Tensor Logic

The Language of AI

Pedro Domingos

University of Washington

Fields Take Off When They Find Their Language

- Physics: Calculus
- Electrical engineering: Complex numbers
- Digital circuits: Boolean logic
- Chip design: HDLs
- Networking: Internet Protocol
- Web: HTML
- Databases: Relational algebra
- Computer science: High-level languages
- Etc.

What a Field's Language Does

- Saves time
- Makes key things obvious
- Focuses attention
- Decreases entropy
- Avoids hacking
- Unites the field
- Changes how people think

Has AI Found Its Language?

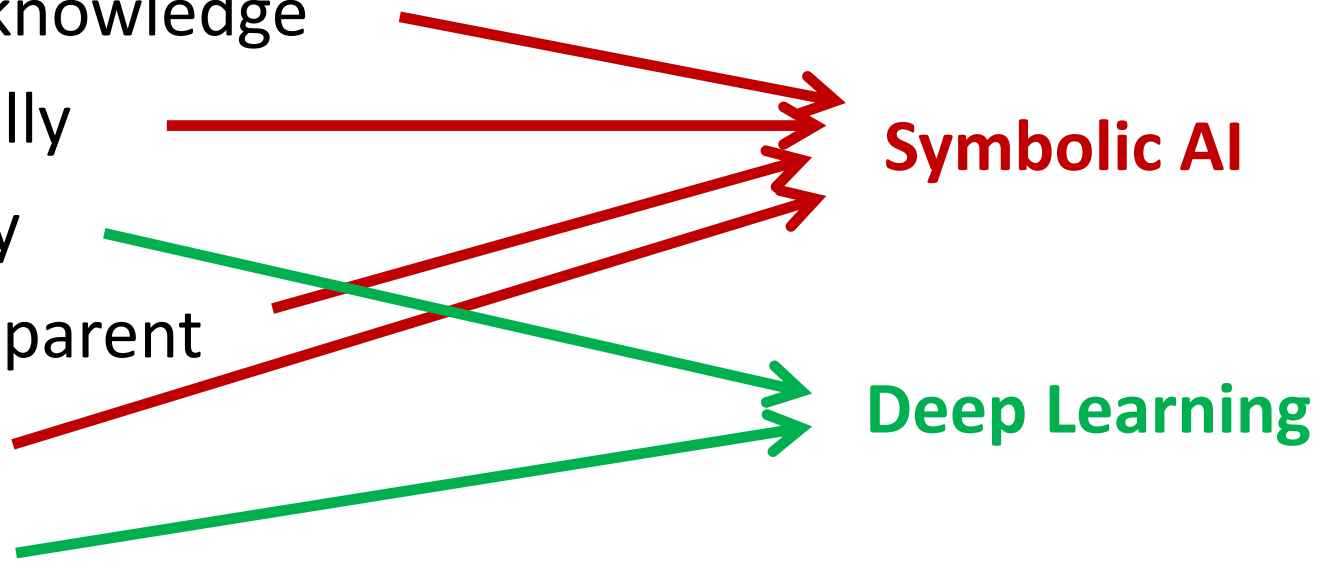
- LISP, Prolog?
- Graphical models?
- Markov logic networks?
- Python?
- NumPy, PyTorch, TensorFlow, Keras, JAX, etc.?
- Neurosymbolic AI?

What Should the Language of AI Do?

- Hide everything that’s not AI
- Easily incorporate knowledge
- Reason automatically
- Learn automatically
- Make models transparent
- Ensure reliability
- Scale effortlessly

What Should the Language of AI Do?

