. *Proc. ACM Program. Lang.*, 8(POPL):457–481, 2024.
- [145] Sachit Menon and Carl Vondrick. Visual classification via description from large language models. In *ICLR*, 2023.
- [146] Hoshi Mistry, Prasan Roy, S. Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD*, pages 307–318, 2001.
- [147] Chancharik Mitra, Brandon Huang, Trevor Darrell, and Roei Herzig. Compositional chain-of-thought prompting for large multimodal models. In *CVPR*, pages 14420–14431, 2024.
- [148] Chancharik Mitra, Brandon Huang, Trevor Darrell, and Roei Herzig. Compositional chain-of-thought prompting for large multimodal models. In *CVPR*, pages 14420–14431, 2024.
- [149] Oscar Moll, Favyen Bastani, Sam Madden, Michael Stonebraker, Vijay N. Gadeppally, and Tim Kraska. Exsample: Efficient searches on video repositories through adaptive sampling. *ICDE*, pages 3065–3077, 2020.
- [150] Ravi Teja Mullapudi, Fait Poms, William R. Mark, Deva Ramanan, and Kayvon Fatahalian. Learning rare category classifiers on a tight labeling budget. In *ICCV*, pages 8403–8412, 2021.
- [151] Bolin Ni, Houwen Peng, Minghao Chen, Songyang Zhang, Gaofeng Meng, Jianlong Fu, Shiming Xiang, and Haibin Ling. Expanding language-image pretrained models for general video recognition. In *ECCV*, volume 13664, pages 1–18, 2022.
- [152] Augustus Odena, Kensen Shi, David Bieber, Rishabh Singh, Charles Sutton, and Hanjun Dai. BUSTLE: bottom-up program synthesis through learning-guided exploration. In *ICLR*, 2021.

- [153] Eitetsu Oomoto and Katsumi Tanaka. OVID: design and implementation of a video-object database system. *TKDE*, 5(4):629–643, 1993.
- [154] OpenAI. Gpt-4 technical report, 2024.
- [155] OpenAI. Vision - openai api documentation. <https://developers.openai.com/api/docs/guides/images-vision>, 2026.
- [156] Stefan Palan and Christian Schitter. Prolific.ac—a subject pool for online experiments. *Journal of Behavioral and Experimental Finance*, 17:22–27, 2018.
- [157] Yuren Pang, Katharina Reinecke, and René Just. Apéritif: Scaffolding preregistrations to automatically generate analysis code and methods descriptions. In *CHI*, pages 207:1–207:15, 2022.
- [158] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Trans. Graph.*, 37(4), 2018.
- [159] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. Towards a theory of natural language interfaces to databases. In *IUI*, pages 149–157, 2003.
- [160] Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*, 2023.
- [161] Sarah M. Pratt, Ian Covert, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. In *ICCV*, pages 15645–15655, 2023.
- [162] Fotis Psallidas, Bolin Ding, Kaushik Chakrabarti, and Surajit Chaudhuri. S4: top-k spreadsheet-style search for query discovery. In *SIGMOD*, pages 2001–2016, 2015.
- [163] Wei Qiao, Tushar Dogra, Otilia Stretcu, Yu-Han Lyu, Tiantian Fang, Dongjin Kwon, Chun-Ta Lu, Enming Luo, Yuan Wang, Chih-Chun Chia, Ariel Fuxman, Fangzhou Wang, Ranjay Krishna, and Mehmet Tek. Scaling up LLM reviews for google ads content moderation. In *WSDM*, pages 1174–1175, 2024.
- [164] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*, 2024.

- [165] Mark Raasveldt and Hannes Muehleisen. DuckDB.
- [166] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021.
- [167] Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In Iryna Gurevych and Yusuke Miyao, editors, *ACL (1)*, pages 2737–2746, 2018.
- [168] Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. Learning programs from noisy data. In *POPL*, pages 761–774, 2016.
- [169] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016.
- [170] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NeurIPS*, pages 91–99, 2015.
- [171] Shuhuai Ren, Linli Yao, Shicheng Li, Xu Sun, and Lu Hou. Timechat: A time-sensitive multimodal large language model for long video understanding. In *CVPR*, pages 14313–14323, 2024.
- [172] Francisco Romero, Johann Hauswald, Aditi Partap, Daniel Kang, Matei Zaharia, and Christos Kozyrakis. Optimizing video analytics with declarative model relationships. *PVLDB*, 16(3):447–460, 2022.
- [173] Francisco Romero, Caleb Winston, Johann Hauswald, Matei Zaharia, and Christos Kozyrakis. Zelda: Video analytics using vision-language models, 2023.
- [174] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines. In *SoCC*, pages 1–17, 2021.
- [175] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *ICML*, 2001.

