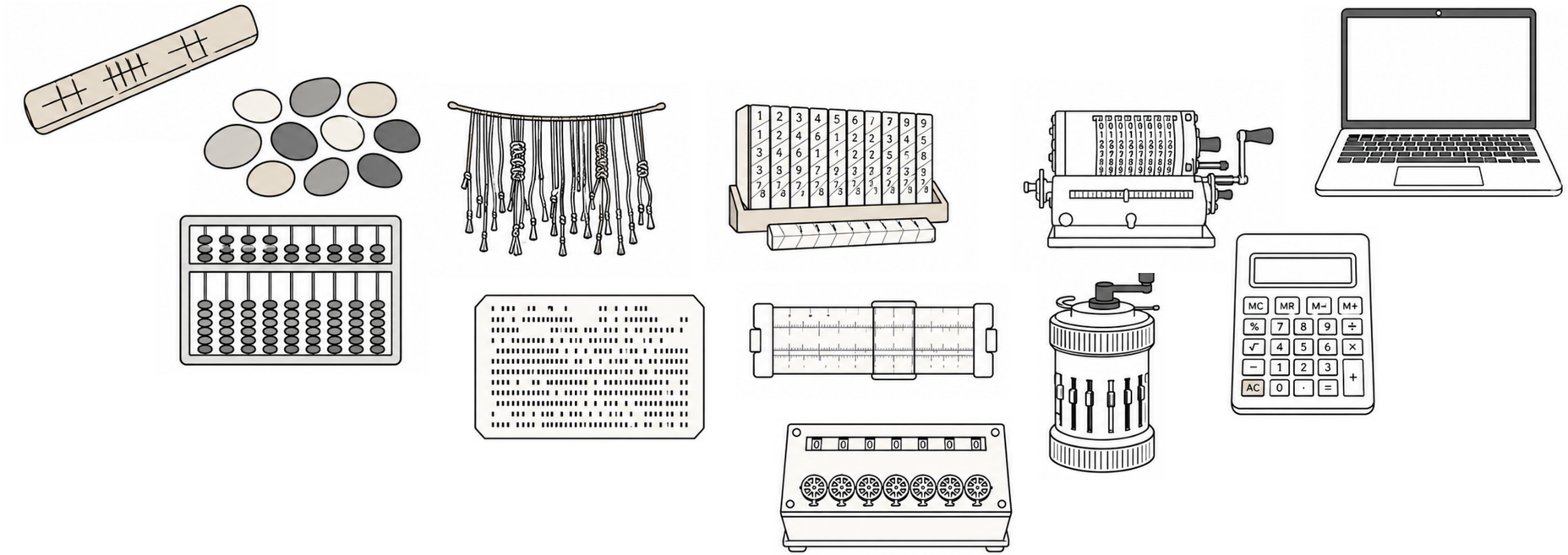
Computational Algebra with Transformers: What Deep Learning Adds to Computational Algebra

Hiroshi Kera (Chiba University / National Institute of Informatics)

Symbolic Computation and Machine Learning 2026 *Jul. 06-08, 2026*

Tools for Computation



- Tools have scaled up computation.
- Observing many instances leads to discovery of computational rules and heuristics.

Research Q.

Can we leverage *learning* as a new computational tool?

This Survey Talk

Goal

Overviewing learning symbolic computation.

- Suggesting the first entry point of SC x ML for newcomers
- Offering a new entry point of SC x ML researchers

What this talk *doesn't* cover

- Details of the experiments and the "good results".
- Theorem proving
- LLM-based reasoning

Symbolic Computation x Machine Learning

Seminal works have already tried machine learning in symbolic computation.

- CAD variable order selection[Huang+'14'19]
- S-pair selection in Buchberger algorithm[Peifer+'20]

Symbolic Computation x Machine Learning

Seminal works have already tried machine learning in symbolic computation.

- CAD variable order selection[Huang+'14'19]
- S-pair selection in Buchberger algorithm[Peifer+'20]

However, classical ML lacks flexibility with limited learning ability for math tasks.

Symbolic Computation x Machine Learning

Seminal works have already tried machine learning in symbolic computation.

- CAD variable order selection[Huang+'14'19]
- S-pair selection in Buchberger algorithm[Peifer+'20]

However, classical ML lacks flexibility with limited learning ability for math tasks.