- Hide everything that's not AI
- Easily incorporate knowledge
- Reason automatically
- Learn automatically
- Make models transparent
- Ensure reliability
- Scale effortlessly



Tensor Logic = Tensor Algebra + Logic Programming

Deep Learning **Symbolic AI**

Logic Programming

- **Logic program** = Rules + Facts
- **Fact:** *Relation(Object$_1$, … , Object$_k$)*

  E.g.: Parent(Bob,Chris), Ancestor(Alice,Bob)
- **Rule:** *Head :- Body*

  Or: *Consequent :- Antecedent$_1$, … , Antecedent$_n$*

  E.g.: Ancestor(x,y) :- Parent(x,y)

  Ancestor(x,z) :- Ancestor(x,y),Parent(y,z)
- **Prolog:** arguments may be constants, variables or functions
- **Datalog:** no functions

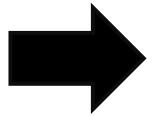
The Database View

- In database terms, a rule is a series of joins followed by a projection
- The **join** of relations *R* and *S* is the set of all tuples formed from tuples in *R* and *S* having the same values of the same arguments

E.g.:

x	y
Alice	Bob
Alice	Ed

y	z
Bob	Chris
Bob	Dan



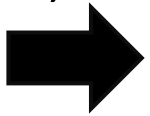
x	y	z
Alice	Bob	Chris
Alice	Bob	Dan

- The **projection** of a relation *R* onto a subset *G* of its arguments is the relation obtained by discarding all arguments of *R* not in *G*

E.g:

x	y	z
Alice	Bob	Chris
Alice	Bob	Dan

x, z



x	z
Alice	Chris
Alice	Dan

Inference in Logic Programming

- **Forward chaining:**
  Repeatedly apply all rules until no new facts can be inferred
- **Backward chaining:**
  Given query, check if it's a fact
  If not, find rule(s) with head = query & repeat with their bodies
- E.g.: Query: Ancestor(Alice,Chris)?
  Answer: True
- E.g. Query: Ancestor(Alice,x)?
  Answer: {Bob,Chris}

Inductive Logic Programming

- **Input:** Database
- **Output:** Logic program
- **Inverse deduction:** What rules would allow inferring target predicate from evidence?
- **Search:** greedy, beam, etc.
- **Objective:** accuracy, information gain, simplicity, etc.
- **Prior knowledge** is easy to incorporate
- **Declarative bias:** predefined form for rules, etc.
- **Predicate invention:** discovering hidden relations