- [176] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhole. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, pages 249–260, 2000.
- [177] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.
- [178] Olga Russakovsky, Li-Jia Li, and Li Fei-Fei. Best of both worlds: Human-machine collaboration for object annotation. In *CVPR*, pages 2121–2131, 2015.
- [179] Mohammadreza Salehi, Jae Sung Park, Aditya Kusupati, Ranjay Krishna, Yejin Choi, Hanna Hajishirzi, and Ali Farhadi. Actionatlas: A videoqa benchmark for domain-specialized action recognition. In *NeurIPS*, 2024.
- [180] Ville Satopaa, Jeannie R. Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *ICDCS Workshops*, pages 166–171, 2011.
- [181] Dario Satriani, Enzo Veltri, Donatello Santoro, Sara Rosato, Simone Varriale, and Paolo Papotti. Logical and physical optimizations for SQL query execution over large language models. *Proc. ACM Manag. Data*, 3(3):181:1–181:28, 2025.
- [182] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.
- [183] Kyra Schlining, Susan von Thun, Linda Kuhn, Brian Schlining, Lonny Lundsten, Nancy Jacobsen Stout, Lori Chaney, and Judith Connor. Debris in the deep: Using a 22-year video annotation database to survey marine litter in monterey canyon, central california, usa. *Deep Sea Research Part I: Oceanographic Research Papers*, 79:96–105, 2013.
- [184] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018.
- [185] Andrew W Senior, L Brown, Arun Hampapur, C-F Shu, Yun Zhai, Rogério Schmidt Feris, Y-L Tian, Sergio Borger, and C Carlson. Video analytics for retail. In *AVSS*, pages 423–428, 2007.

- [186] H. Sebastian Seung, Manfred Opper, and Haim Sompolsky. Query by committee. In *Annual Conference Computational Learning Theory*, 1992.
- [187] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. Docetl: Agentic query rewriting and evaluation for complex document processing. *PVLDB*, 18(9):3035–3048, 2025.
- [188] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving AI tasks with chatgpt and its friends in hugging face. In *NeurIPS*, 2023.
- [189] Amit P. Sheth, Clemens Bertram, and Kshitij Shah. Videoanywhere: A system for searching and managing distributed video assets. *SIGMOD Rec.*, 28(1):104–109, 1999.
- [190] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.
- [191] Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *ICCV*, pages 2986–2997, 2023.
- [192] Manuel Stein, Halldor Janetzko, Andreas Lamprecht, Thorsten Breitzkreutz, Philipp Zimmermann, Bastian Goldlücke, Tobias Schreck, Gennady L. Andrienko, Michael Grossniklaus, and Daniel A. Keim. Bring it to the pitch: Combining video and movement data to enhance team sport analysis. *TVCG*, 24(1):13–22, 2018.
- [193] Otilia Stretcu, Edward Vendrow, Kenji Hata, Krishnamurthy Viswanathan, Vittorio Ferrari, Sasan Tavakkol, Wenlei Zhou, Aditya Avinash, Enming Luo, Neil Gordon Alldrin, MohammadHossein Bateni, Gabriel Berger, Andrew Bunner, Chun-Ta Lu, Javier A Rey, Giulia DeSalvo, Ranjay Krishna, and Ariel Fuxman. Agile modeling: From concept to classifier in minutes. In *ICCV*, pages 22266–22277, 2023.
- [194] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *ICCV*, pages 11854–11864, 2023.
- [195] Keita Takenouchi, Takashi Ishio, Joji Okada, and Yuji Sakata. PATSQL: efficient synthesis of SQL queries from example tables with quick inference of projected columns. *PVLDB*, 14(11):1937–1949, 2021.

