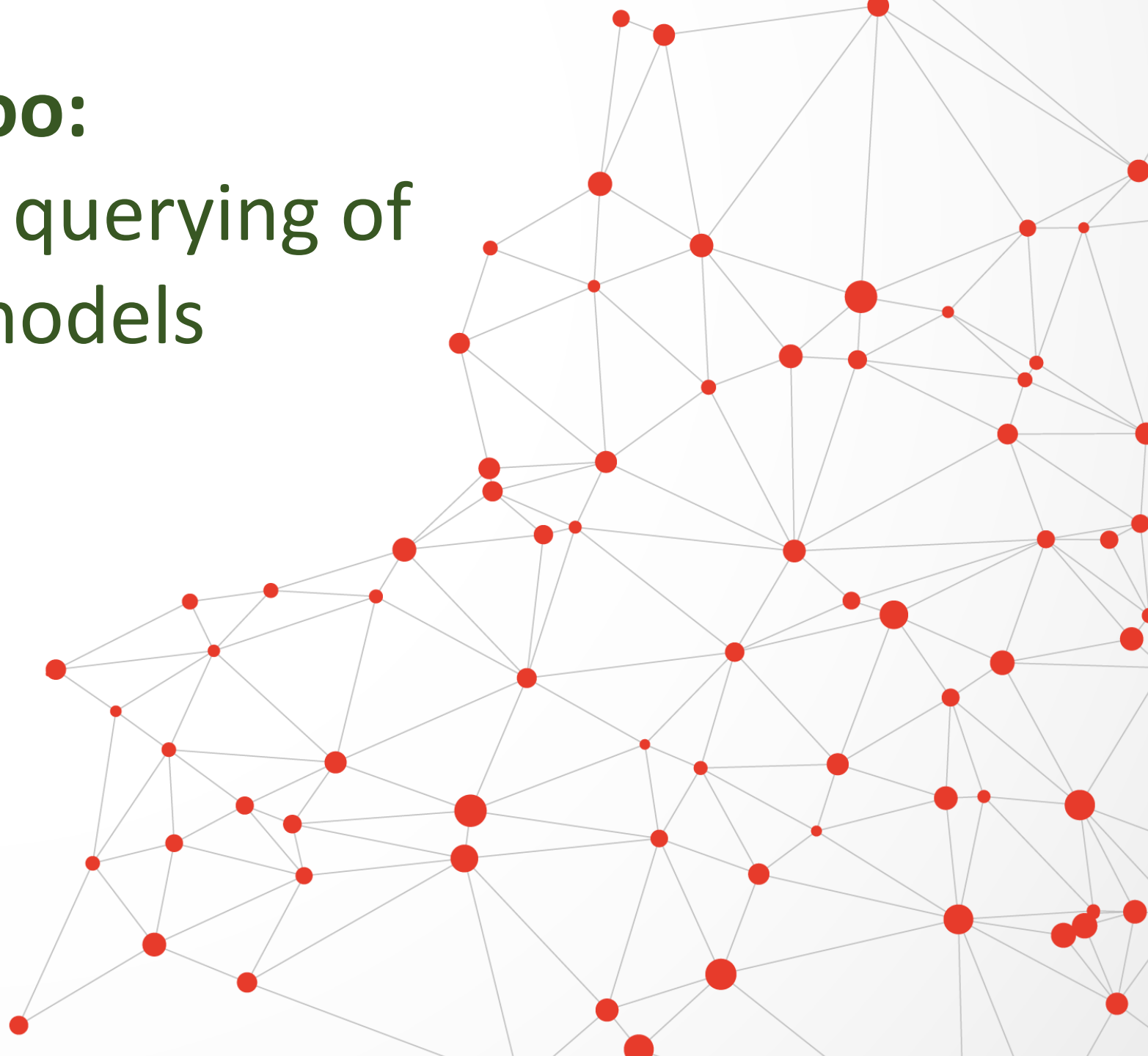
Models Are Data Too:
Towards expressive querying of machine-learning models

Jan Van den Bussche



DSI
DATA SCIENCE INSTITUTE
UHASSELT
WWW.UHASSELT.BE/DSI

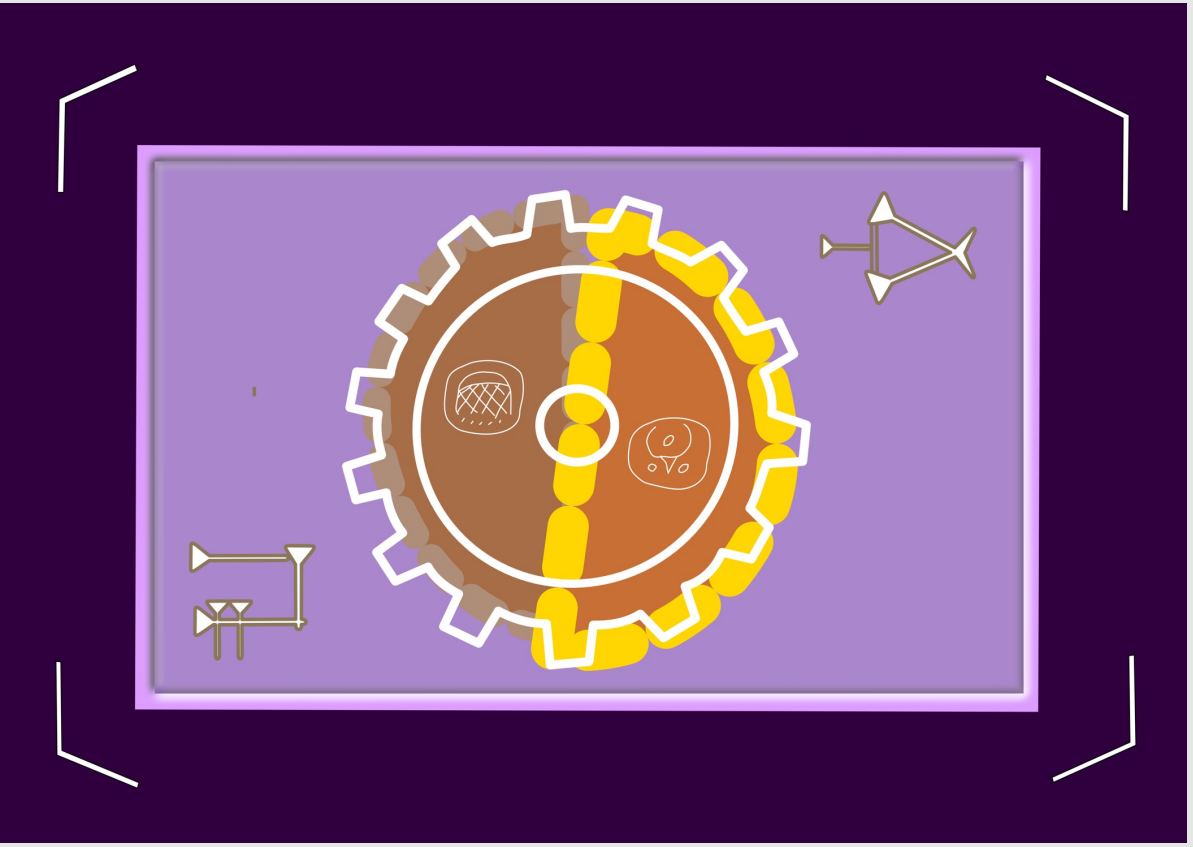
A deluge of models

- Machine Learning (ML) is ubiquitous in science, engineering, technology, society
- Worldwide
  - millions of models
- Enterprise, data science research
  - large numbers of models trained, tested, deployed, archived

Models are data too!