Tensor Algebra

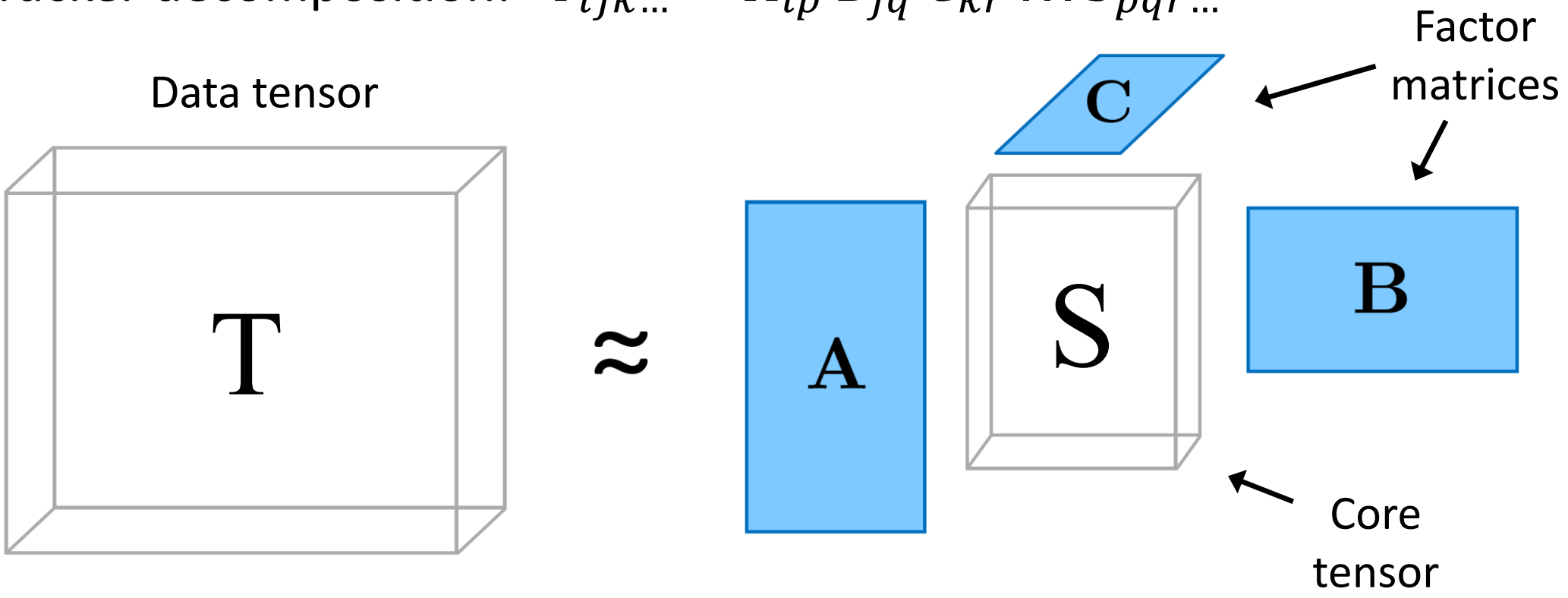
- Neural networks = Tensor algebra + Univariate nonlinearities
- A tensor is defined by its **type** (real, integer, Boolean, etc.) and **shape** (#indices and #elements along each index)
- **Tensor sum:** $C_{ijk\ldots} = A_{ijk\ldots} + B_{ijk\ldots}$
- **Tensor product:** $C_{ijk\ldots i'j'k'\ldots} = A_{ijk\ldots} B_{i'j'k'\ldots}$
- Other operations: elementwise product, tensor contraction, operations on matrices and vectors, etc.

Einstein Summation (Einsum)

- All these operations are special cases of Einstein summation
- Einstein notation: omit all summation signs and sum over all repeated indices
- E.g., matrix multiplication: $A\ B = \sum_j A_{ij}\ B_{jk} = A_{ij}\ B_{jk}$
- So neural networks = Einsum + Univariate nonlinearities
- Implemented in NumPy, PyTorch, TensorFlow, etc.

Tensor Decompositions

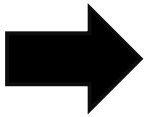
- Singular value decomposition: $M_{ij} = A_{ip}\, B_{jq}\, S_{pq}$
- Tucker decomposition: $T_{ijk\ldots} = A_{ip}\, B_{jq}\, C_{kr} \ldots S_{pqr\ldots}$



First Key Idea

- **Q:** What is the relation between tensors and relations?
- **A:** A relation is a compact representation of a sparse Boolean tensor

	Alice	Bob	Chris	Dan
Alice	0	1	0	0
Bob	0	0	1	1
Chris	0	0	0	0
Dan	0	0	0	0



Alice	Bob
Bob	Chris
Bob	Dan

Second Key Idea

- **Q:** What is the relation between rules and einsums?
- **A:** Rules are einsums over Boolean tensors, with a step function as the nonlinearity

Aunt(x,z) :- Sister(x,y), Parent(y,z) ← Prolog

$$A_{xz} = H(S_{xy}\, P_{yz}) \leftarrow \text{Einsum}$$

Aunt[x,z] = step(Sister[x,y] Parent[y,z]) ← Tensor Logic

Tensor Logic

- **Tensor projection**: $\pi_\alpha(T) = \sum_\beta T_{\alpha\beta}$
- **Tensor join**: $(U \bowtie V)_{\alpha\beta\gamma} = U_{\alpha\beta}\, V_{\beta\gamma}$
- A **tensor equation** is:
  - A series of tensor joins
  - Followed by a tensor projection onto the LHS indices
  - Optionally followed by a univariate nonlinearity, applied elementwise

$$Tensor_0 = f(Tensor_1 \ldots Tensor_n)$$

- A **tensor logic program** is a set of tensor equations
- Tensor elements are 0 by default
- Equations with same LHS are summed
- Tensor types and shapes may be declared or inferred