- [196] Imad Eddine Toubal, Aditya Avinash, Neil Gordon Alldrin, Jan Dlabal, Wenlei Zhou, Enming Luo, Otilia Stretcu, Hao Xiong, Chun-Ta Lu, Howard Zhou, Ranjay Krishna, Ariel Fuxman, and Tom Duerig. Modeling collaborator: Enabling subjective vision classification with minimal human effort via LLM tool-use. In *CVPR*, pages 17553–17563, 2024.
- [197] Michael Tschannen, Alexey A. Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier J. Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. 2025.
- [198] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. TRANSIT: specifying protocols with concolic snippets. In *PLDI*, pages 287–296, 2013.
- [199] Matthias Urban and Carsten Binnig. CAESURA: language models as multi-modal query planners. In *CIDR*, 2024.
- [200] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.
- [201] Carl Vondrick, Donald J. Patterson, and Deva Ramanan. Efficiently scaling up crowd-sourced video annotation - A set of best practices for high quality, economical video labeling. *IJCV*, 101(1):184–204, 2013.
- [202] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *Trans. Mach. Learn. Res.*, 2024, 2024.
- [203] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. In *NeurIPS*, 2023.
- [204] Chenglong Wang, Alvin Cheung, and Rastislav Bodík. Synthesizing highly expressive sql queries from input-output examples. *PLDI*, 2017.
- [205] Chenglong Wang, Yu Feng, Rastislav Bodík, Isil Dillig, Alvin Cheung, and Amy J. Ko. Falx: Synthesis-powered visualization authoring. In *CHI*, pages 106:1–106:15, 2021.

- [206] Jiayu Wang, Yifei Ming, Zhenmei Shi, Vibhav Vineet, Xin Wang, Sharon Li, and Neel Joshi. Is A picture worth A thousand words? delving into spatial reasoning for vision language models. In *NeurIPS*, 2024.
- [207] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *SEC*, pages 159–173, 2018.
- [208] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024.
- [209] Sida I. Wang, Samuel Ginn, Percy Liang, and Christopher D. Manning. Naturalizing a programming language via interactive learning. In *ACL*, pages 929–938, 2017.
- [210] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. SkyEyes: adaptive video streaming from UAVs. In *HotWireless*, pages 2–6, 2016.
- [211] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.
- [212] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *WACV*, pages 748–756, 2018.
- [213] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.
- [214] Renzhi Wu, Pramod Chunduri, Ali Payani, Xu Chu, Joy Arulraj, and Kexin Rong. Sketchql: Video moment querying with a visual query interface. *Proc. ACM Manag. Data*, 2(4):204:1–204:27, 2024.
- [215] Yanchao Xu, Dongxiang Zhang, Shuhao Zhang, Sai Wu, Zexu Feng, and Gang Chen. Predictive and near-optimal sampling for view materialization in video databases. *Proc. ACM Manag. Data*, 2(1), 2024.

- [216] Zhuangdi Xu, Gaurav Tarlok Kakkar, Joy Arulraj, and Umakishore Ramachandran. EVA: A symbolic approach to accelerating exploratory video analytics with materialized views. In *SIGMOD*, pages 602–616, 2022.
- [217] Piyush Yadav and Edward Curry. Vidcep: Complex event processing framework to detect spatiotemporal patterns in video streams. In *Big Data*, pages 2513–2522, 2019.
- [218] Tanush Yadav, Mohammadreza Salehi, Jae Sung Park, Vivek Ramanujan, Hannaneh Hajishirzi, Yejin Choi, Ali Farhadi, Rohun Tripathi, and Ranjay Krishna. Videonet: A large-scale dataset for domain-specific action recognition, 2026.
- [219] Yuxuan Yan, Shiqi Jiang, Ting Cao, Yifan Yang, Qianqian Yang, Yuanchao Shu, Yuqing Yang, and Lili Qiu. AVA: Towards agentic video analytics with vision language models. In *NSDI*, pages 1939–1957, 2026.
- [220] Junran Yang, Alex Bäuerle, Dominik Moritz, and Çagatay Demiralp. Vegaprof: Profiling vega visualizations. In *UIST*, pages 94:1–94:11, 2023.
- [221] Yue Yang, Artemis Panagopoulou, Shenghao Zhou, Daniel Jin, Chris Callison-Burch, and Mark Yatskar. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. In *CVPR*, pages 19187–19197, 2023.
- [222] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.
- [223] Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. In *NeurIPS*, 2023.
- [224] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B. Tenenbaum. CLEVRER: collision events for video representation and reasoning. In *ICLR*, 2020.
- [225] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *TACL*, 2:67–78, 2014.
- [226] Ruichi Yu, Ang Li, Vlad I. Morariu, and Larry S. Davis. Visual relationship detection with internal and external linguistic knowledge distillation. In *ICCV*, pages 1068–1076, 2017.