- We should manage them like other valuable data
- Allowing **querying** is a core purpose of a data system
- We should be able to query our models
  - What does that even mean?

Outline



I. A step back; some history
II. Querying models: a taxonomy
III. SQL can verify neural networks
IV. Conclusions

What's in a model? Many meanings

- Statistical model
- 3D model (graphics)
- Database schema [Ph. Bernstein: Model management]
- Solution to a problem; a program; code
- ML model (representation of a function)
- Database instance (possibly infinite)
- Data model (schema & query languages)

Can we develop a logical data model for models?

Code as data

- Lowell database research self-assessment [2005]

"We would like to see **code** become a first-class citizen of the DBMS"

Jim Gray, Turing Award 1998

Michael Stonebraker, Turing Award 2014

Quel as a datatype [1984]

```
append to EMP (
 name = "Fred",
 salary-history = "range of s is SALARY
                   retrieve (s.all)
                   where s.name = "Fred"",
 hobbies = "range of m is MUSIC
            retrieve (m.all) where m.name = "Fred"
            range of r is RACING
            retrieve (r.all) where r.name = "Fred"",
 dept = "range of d is DEPT
         retrieve (d.dname) where d.dname = "toy"",
 age = 25,
 bonus = 10)
```

```
range of e is EMP
retrieve (e.hobbies.instrument)
        where e.name = "Fred"
        and e.hobbies.level = "novice"
```

QUEL AS A DATA TYPE

by

Michael Stonebraker, Erika Anderson, Eric Hanson
and Brad Rubenstein

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CA.

ABSTRACT

This paper explores the use of commands in a query language as an abstract data type (ADT) in data base management systems. Basically, an ADT facility allows new data types, such as polygons, lines, money, time, arrays of floating point numbers, bit vectors, etc., to supplement the built-in data types in a data base system. In this paper we demonstrate the power of adding a data type corresponding to commands in a query language. We also propose three extensions to the query language QUEL to enhance its power in this augmented environment.

I INTRODUCTION

Abstract data types (ADTs) [LISK74, GUTT77] have been extensively investigated in a programming language context. Basically, an ADT is an encapsulation of a data structure (so that its implementation details are not visible to an outside client procedure) along with a collection of related operations on this encapsulated structure. The canonical example of an ADT is a stack with related operations: new, push, pop and empty.

The use of ADTs in a relational data base context has been discussed in [ROWE79, SCHM78, WASS79]. In these proposals a relation is considered an abstract data type whose implementation details are hidden from application level software. Allowable operations are defined by procedures written in a programming language that supports both data base access and ADTs. One use of this kind of data type is suggested in [ROWE79] and involves an EMPLOYEE abstract data type with related operations hire-employee, fire-employee and change-salary.

In [STON82, STON83] we presented an alternate use of ADTs. Instead of treating an entire relation as an ADT, we suggested that the individual columns of a relation be ADTs. This use of ADTs is a generalization of data base experts [STON80].

In Section II we briefly review our proposal and then in Section III we introduce QUEL as a data type and indicate desirable operators for this new type. Section IV turns to a discussion of three extensions to the QUEL language that are useful in this environment. In Section V we consider optimization issues related to QUEL ADTs. Lastly, we indicate that several data base problems including referential integrity, non-first normal form relations, and generalization hierarchies can be solved by defining QUEL as an abstract data type. Section VI presents our approach to these problems. Section VII concludes by summarizing the paper.

This Research was supported by the Navy Electronics Systems Command under Contract N00039-83-C-0243 and by the Air Force Office of Scientific Research under Grant 83-0021.

II ABSTRACT DATA TYPES

We explain our use of ADTs with an example concerning geometric objects. In computer aided design of integrated circuits, objects are often made up of rectangular boxes. For a VLSI data base one would like to define a column of a relation as type "box". For example, one might create a boxes relation as follows:

```
create boxes (owner = i4,
              layer = c15,
              box-desc = box-ADT)
```

Here, the boxes relation has three fields: the identifier of the circuit containing the box, the processing layer for the box (polysilicon, diffusion, etc.) and a description of the box's geometry. All fields are represented by built-in types except box-desc which is an ADT.

Tuples can be appended to this relation using QUEL [STON76] as follows:

```
append to boxes (owner = 99,
                 layer = "polysilicon",
                 box-desc = "0,0,2,3")
```

The built-in data types are converted to an internal representation and stored in a data base system. The string "0,0,2,3", represents the box bounded by x=0, y=0, x=2, y=3 and requires special recognition code. An input procedure must be available to the DBMS to perform the conversion of the character string "0,0,2,3" to an object with data type box-ADT. Such a routine is analogous to the procedure ascii-to-float which converts a character string to a floating point number.

It is desirable to have special operators for box-ADTs. for example, one would clip box dimensions as follows:

```
range of b is boxes
replace b (box-desc = b.box-desc * "0,0,4,1")
          where b.owner = 99
```

The * operator represents box intersection. In this case "0,0,4,1" will be converted to an object of type box-ADT, and a procedure must be available to perform box intersection between this ADT and b.box-desc.

In addition, one might want to define new comparison operations. For example, one might wish to define || as an operator meaning "overlaps". The || operator could then be used to return the boxes overlapping the unit square based at the origin as follows:

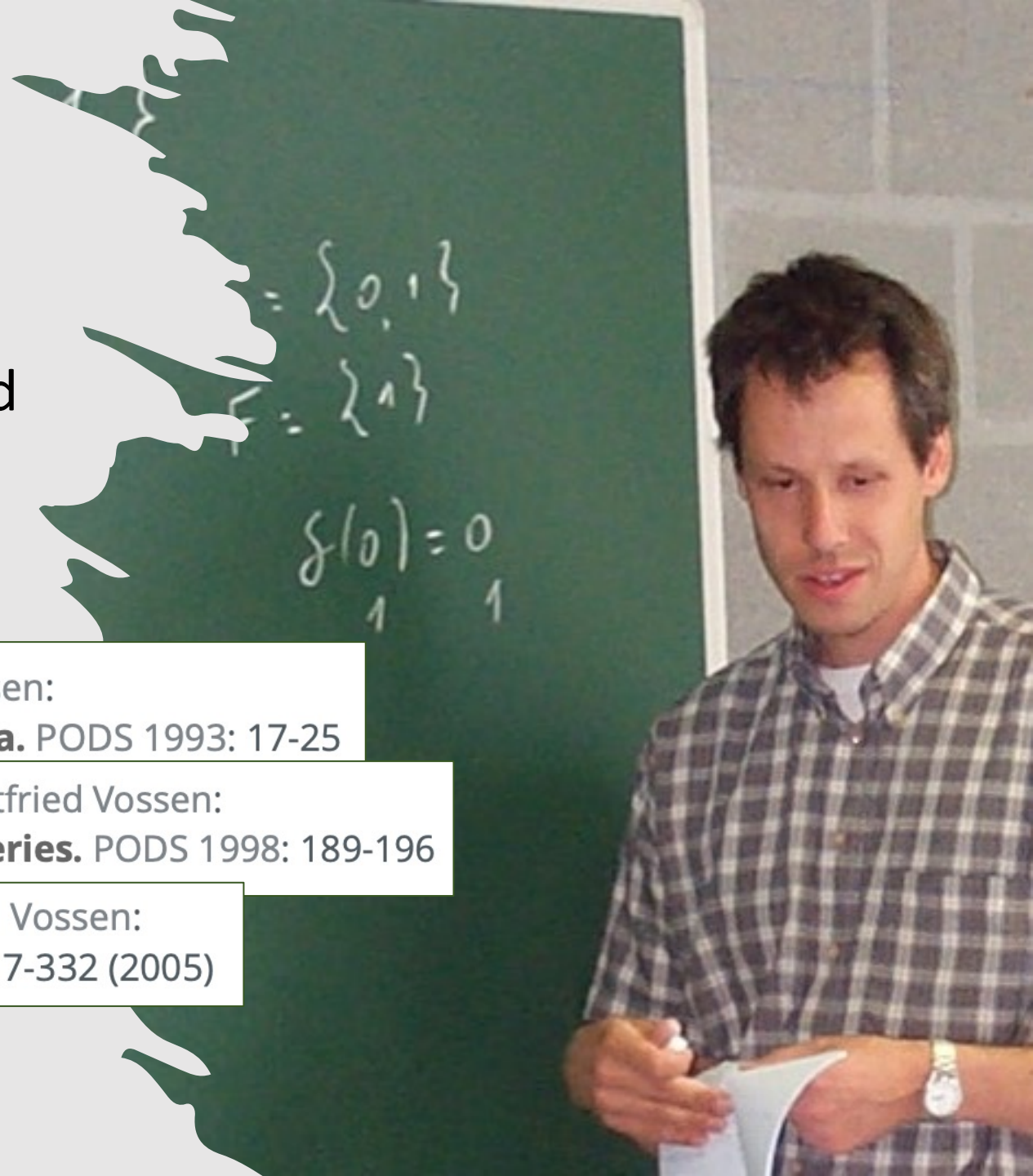
```
range of b is boxes
retrieve (b.box-desc)
          where b.box-desc || "0,0,1,1"
```

Again, a procedure is required for the overlap operator.

As a result an ADT contains the following elements:

My younger self

Inspired to develop logics and data models to formalize Stonebraker's and Gray's ideas

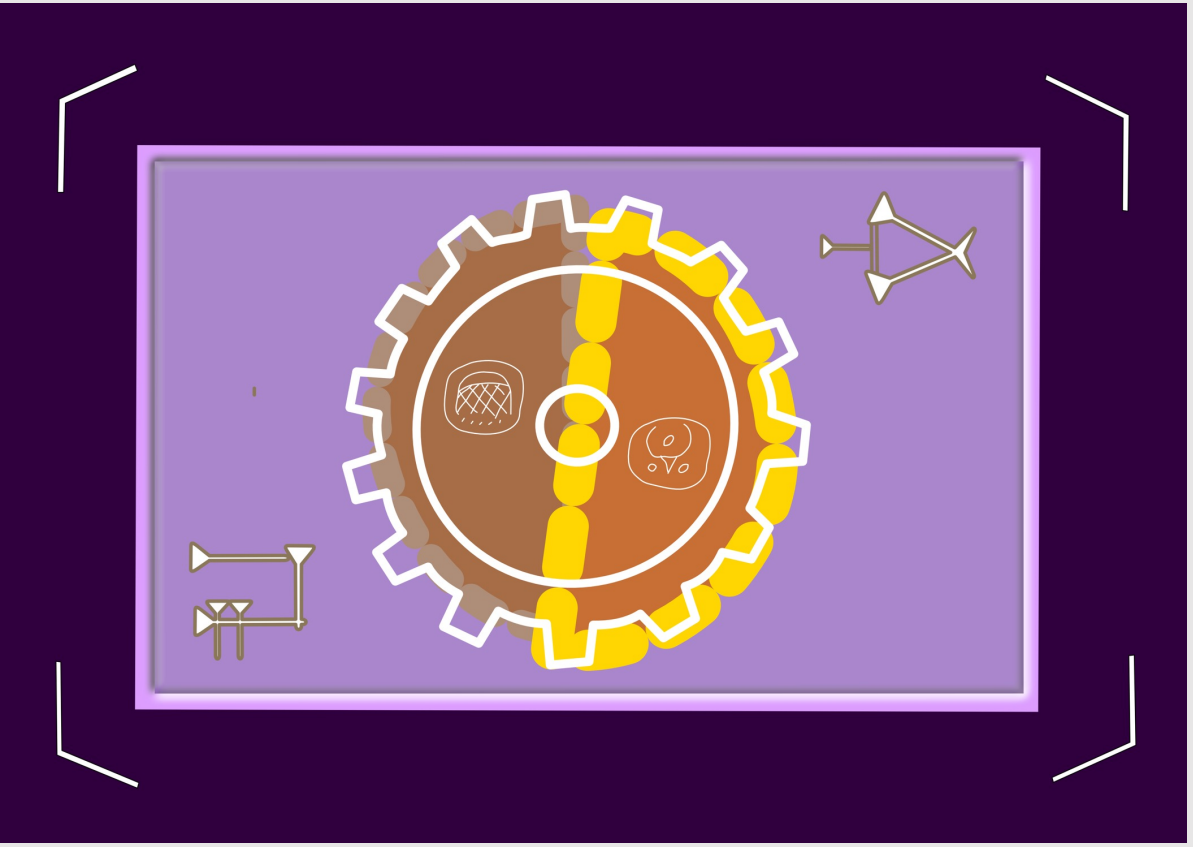


Jan Van den Bussche, Dirk Van Gucht, Gottfried Vossen:
Reflective Programming in the Relational Algebra. PODS 1993: 17-25

Frank Neven, Jan Van den Bussche, Dirk Van Gucht, Gottfried Vossen:
Typed Query Languages for Databases Containing Queries. PODS 1998: 189-196

Jan Van den Bussche, Stijn Vansummeren, Gottfried Vossen:
Towards practical meta-querying. Inf. Syst. 30(4): 317-332 (2005)