Syntactic Sugar

- Multiple terms in one equation: $Y = \text{step}(W[i]\,X[i] + C)$
- Index functions: $X[i,t+1] = W[i,j]\,X[j,t]$
- Normalization: $Y[i] = \text{softmax}(X[i])$
- Other tensor functions: $Y[k] = \text{concat}(X[i,j])$
- Alternate aggregations: +=, max=, avg=
- Procedural attachment
- Prolog syntax
- Etc.

Neural Networks in Tensor Logic

- Perceptron (complete program):

  $Y = \text{step}(W[i]\, X[i])$

  $W = [0.2, 1.9, -0.7, 3]$

  $X = [0, 1, 1, 0]$ (or: $X(1), X(2)$)

  $Y?$

- Multilayer perceptron:

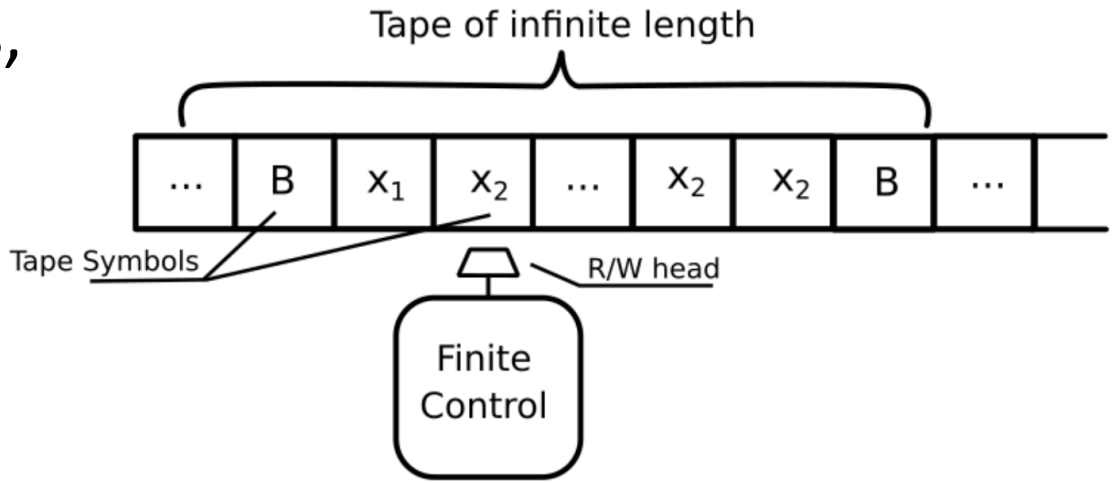
  $X[i,j] = \text{sig}(W[i,j,k]\, X[i-1,k])$

- Recurrent neural network:

  $X[i,t+1] = \text{sig}(W[i,j]\, X[j,t] + V[i,j]\, U[j,t])$

Tensor Logic Is Turing-Complete

- RNNs are Turing-complete (Siegelmann & Sontag, 1995)
- RNNs can be implemented in tensor logic
- Therefore tensor logic is Turing-complete
- But Datalog is not
- Kolmogorov-Arnold representation theorem:
  Every multivariate function is a sum of univariate ones
- Prolog puts functions inside predicates,
  tensor logic puts them outside



Inference

- Forward chaining (cf. Datalog, Rete):
  - Treat program as linear code
  - At each step compute tensor elements whose inputs are available
  - Repeat until no new elements can be computed
- Backward chaining (cf. Prolog):
  - Treat each tensor equation as a function
  - Query is top-level call
  - Recurse until query is answered
- Best choice depends on application