- [227] Shan Yu, Zhenting Zhu, Yu Chen, Hanchen Xu, Pengzhan Zhao, Yang Wang, Arthi Padmanabhan, Hugo Latapie, and Harry Xu. Vqpy: An object-oriented approach to modern video analytics. In *MLSys*, 2024.
- [228] Shoubin Yu, Jaemin Cho, Prateek Yadav, and Mohit Bansal. Self-chained image-language model for video localization and question answering. In *NeurIPS*, 2023.
- [229] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, pages 3911–3921, 2018.
- [230] Jeffrey M Zacks, Barbara Tversky, and Gowri Iyer. Perceiving, remembering, and communicating structure in events. *Journal of experimental psychology: General*, 130(1):29, 2001.
- [231] Hamed Zamani, Susan T. Dumais, Nick Craswell, Paul N. Bennett, and Gord Lueck. Generating clarifying questions for information retrieval. In *WWW*, pages 418–428, 2020.
- [232] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI*, pages 1050–1055, 1996.
- [233] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *CVPR*, pages 5831–5840, 2018.
- [234] Enhao Zhang, Maureen Daum, Dong He, Manasi Ganti, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. EQUI-VOCAL demonstration: Synthesizing video queries from user interactions. *PVLDB*, 16(12):3978–3981, 2023.
- [235] Enhao Zhang, Maureen Daum, Dong He, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. EQUI-VOCAL: synthesizing queries for compositional video events from limited user interactions. *PVLDB*, 16(11):2714–2727, 2023.
- [236] Enhao Zhang, Nicole Sullivan, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. Self-enhancing video data management system for compositional events with large language models. *Proc. ACM Manag. Data*, 3(3):215:1–215:29, 2025.
- [237] Michael J. Q. Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with lms. In *Findings of the Association for Computational*

Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025, Findings of ACL, pages 5526–5543, 2025.

- [238] Michael J. Q. Zhang, W. Bradley Knox, and Eunsol Choi. Modeling future conversation turns to teach llms to ask clarifying questions. In *ICLR*, 2025.
- [239] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017.
- [240] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *ICLR*, 2024.
- [241] Linchao Zhu and Yi Yang. Compound memory networks for few-shot video classification. In *ECCV*, pages 782–797, 2018.
- [242] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. Learning optimal tree models under beam search. In *ICML*, pages 11650–11659, 2020.

Appendix A

LLM PROMPTS

We provide a set of example prompts we use in VOCAL-UDF.

Prompt of query parsing

Legend: DSL definition, UDF definition, Registered UDF, Instruction

Each video segment is a sequence of N frames. The visual content of each frame is represented by a region graph: A region graph contains a set of objects in a frame, along with a set of relationships between those objects. Objects can optionally have attributes. In our DSL, we use a variable o to represent an object in a query. Different variables represent different objects. All predicates of a region graph are connected by commas. Then, region graphs are connected in temporal sequence with semicolons. Region graphs that appear earlier in the sequence represent temporally earlier frames in the video. We further use the notation `Duration(g, d)` to require that the region graph g exist in at least d consecutive frames. Negation operation is not supported in our DSL. Remember to always add parentheses around comma-connected predicates. Assume the video segments capture 25 frames per second.

A function can take one of the following three formats, depending on if it is a relationship predicate or an attribute predicate:

- relationship predicate: `relationshipName(o0, o1)`. For example, `jumping_in(o0, o1)` checks whether o0 is jumping in o1.
- attribute predicate: `key_value(o0)`. For example, `color_bronze(o0)` checks whether the color of o0 is bronze.

You have access to the following functions:

`left_of(o0, o1)`: Whether o0 is on the left of o1.

`front_of(o0, o1)`: Whether o0 is in front of o1.

`location_left(o0)`: Whether o0 is on the left of the frame.

`location_top(o0)`: Whether o0 is at the top of the frame.

`color_gray(o0)`: Whether the color of o0 is gray.