Outline



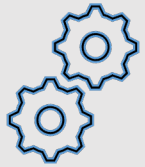
I. A step back; some history

II. Querying models: a taxonomy

III. SQL can verify neural networks

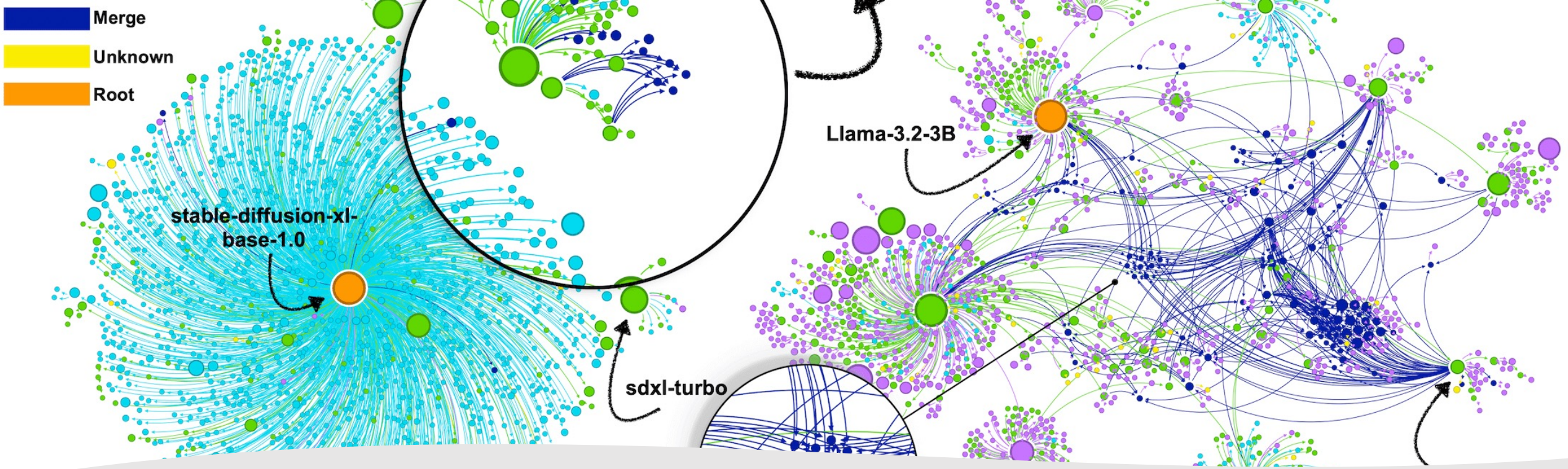
IV. Conclusions

Taxonomy of model-querying

1. Administrative 
2. Applicative 
3. Behavioral 
4. Looking inside 

Administrative model-querying

- ML platforms, e.g., MLflow
  - keep track of training data, deployments, dates, metadata, …
- Model Zoos, e.g., Hugging Face
  - public repository, millions of models
  - also called Model Lakes
- Query facility: limited to search & filtering on attributes



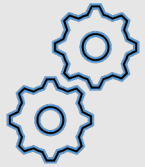
Weight spaces

- Model Atlas [Horwitz]
  - visualize model zoos
- Mine for structural patterns
- Predict properties
- Generative models
  - trained on models!

Taxonomy of model-querying

1. Administrative

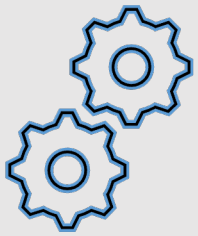
2. Applicative



3. Behavioral

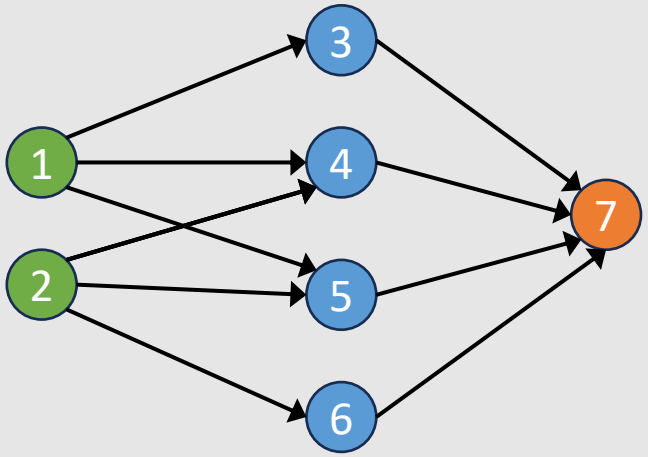
4. Looking inside

Applicative model-querying



- Calling models inside queries
  - cf. QUEL as a data type!
- Manage models (neural networks) in databases
  - “Data Management for End-to-End Machine Learning” 10th anniversary
- Compile model inferencing to SQL

👉 Alternatively: Express inference **uniformly** in SQL



Inputs	
1	i1
2	i2

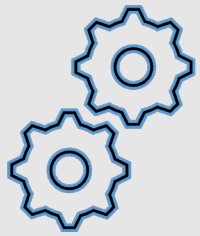
Edges		
1	3	w13
1	4	w14
1	5	w15
2	4	w24
2	5	w25
2	6	w26
3	7	w37
4	7	w47
5	7	w57
6	7	w57

Biases	
3	b3
4	b4
5	b5
6	b6
7	b7

Model(id, name, …)
Node(id, bias, **model_id**)
Edge(src, dst, weight, **model_id**)
Input(vec_id, in_id, value)

```
WITH RECURSIVE
input_nodes AS (/**/), output_nodes AS (/**/),
tx AS (
   SELECT
       i.model_id, v.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * v.val)) AS value,
       e.dst AS id
   FROM edge e
   JOIN input_nodes i ON i.id = e.src
   JOIN node n ON e.dst = n.id
   JOIN input v ON i.id = v.in_id
     AND v.model_id = i.model_id
   GROUP BY i.model_id, e.dst, n.bias, v.vec_id
   UNION ALL
   SELECT
       tx.model_id, tx.vec_id AS vec_id,
       GREATEST(0, n.bias + SUM(e.weight * tx.value)) AS value,
       e.dst AS id
   FROM edge e
   JOIN tx ON tx.id = e.src AND tx.model_id = e.model_id
   JOIN node n ON e.dst = n.id
   GROUP BY tx.model_id, e.dst, n.bias, tx.vec_id
),
t_out AS ( /* Join on last layer of tx and omit ReLU */ )
SELECT model_id, vec_id, output_value, output_node_id
FROM t_out
ORDER BY model_id, vec_id, output_node_id;
```

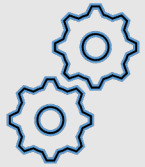
Applicative model-querying



- Calling models inside queries
  - cf. QUEL as a data type!
- Manage models (neural networks) in databases
  - “Data Management for End-to-End Machine Learning” 10th anniversary
- Compile to SQL
- Express inference uniformly in SQL
- Massive execution many models on many data [Alsatian]
- Backed by improvements and extensions of query processing

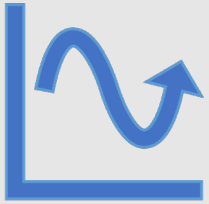
Taxonomy of model-querying

1. Administrative
2. Applicative
3. Behavioral
4. Looking inside



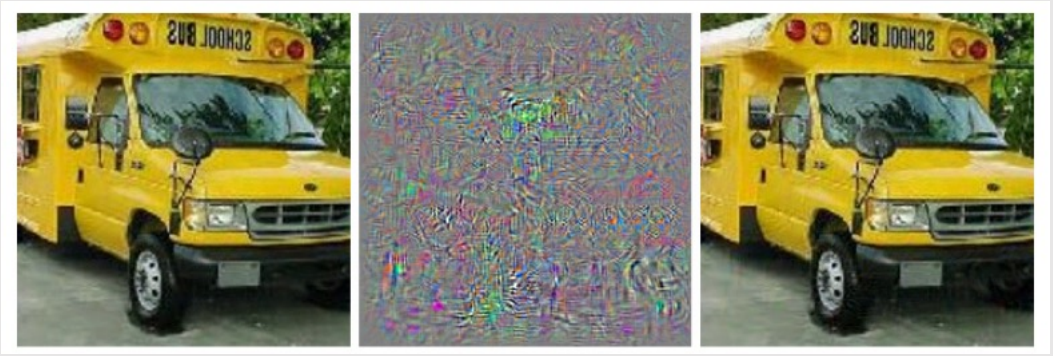
Well understood,
lots of systems efforts