Transformer[Vaswani+'17]: an efficient sequence-to-sequence model.

$$\mathcal{T} : \Sigma^* \times \Sigma^* \to \Sigma^*$$

Σ : Set of all tokens ("vocabulary")

Σ^* : Set of all sequences of tokens

- **Vast applications:** sequence of words (language), pixels (images), states & actions (game)…
- **Symbolic computation, too**[Lample & Charton'20]

Autoregressive Generation & Attention

$$\mathcal{T} : \Sigma^* \times \Sigma^* \to \Sigma^*$$

Autoregressive Generation & Attention

$$\mathcal{T} : \Sigma^* \times \Sigma^* \to \Sigma^*$$

Autoregressive generation: task breakdown

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>}]) \quad \to \quad 8$$

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>},\ 8]) \quad \to \quad 3$$

$$\vdots$$

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>},\ 8,\ 3,\ 7, 3]) \quad \to \quad \texttt{<end>}$$

Autoregressive Generation & Attention

$\mathcal{T} : \Sigma^* \times \Sigma^* \to \Sigma^*$

Autoregressive generation: task breakdown

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>}]) \quad \to \quad 8$$

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>},\ 8]) \quad \to \quad 3$$

$$\vdots$$

$$\mathcal{T}([5,\ 9,\ 1,\ 2,\ +,\ 2,\ 4,\ 6,\ 1], [\texttt{<start>},\ 8,\ 3,\ 7, 3]) \quad \to \quad \texttt{<end>}$$

Attention module: input-adaptive weights

$X \in \mathbb{R}^{\ell \times d}$: sequence of *d*-dimensional vectors in length ℓ (corrsp. to ℓ tokens)

(Attention layer; simplified) $\mathcal{A}(X) = \underbrace{\left((XW_{\mathrm{Q}})^\top XW_{\mathrm{K}}\right)}_{\mathbb{R}^{\ell \times \ell}\ \textbf{Input-adaptive}} X$ $\qquad W_{\mathrm{Q}}, W_{\mathrm{K}} \in \mathbb{R}^{d \times d}$ **trainable weights**

Case Study #0

Caveats of Symbolic Computation x Transformers

Learning to Compute Gröbner bases

Hiroshi Kera, Yuki Ishihara, Yuta Kambe, Tristan Vaccon, Kazuhiro Yokoyama

NeurIPS 2024

Overview

Polynomial set F

$$\begin{cases} f_1 = & x^3 - 3x^2 - y + 1 \\ f_2 = & -x^2 + y^2 - 1 \end{cases}$$

$\xrightarrow{\mathcal{T}}$ **Transformer**

Gröbner basis G

$$\begin{cases} g_1 = & x^2 - y^2 + 1 \\ g_2 = & xy - x - y^4 + 11y^2 + 3y - 13 \\ g_3 = & y^5 + y^4 - 11y^3 - 17y^2 + 9y + 17 \end{cases}$$

- **Restricted to ideals in shape position:** $G = \{h(x_n), x_1 - g_1(x_n), \ldots, x_{n-1} - g_{n-1}(x_n)\}$
- **Simple tokenization:** $\{x^2 + y,\ 2y\} \quad \rightarrow \quad [\texttt{C1}, \texttt{E2}, \texttt{E0}, \texttt{+}, \texttt{C1}, \texttt{E0}, \texttt{E1}, \texttt{|}, \texttt{C2}, \texttt{E0}, \texttt{E1}]$

Caveats

GB generation [accuracy / support accuracy]

Ring	$n = 2, \sigma = 1$	$n = 3, \sigma = 0.6$	$n = 4, \sigma = 0.3$	$n = 5, \sigma = 0.2$
$\mathbb{Q}[x_1, \ldots, x_n]$	94.6 / 97.9	96.1 / 98.6	96.2 / 98.6	91.8 / 97.9
$\mathbb{F}_7[x_1, \ldots, x_n]$	66.6 / 76.6	78.8 / 87.6	80.9 / 91.1	83.2 / 91.4
$\mathbb{F}_{31}[x_1, \ldots, x_n]$	44.7 / 82.7	58.5 / 89.3	73.9 / 93.9	80.0 / 93.4

- Observing 100x accelaration by Transformer against F4 for a few instances.

Caveats

GB generation [accuracy / support accuracy]

Ring	$n=2,\sigma=1$	$n=3,\sigma=0.6$	$n=4,\sigma=0.3$	$n=5,\sigma=0.2$
$\mathbb{Q}[x_1,\ldots,x_n]$	94.6 / 97.9	96.1 / 98.6	96.2 / 98.6	91.8 / 97.9
$\mathbb{F}_7[x_1,\ldots,x_n]$	66.6 / 76.6	78.8 / 87.6	80.9 / 91.1	83.2 / 91.4
$\mathbb{F}_{31}[x_1,\ldots,x_n]$	44.7 / 82.7	58.5 / 89.3	73.9 / 93.9	80.0 / 93.4

- Observing 100x accelaration by Transformer against F4 for a few instances.

Limited accuracy over finite fields

Caveats

GB generation [accuracy / support accuracy]

Ring	$n=2,\sigma=1$	$n=3,\sigma=0.6$	$n=4,\sigma=0.3$	$n=5,\sigma=0.2$
$\mathbb{Q}[x_1,\ldots,x_n]$	94.6 / 97.9	96.1 / 98.6	96.2 / 98.6	91.8 / 97.9
$\mathbb{F}_7[x_1,\ldots,x_n]$	66.6 / 76.6	78.8 / 87.6	80.9 / 91.1	83.2 / 91.4
$\mathbb{F}_{31}[x_1,\ldots,x_n]$	44.7 / 82.7	58.5 / 89.3	73.9 / 93.9	80.0 / 93.4

- Observing 100x accelaration by Transformer against F4 for a few instances.