`color_red(o0)`: Whether the color of o0 is red.

`color_blue(o0)`: Whether the color of o0 is blue.

`color_green(o0)`: Whether the color of o0 is green.

`shape_cube(o0)`: Whether the shape of o0 is cube.

`shape_sphere(o0)`: Whether the shape of o0 is sphere.

`material_rubber(o0)`: Whether the material of o0 is rubber.

For text-to-DSL translation tasks, only use the functions you have been provided with. Reply `PARSE_YES` when the text is successfully translated into the DSL and verified by the provided function, or `PARSE_NO` if parsing the user input requires new predicates that are not listed in the current functions list. The predicates **MUST** be selected from the provided functions.

Figure A.1: Prompt of query parsing.

Prompt of UDF proposal

Legend: Instruction

For function proposal tasks, only use the functions you have been provided with. Reply `TERMINATE` when the task is done. Please propose the new functions that are necessary to parse the user query, and also include a brief description for each

proposed function that explains its purpose as described in the query. The function description should always start with the word “Whether” and not contain other comments, explanations, or reasoning.

Let’s think step by step. Based on the existing functions, determine what new functions are needed. The proposed function must follow the format. Don’t propose functions that contain changes in states. If you have those, propose a separate function for each state instead. For example, do not propose a function `merge_from_A_into_B(o0)` that checks whether an object `o0` merges from lane A to lane B because it contains two states: `o0` is in lane A and `o0` is in lane B. Instead, replace it with two separate functions: `in_lane_A(o0)` and `in_lane_B(o0)`. Propose as few functions as possible while ensuring that the user’s intent can be precisely captured.

Figure A.2: Prompt of UDF proposal uses the same prompt as in Figure A.1, but with an updated instruction.

Prompt of program-based UDF generation

Legend: Instruction, Schema info, Output format

Generate 10 Python functions with different, diverse semantic interpretations for the following Python task. Each generation should include the semantic interpretation and the Python function implementation, formatted as a dictionary. The response should strictly adhere to the formats described below:

- Task: Write a python function called ‘`py_near(img, o0_otype, o0_x1, o0_y1, o0_x2, o0_y2, o0_otypes, o1_otype, o1_x1, o1_y1, o1_x2, o1_y2, o1_otypes, o0_o1_rnames, o1_o0_rnames, height, width, **kwargs)`’ that determines whether `o0` is near `o1`.
- Each interpretation should offer a different but reasonable understanding of the task, not just superficial differences like variable names. Seek interpretations that vary in

logic and conceptual understanding of the task. Consider geometric, visual, and spatial perspectives. Include assumptions or constraints where relevant.

- Prioritize generating functions that are likely to see frequent use, starting with the most common.

- The input to the function contains the following parameters:

- `img`: `np.ndarray` of shape (H, W, C). The image is in the RGB color space, where H is the height, W is the width, and C is the number of channels.

- `o0_otype`: `str`. The class name of object `o0`.

- `o0_x1`: `int`. The x-coordinate of the top-left corner of the bounding box of object `o0`.

- `o0_y1`: `int`. The y-coordinate of the top-left corner of the bounding box of object `o0`.

- `o0_x2`: `int`. The x-coordinate of the bottom-right corner of the bounding box of object `o0`.

- `o0_y2`: `int`. The y-coordinate of the bottom-right corner of the bounding box of object `o0`.

- `o0_otypes`: `List[str]`. The list of attribute names of object `o0`.

- `o1_otype`: `str`. The class name of object `o1`.

- `o1_x1`: `int`. The x-coordinate of the top-left corner of the bounding box of object `o1`.

- `o1_y1`: `int`. The y-coordinate of the top-left corner of the bounding box of object `o1`.

- `o1_x2`: `int`. The x-coordinate of the bottom-right corner of the bounding box of object `o1`.

- `o1_y2`: `int`. The y-coordinate of the bottom-right corner of the bounding box of object `o1`.

- `o1_otypes`: `List[str]`. The list of attribute names of object `o1`.

- `o0_o1_rnames`: `List[str]`. The list of relationship names between object `o0` and object `o1`, where object `o0` is the subject and object `o1` is the target.