Querying the behavior of a model



- A model represents a function $f: \mathbb{R}^d \rightarrow \mathbb{R}^p$
- **Robustness** around a given input $\boldsymbol{a}$:

$$\forall \boldsymbol{x}\ |\boldsymbol{x} - \boldsymbol{a}| < \delta \Rightarrow |f(\boldsymbol{x}) - f(\boldsymbol{a})| < \varepsilon$$

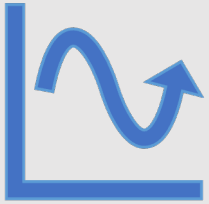


school bus

ostrich

[Szegedy et al 2013: Intriguing properties of neural networks]

Querying the behavior of a model

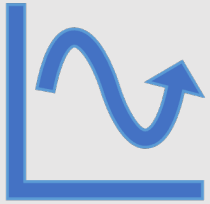


- A model represents a function $f: \mathbb{R}^d \rightarrow \mathbb{R}^p$
- **Robustness** around a given input $\boldsymbol{a}$:

$$\forall \boldsymbol{x}\ (|\boldsymbol{x} - \boldsymbol{a}| < \delta \Rightarrow |f(\boldsymbol{x}) - f(\boldsymbol{a})| < \varepsilon)$$

- Counterfactuals: $\exists \boldsymbol{x}\ (|\boldsymbol{x} - \boldsymbol{a}| < \delta \wedge |f(\boldsymbol{x}) - f(\boldsymbol{a})| > \varepsilon)$
- Boundedness: $\forall \boldsymbol{x}\ f(\boldsymbol{x}) < 120$
- Monotonicity: $\forall x_1 \forall x_2\ (x_1 < x_2 \Rightarrow f(x_1) < f(x_2))$
- Insensitivity: $\forall x \forall y \forall z\ |f(x, y) - f(x, z)| < \varepsilon$

Querying the behavior of a model



- A model represents a function $f: \mathbb{R}^d \rightarrow \mathbb{R}^p$
- **Robustness** around a given input $\boldsymbol{a}$:

$$\forall \boldsymbol{x}\,(|\boldsymbol{x}-\boldsymbol{a}| < \delta \Rightarrow |f(\boldsymbol{x}) - f(\boldsymbol{a})| \leq \varepsilon)$$

- Counterfactuals: $\exists \boldsymbol{x}\,(|\boldsymbol{x}-\boldsymbol{a}| < \delta \wedge |f(\boldsymbol{x}) - f(\boldsymbol{a})| > \varepsilon)$
- Boundedness: $\forall \boldsymbol{x}\, f(\boldsymbol{x}) < 120$
- Monotonicity: $\forall x_1 \forall x_2\,(x_1 < x_2 \Rightarrow f(x_1) < f(x_2))$
- Insensitivity: $\forall x \forall y \forall z\, |f(x,y) - f(x,z)| < \varepsilon$
- When models live inside the data system, verification becomes a kind of **analytical querying**

Neural network verification

Querying an infinite table

- Assume a classifier model $f: \mathbb{R}^d \rightarrow \{0,1\}$
- Can be thought of as an **infinite table** (with d numerical columns)

E.g. social network link prediction:

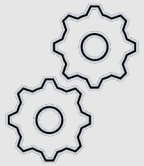
  - node = vector in $\mathbb{R}^4$
  - classifier $f: \mathbb{R}^4 \times \mathbb{R}^4 \rightarrow \{0,1\}$ so $d = 8$

- "Are $\boldsymbol{a}$ and $\boldsymbol{b}$ connected via a common friend?"

$$\exists \boldsymbol{x}\ (f(\boldsymbol{a}, \boldsymbol{x}) = 1 \wedge f(\boldsymbol{x}, \boldsymbol{b}) = 1)$$

  = **self-join** on an infinite table

Taxonomy of model-querying

1. Administrative 
2. Applicative 
3. Behavioral 
4. Looking inside 

Beyond model-agnostic querying

- Model-agnostic = only about the model **function**
- Look **inside** the model:
  - neural network pruning: useless connections or neurons
  - backpropagation as a model-query
  - computing **distance** between two models (Model Atlas)
    E.g. weight distance between their first layers
  - structure finding in model zoos

Taxonomy of model-querying

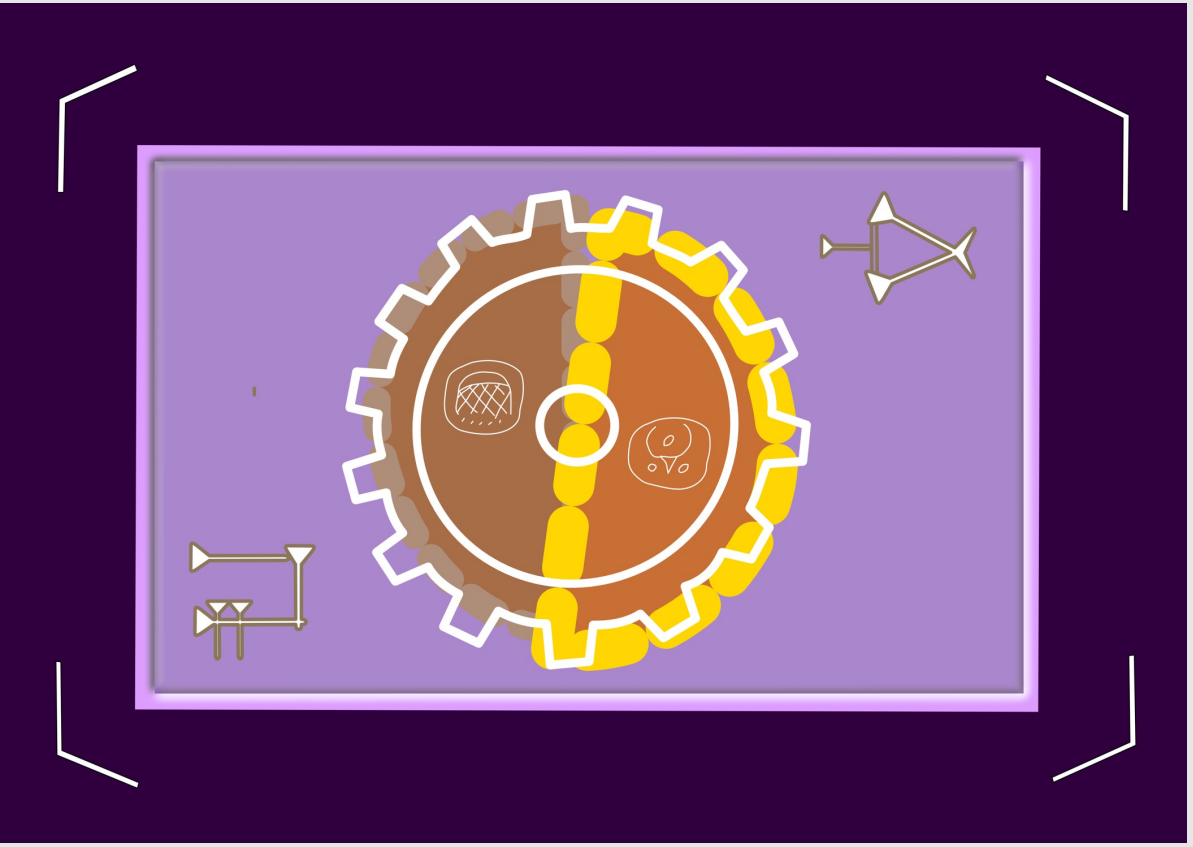
1. Administrative
2. Applicative
3. Behavioral
4. Looking inside