Learning

- Gradients via tensor equations:

$$y = ax \;\Rightarrow\; \frac{dy}{dx} = a \qquad Y_{ij} = M_{ik}X_{kj} \;\Rightarrow\; \frac{\partial Y_{ij}}{\partial M_{ik}} = X_{kj}$$

$$Y = W_i X_i \;\Rightarrow \frac{dY}{dW_i} = X_i \qquad Y_{\dots} = T_{\dots}X^1{}_{\dots}\,X^2{}_{\dots}\ldots Xn_{\dots} \;\Rightarrow\; \frac{\partial Y_{\dots}}{\partial T_{\dots}} = X^1{}_{\dots}X^2{}_{\dots}\ldots X^n{}_{\dots}$$

- The gradient of a tensor logic program is a tensor logic program:

$$\frac{\partial Loss}{\partial T} = \sum_{RHS \ni T} \frac{\partial Loss}{\partial LHS}\frac{\partial LHS}{\partial RHS} \prod_{\{X \in RHS\} \setminus \mathrm{T}} X$$

- Backpropagation through structure
- Predicate invention by Tucker decomposition
- Split tensors into constant, data and learnable

Convnets in Tensor Logic

- Convolutional layer:

Features[x,y] = relu(Filter[dx,dy,ch] Image[x+dx,y+dy,ch])

- Sum-pooling layer:

Pooled[x/S,y/S] = Features[x,y]

Graph Neural Networks in Tensor Logic

- Graph: Neig(Alice,Bob), Neig(Bob,Chris), etc.
- Initialization: Emb[n,0,d] = X[n,d] (node, layer, dimension)
- MLP: Z[n,l,d'] = relu(W_P[l,d',d] Emb[n,l,d]) , etc.
- Aggregation: Agg[n,l,d] = Neig(n,n') Z[n',l,d']
- Update: Emb[n,l+1,d] = relu(W_{Agg} Agg[n,l,d] + W_{Self} Z[n,l,d])
- Node classification: Y[n] = sig(W_{Out}[d] Emb[n,L,d])
- Edge prediction: Y[n,n'] = sig(Emb[n,L,d] Emb[n',L,d])
- Graph classification: Y = sig(W_{Out}[d] Emb[n,L,d])

Attention in Tensor Logic

$$Query[p,d_k] = W_Q[d_k,d]\,X[p,d]$$

$$Key[p,d_k] = W_K[d_k,d]\,X[p,d]$$

$$Val[p,d_v] = W_V[d_v,d]\,X[p,d]$$

$$Comp[p,p'.] = softmax(Query[p,d_k]\,Key[p',d_k] / sqrt(D_k))$$

$$Attn[p,d_v] = Comp[p,p']\,Val[p',d_v]$$

Transformers in Tensor Logic

- Input: $X(p,t)$
- Embedding: $XE[p,d] = X(p,t)\,Emb[t,d]$
- Positional encoding:

  $PE[p,d] = Even(d)\sin(p/(L\hat{}(d/D_e))) + Odd(d)\cos(p/(L\hat{}((d-1)/D_e)))$
- Residual stream: $Stream[0,p,d] = XE[p,d] + PE[p,d]$
- Attention:

  $Query[b,h,p,d_k] = W_Q[b,h,d_k,d]\,Stream[b,p,d]$, etc.

  $Comp[b,h,p,p'.] = softmax(Query[b,h,p,d_k]\,Key[b,h,p',d_k]/sqrt(D_k))$

  $Attn[b,h,p,d_v] = Comp[b,h,p,p']\,Val[b,h,p',d_v]$
- Merge and layer norm:

  $Merge[b,p,d_m] = concat(Attn[b,h,p,d_v])$

  $Stream[b,p,d.] = norm(W_S[b,d,d_m]\,Merge[b,p,d_m] + Stream[b,p,d])$
- MLP: $MLP[b,p,d'] = relu(W_P[b,p,d',d]\,Stream[b,p,d])$, etc.
- Output: $Y[p,t.] = softmax(W_O[t,d]\,Stream[B,p,d])$

Symbolic AI in Tensor Logic

Just write it in Prolog.