- o1_o0_rnames: List[str]. The list of relationship names between object o1 and object o0, where object o1 is the subject and object o0 is the target.
 - height: int. The height of the frame.
 - width: int. The width of the frame.
 - **kwargs: Optional numeric parameters that can be adjusted as needed.
 - Available object names: ['object']
 - Available attribute names: ['location_left', 'location_top', 'color_gray', 'color_red', 'color_blue', 'color_green', 'shape_cube', 'shape_sphere', 'material_rubber']
 - Available relationship names: ['left_of', 'front_of']
 - The origin $(x, y) = (0, 0)$ is located at the top left corner. The x axis is oriented from left to right; the y axis is oriented from top to bottom.
 - The function should return a boolean value, indicating whether the relationship between the two objects is true or false.
 - Include '**kwargs' in the function's arguments only if necessary. Only arguments of numeric data types are allowed in '**kwargs'. String, boolean, or object data types are not allowed in '**kwargs'.
 - You can use any python packages you want (except for sklearn). IT IS LIFE THREATENING THAT you do not use sklearn library. You do not need to install but only import them before using. You can not use supervised-learning method as there is no training data. Though, you can use frozen models if you want.
 - The function should only contain the implementation itself, with no other comments, inline comments, syntax highlighter, explanations, reasoning, or dialogue.
 - Use the following output format:
- ```
```json
{
```

```

"answer": [
{
"semantic_interpretation": "interpretation",
"function_implementation": "def py_near(img, o0_otype, o0_x1, o0_y1, o0_x2,
o0_y2, o0_otypes, o1_otype, o1_x1, o1_y1, o1_x2, o1_y2, o1_otypes, o0_o1_rnames,
o1_o0_rnames, height, width, **kwargs):\n    # Your code here",
"kwargs": {
"arg_name1": {"min": minimum_value, "max": maximum_value, "default":
default_value},
// Add more arguments as needed.
},
},
// Add more functions as needed.
]}
'''

```

Figure A.3: Prompt of program-based UDF generation.

Prompt of object-aware sampling

Legend: Instruction, Object classes, Output format

Given a list of object classes: ['person', 'bag', 'bed', 'blanket', 'book', 'box', 'broom', 'chair', 'closet/cabinet', 'clothes', 'cup/glass/bottle', 'dish', 'door', 'doorknob', 'doorway', 'floor', 'food', 'groceries', 'laptop', 'light', 'medicine', 'mirror', 'paper/notebook', 'phone/camera', 'picture', 'pillow', 'refrigerator', 'sandwich', 'shelf', 'shoe', 'sofa/couch', 'table', 'television', 'towel', 'vacuum', 'window'], and a function

"eating(o0, o1)" that determines "Whether o0 is eating o1", assume that objects are chosen from the object classes listed above. Your task is to identify and list all object classes that can possibly be involved in this concept. It's LIFE THREATENING not to remove object classes that can possibly be involved in this concept.

Please format your answer in the JSON format shown below:

```
```json
{"answer": [
 "object_class1",
 "object_class2",
 // Add more object classes as needed.
]}
```
```

Figure A.4: Prompt of object-aware sampling, utilized in distilled-model UDF generation.

Prompt of deciding the UDF type

Legend: Instruction, Schema info

You are tasked with creating a solution to determine "Whether o0 is behind o1". You can choose to use either a python function or a computer vision model.

1. Python function: This approach is suitable for tasks that can be determined based on any of the following:

- Existing concepts of objects. You can only leverage concepts from the following predefined list: ['object', 'left_of', 'front_of', 'location_left', 'location_top', 'color_gray', 'color_red', 'color_blue', 'color_green', 'shape_cube', 'shape_sphere', 'material_rubber']. These concepts are pre-extracted for each object in the image. Concepts not listed are not available.

- Bounding box coordinates of objects.

- Statistical analysis of pixel values in the image using computer vision libraries.

2. Computer vision model: This approach is suitable for tasks that require understanding the visual content and contextual interpretation of the image.

Please specify your choice by responding with 'programUDF' to use the Python function or 'modelUDF' to use the computer vision model. Choose the approach that you believe will achieve the highest accuracy for the task. Consider only the effectiveness of each approach without concern for computational resources, time, or other constraints. Please respond with the answer only, and do not output any other responses or any explanations.

Figure A.5: Prompt of deciding the UDF type, utilized in the 11m UDF generation strategy.