Lots of work from the DB perspective

Less so here … *

* for Boolean models: Q-DT-FOIL [Arenas et al]

Outline



I. A step back; some history
II. Querying models: a taxonomy
III. SQL can verify neural networks
IV. Conclusions

Verification of neural networks: A benchmark logic

- Querying a function $f: \mathbb{R}^d \to \mathbb{R}^p$
- **"Black-box" logic** FO$(\mathbb{R}_{lin}, f)$
  - Linear Real Arithmetic (LRA)
  - function symbol f
  - $\wedge$, $\vee$, $\neg$
  - $\exists$, $\forall$ over $\mathbb{R}$

Verification of neural networks: A benchmark logic

- Querying a function $f: \mathbb{R}^d \to \mathbb{R}^p$
- "Black-box" logic FO$(\mathbb{R}_{lin}, f)$
  - Linear Real Arithmetic (LRA)
  - function symbol f
  - $\wedge$, $\vee$, $\neg$
  - $\exists$, $\forall$ over $\mathbb{R}$

- Robustness: $\forall \boldsymbol{x}\ (|\boldsymbol{x} - \boldsymbol{a}| < \delta \Rightarrow |f(\boldsymbol{x}) - f(\boldsymbol{a})| < \varepsilon)$
- Counterfactuals: $\exists \boldsymbol{x}\ (|\boldsymbol{x} - \boldsymbol{a}| < \delta \wedge |f(\boldsymbol{x}) - f(\boldsymbol{a})| > \varepsilon)$
- Boundedness: $\forall \boldsymbol{x}\ f(\boldsymbol{x}) < 120$
- Monotonicity: $\forall x_1 \forall x_2\ (x_1 < x_2 \Rightarrow f(x_1) < f(x_2))$
- Insensitivity: $\forall x \forall y \forall z\ |f(x, y) - f(x, z)| < \varepsilon$
- Self-join: $\exists \boldsymbol{x}\ (f(\boldsymbol{a}, \boldsymbol{x}) = 1 \wedge f(\boldsymbol{x}, \boldsymbol{b}) = 1)$

Effectiveness of black-box logic

Assume given:

- Function f that **itself** is definable using LRA constraints
- Black-box query $Q(f)$

Then:

- The answer $Q(F)$ can be effectively computed by **LRA constraint solving**

- Z3 constraint solver

Some caveats

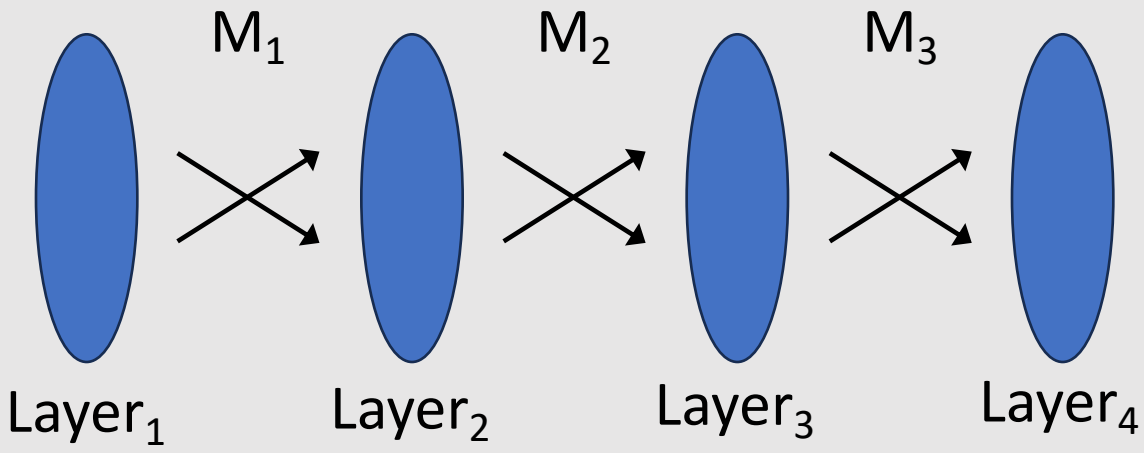
- If f is represented by a feedforward neural network with ReLU activations, then f is indeed LRA-definable
- So, black-box querying will be computable
- However, **complexity:** ReLU blows up formulas

$$y = \text{ReLU}(f) \Rightarrow ((f > 0 \wedge y = f) \vee (f \leq 0 \wedge y = 0))$$

- "ReLUplex" to the rescue [Katz]
  - solving ∃ quantifiers in LRA = linear programming (**Simplex** algorithm)
  - extend Simplex to deal with ReLU
- Even w/o ReLU, solving alternations of ∃ and ∀ still exponential

SQL as a “white-box” logic

- Feedforward neural network = finite sequence of matrices



- Store weight matrix in a table **M(row,col,weight)**
  - 👉 can use SQL to query neural networks

SQL can verify neural networks

Theorem: [Grohe, Standke, Steegmans, VdB ICDT 2025]

Every black-box logic query over feedforward ReLU-networks can be expressed in SQL

Warning:

Assumes **fixed depth** (but not layer sizes)

PODS 3

Day: Monday
Time: 2:30–4:00
Session: Query Languages
Session Chair: Matthias Lanzinger

Paper: Recursive querying of neural networks via weighted structures
Authors: Martin Grohe (RWTH Aachen University); Christoph Standke (RWTH Aachen University); Juno Steegmans (Hasselt University); Jan Van den Bussche (Hasselt University)*

Proof idea (1): PWL

- Relation M contains a network $\mathcal{N}$ representing $f_{\mathcal{N}}: \mathbb{R}^d \to \mathbb{R}$
- Every unit in $\mathcal{N}$ represents a **piecewise linear function** $\mathbb{R}^d \to \mathbb{R}$
  - **PWL**
- Write an SQL view PWL over M:
  - holds an LRA formula defining $f_{\mathcal{N}}$
  - encoded as a relation
  - PWL$(y, x_1, \ldots, x_d)$

Proof idea (2): Formula replacement

- Let φ be a FO$(\mathbb{R}_{lin}, f)$ formula: $\varphi \equiv \exists z\, \psi(z)$
- Formula ψ contains atoms $v = f(\boldsymbol{u})$
- Replace these atoms by the result of PWL$(v, \boldsymbol{u})$

Proof idea (3): LRA constraint solving

- Ferrante–Rackoff algorithm
- Can be simulated in SQL!

Comments:

- Proof encodes models ánd LRA formulas in relational structures
- Technique of “interpreting one structure into another” from **model theory** (mathematical logic)
  - We extend it to numerical SQL-like logics
- Model theory for models 😉

Does it work in practice?