Embedded Databases

- Embedding objects: $\mathrm{Emb}[\mathrm{obj},\mathrm{dim}]$
- Embedding relations: $\mathrm{EmbRel}[i,j] = \mathrm{Rel}(x,y)\,\mathrm{Emb}[i,x]\,\mathrm{Emb}[j,y]$
- This is a Tucker decomposition of the relation
- EmbRel is the core tensor of Rel
- Can be constructed in O(#tuples) time
- Relation symbols can also be embedded
- Reified database: $\mathrm{DB}(r,x,y)$
- Embedded database:

$$\mathrm{EmbDB}[h,i,j] = \mathrm{DB}(r,x,y)\,\mathrm{Emb}[h,r]\,\mathrm{Emb}[i,x]\,\mathrm{Emb}[j,y]$$

Embedded Knowledge Bases and Reasoning

- To embed a rule, replace antecedents & consequent by their embeddings:

  $\text{EmbCons}[\ldots] = \text{EmbAnt}_1[\ldots]\,\text{EmbAnt}_2[\ldots] \ldots \text{EmbAnt}_n[\ldots]$
- All reasoning can be done in embedding space:
  1. Embed query and evidence
  2. Reason with embedded rules
  3. Extract answer
- Works because einsum factors are commutative and associative:

  $(\text{Rel}(x,y)\,\text{Emb}[i,x]\,\text{Emb}[j,y])\,\text{Emb}[i,x']\,\text{Emb}[j,y'] \approx \text{Rel}(x',y')$, etc.
- Combines symbolic and analogical reasoning
- Similarity of two objects = Dot product of their embeddings
- Transparent and reliable

Embeddings are random unit vectors

Kernel Machines in Tensor Logic

- Kernel machine: $Y[Q] = f(A[i]\,Y[i]\,K[Q,i] + B)$
- Polynomial kernel: $K[i,i'] = (X[i,j]\,X[i',j])^n$
- Gaussian kernel: $K[i,i'] = \exp(-(X[i,j] - X[i',j])^2 / Var)$
- Structured prediction: $Y[Q,n]$

Graphical Models in Tensor Logic

Graphical Models	Tensor Logic
Potential	Tensor
Marginalization	Projection
Pointwise product	Join
Join tree	Tree-like program
Pr(Query | Evidence)	Prog(Q,E) / Prog(E)

- To encode a **Bayes net**, add an equation for each variable V:

  $Pr[var] = CondPr[var,par_1,...,par_n]\, Pr[par_1]...Pr[par_n]$

- **Sum-product theorem (Friesen & Domingos, 2016):**
  No tensor appears in more than one RHS
  ⇒ Tensor logic program computes correct probabilities in linear time
- **Otherwise:** forward chaining = belief propagation (or sample paths, etc.)

Beyond AI

- Science: tensor logic minimizes distance from equations to code
- Scientific computing is tensor operations wrapped in logic
- To make the logic learnable, use tensor logic
- To make anything learnable, write it in tensor logic

Scaling Up

- Option 1: Separation of concerns
  - Dense subtensors: GPUs
  - Sparse subtensors: database query engines, etc.
- Option 2: All on GPUs via Tucker decomposition
  - Exponential efficiency gains
  - Bounded error probability
  - Dovetails with embedding and learning

Driving Adoption

- AI is no longer a niche → Ride the wave
- Backward compatibility with Python
- Cure the big pains (e.g., hallucinations)
- Killer apps (e.g., reasoning models, code, math)
- IDEs for coding, data wrangling, modeling, evaluation, etc.
- Open source community
- Vendor competition
- Education

Next Steps

- CUDA implementation
- Applications
- Libraries
- Extensions
- New research directions

Summary

- One language for all of AI
- Tensor logic program = Set of tensor equations
- Tensor equation = Numeric Datalog rule = Einsum
- Reasoning and learning out of the box
- Transparent and reliable
- Don’t leave home without it

tensor-logic.org