- Theoretical construction produces very complex SQL expressions

👉 Opportunities for clever systems work, better algorithms

- One idea: use **geometric representations** of the PWL view
  - piecewise linear function $\mathbb{R}^d \to \mathbb{R}$ partitions space in **polytopes**

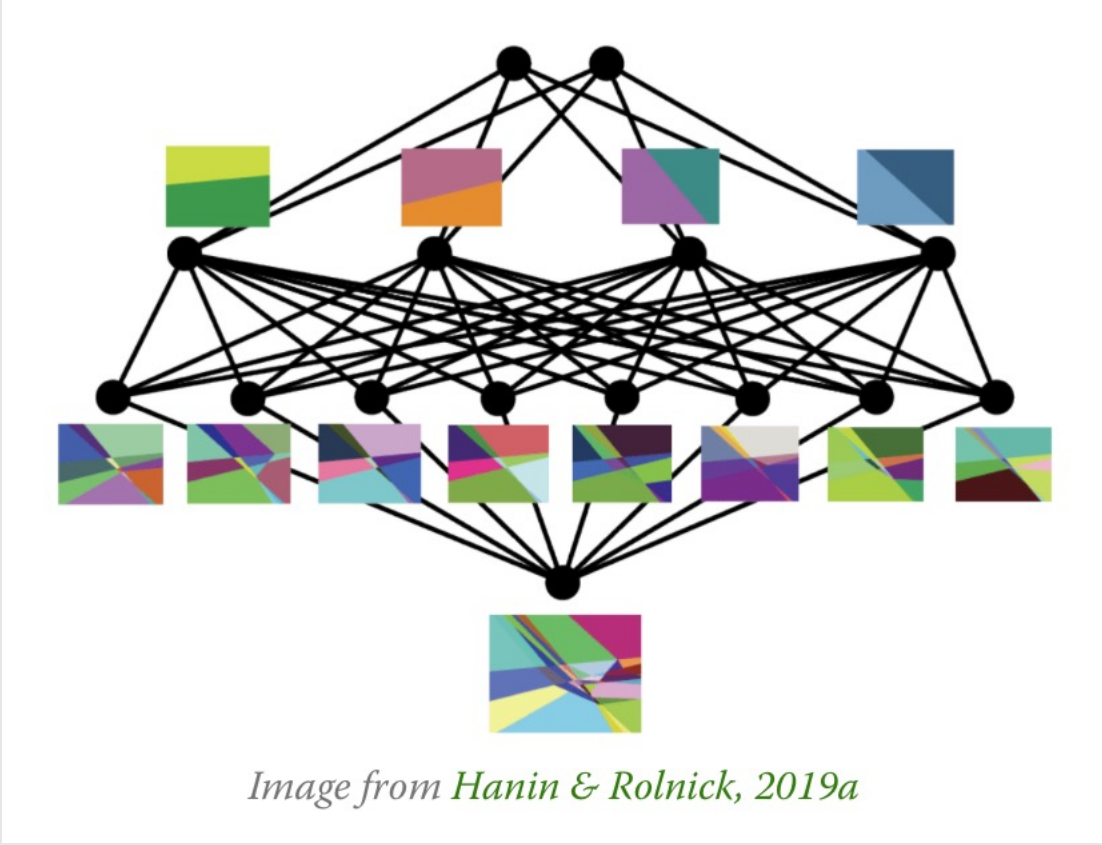
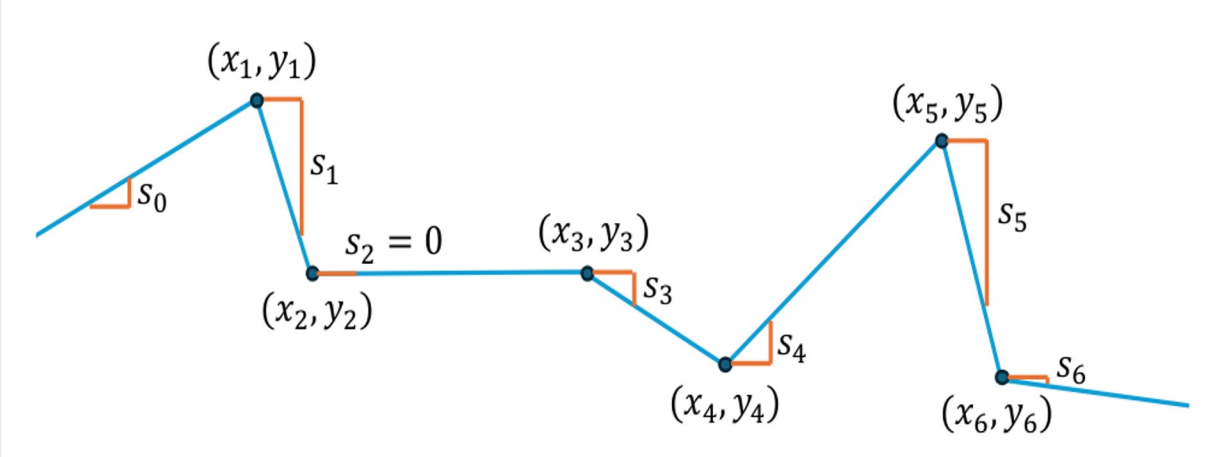


Image from Hanin & Rolnick, 2019a

Illustration: the 1D case, one hidden layer

$$break_x := \frac{-b(u)}{w(in_1, u)}$$




```
WITH /* ... */
breakpoint_values AS (
    SELECT (-n.bias) / e.weight AS break_x,
    FROM node n
    JOIN edge e ON e.dst = n.id
    JOIN input_nodes i ON e.src = i.id
    JOIN hidden_nodes h ON h.id = n.id
    WHERE e.weight <> 0
    GROUP BY break_x, n.id
    ORDER BY break_x
),
/* ... */
points_and_slopes AS (
    SELECT
        u1_break_x AS x,
        u1_break_y AS y,
        (u2_break_y - u1_break_y) / (u2_break_x - u1_break_x)
AS slope
    FROM breakpoint_pairs
    ORDER BY x
)
SELECT * FROM points_and_slopes;
```

Verifying monotonicity (1D, one hidden layer)

$$\forall x \forall x' (x \geq x') \implies (F(x) \geq F(x'))$$

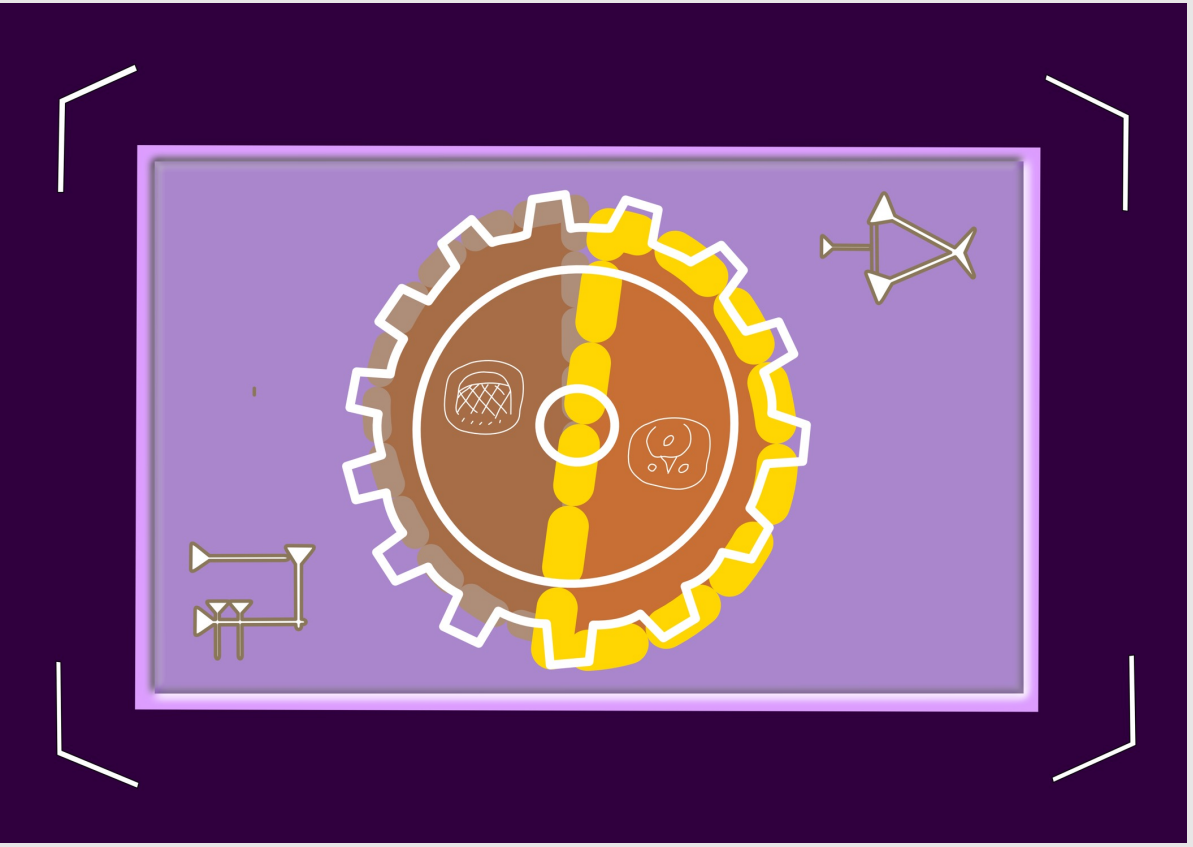
```
SELECT NOT EXISTS(
    -- Search for counterexample
    SELECT 1
    FROM points_and_slopes
    -- Closest breakpoint left of lower
bound
    WHERE x >= (SELECT MAX(x)
                FROM points_and_slopes
                WHERE x < -1)
    AND x < 1
    AND slope < 0
) AS 'Monotonicity property holds';
```

Verifying boundedness (1D, one hidden layer)

$$\forall x(x \geq -2\pi \wedge x \leq 2\pi)$$
$$\implies (F(x) \geq -1 \wedge F(x) \leq 1)$$

```
WITH all_points_in_range AS (
    SELECT x, y FROM points_and_slopes
    WHERE x >= -6.28 AND x <= 6.28
    UNION ALL
    -- Lower bound of the input range.
    SELECT -6.28, (-6.28 - x) * slope + y AS y
    FROM points_and_slopes
    WHERE x = (SELECT MAX(x) FROM points_and_slopes WHERE x < -6.28)
    UNION ALL
    -- Upper bound of the input range.
    SELECT 6.28, (6.28 - x) * slope + y AS y
    FROM points_and_slopes
    WHERE x = (SELECT MIN(x) FROM points_and_slopes WHERE x > 6.28)
)
SELECT NOT EXISTS(
    SELECT 1
    FROM all_points_in_range
    WHERE (y < -1 OR y > 1)
) AS 'Boundedness property holds';
```

Outline

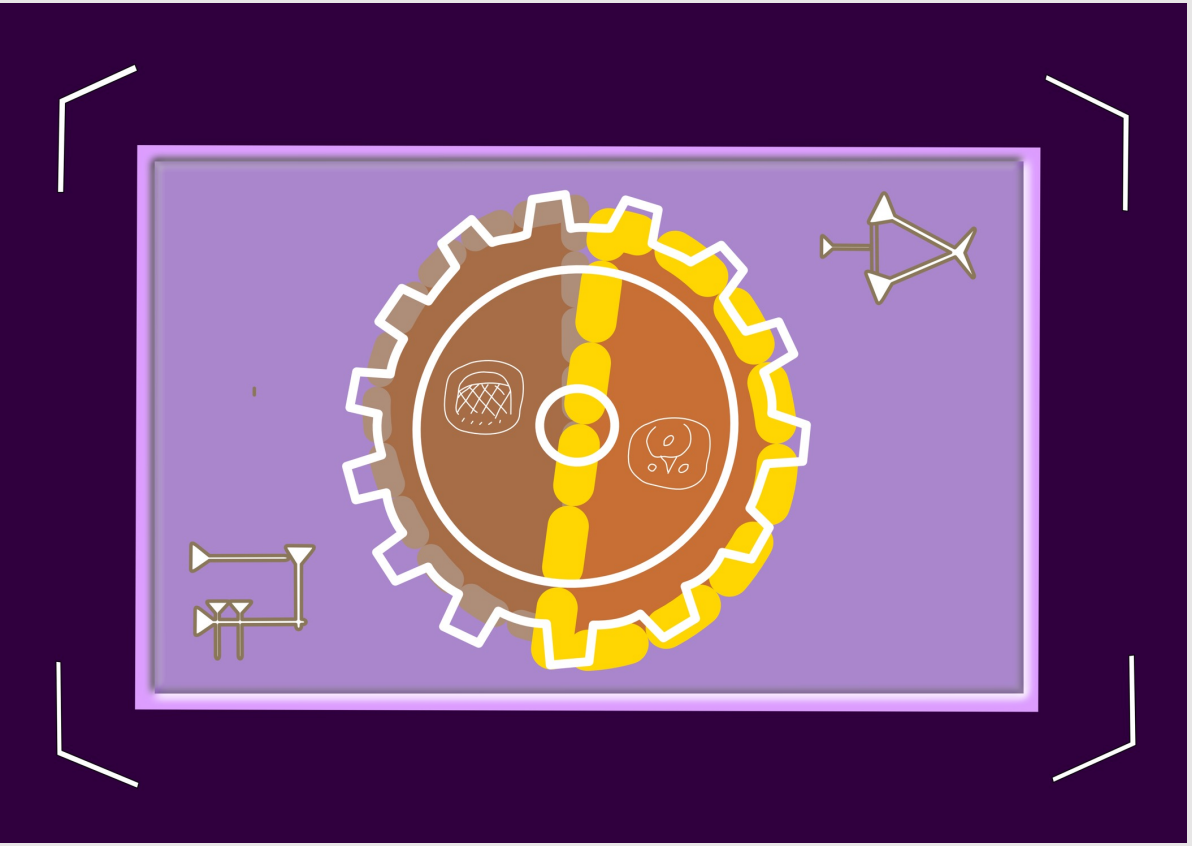


I. A step back; some history
II. Querying models: a taxonomy
III. SQL can verify neural networks
IV. Conclusions

Conclusions

- New class of complex analytical queries
- Many **challenges, open problems:**
- Design data models, query languages
- Other model classes, e.g., Transformer
- Expressive power (constraint query languages)
- Query processing, optimization
- Querying model zoos, **joins** in model zoos
- Querying code!

Cover art



- Original art created by art collective Robbert&Frank Frank&Robbert
- The scripts in bottom left and top right are cuneiform and signify “the mythical box at our service,” i.e., a black-box model opened for querying.
- The glyphs in the disk are the Mayan syllables ‘al’ and ‘ap’. In the Chuj language, ‘alap’ means ‘to speak’, representing our focus on languages.
- The two halves of the brain represent the importance of logic and rigor as well as of creativity and inspiration.
- Gears, disks suggest code, automation, as data.
- The mirror boundaries (from gaming), represent reflection and bounding of expressive power. The open V also denotes the unknown in mathematics.