Limited accuracy over finite fields

Many errors in coefficients

Caveats

Small *n* due to the simple tokenization (i.e., long input sequence)

GB generation [accuracy / support accuracy]

Ring	$n=2,\sigma=1$	$n=3,\sigma=0.6$	$n=4,\sigma=0.3$	$n=5,\sigma=0.2$
$\mathbb{Q}[x_1,\ldots,x_n]$	94.6 / 97.9	96.1 / 98.6	96.2 / 98.6	91.8 / 97.9
$\mathbb{F}_7[x_1,\ldots,x_n]$	66.6 / 76.6	78.8 / 87.6	80.9 / 91.1	83.2 / 91.4
$\mathbb{F}_{31}[x_1,\ldots,x_n]$	44.7 / 82.7	58.5 / 89.3	73.9 / 93.9	80.0 / 93.4

- Observing 100x accelaration by Transformer against F4 for a few instances.

Limited accuracy over finite fields

Many errors in coefficients

Caveats

No guarantee on giving GB of F

Only Ideals in shape position

Small *n* due to the simple tokenization (i.e., long input sequence)

GB generation [accuracy / support accuracy]

Ring	$n=2,\sigma=1$	$n=3,\sigma=0.6$	$n=4,\sigma=0.3$	$n=5,\sigma=0.2$
$\mathbb{Q}[x_1,\ldots,x_n]$	94.6 / 97.9	96.1 / 98.6	96.2 / 98.6	91.8 / 97.9
$\mathbb{F}_7[x_1,\ldots,x_n]$	66.6 / 76.6	78.8 / 87.6	80.9 / 91.1	83.2 / 91.4
$\mathbb{F}_{31}[x_1,\ldots,x_n]$	44.7 / 82.7	58.5 / 89.3	73.9 / 93.9	80.0 / 93.4

- Observing 100x accelaration by Transformer against F4 for a few instances.

Limited accuracy over finite fields

Many errors in coefficients

Three Case Studies

Case study #1 [Kera-Pelleriti+'25]

Computational Algebra with Attention: Transformer Oracles for Border Basis Algorithms

- Algorithm + learning-based oracle
- Monomial-level embedding

Case study #2 [Wenger+'22]

SALSA: Attacking Lattice Cryptography with Transformers

- Utility of weak learners
- Learning techniques of arithmetic tasks
- Autoregressive generation is powerful given adequate task decomposition.

Case study #3 [Hruby+'22]

Learning to Solve Hard Minimal Problems

- Classification to initial systems (not generation of them)

Case Study #1

Symbolic Algorithm x Transformer Oracle

Computational Algebra with Attention: Transformer Oracles for Border Basis Algorithms

Hiroshi Kera*, Nico Pelleriti*, Yuki Ishihara, Max Zimmer, Sebastian Pokutta

NeurIPS 2025

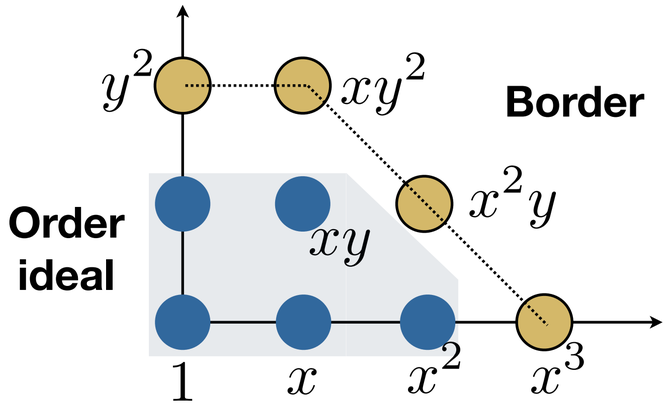
Border bases[Kehrein-Kreuzer-Robbiano'05]

Polynomial ring $K[X] := K[x_1, \ldots, x_n]$

Set of all terms $\mathbb{T}[X]$

Order ideal: a set of terms $\mathcal{O} \subset \mathbb{T}[X]$ such that: $\forall t \in \mathcal{O},\ (\text{all divisors of } t) \subset \mathcal{O}$

Border of $\mathcal{O}$: a set of terms $\partial\mathcal{O} = \left(\bigcup_i x_i\mathcal{O}\right) \setminus \mathcal{O}$



$\mathcal{O}$ -border basis: a set of polynomials of the form

$$G = \left\{ b - \sum_{t \in \mathcal{O}} c_{b,t} t \mid b \in \partial\mathcal{O} \right\}$$

Border — order terms

and $K[X] = \langle G \rangle \oplus \langle O \rangle_K$

- **A generalization of Gröbner bases of 0-dim ideals.**

Oracle Border Basis Algorithm

Border basis algorithm (sketch) [Kehrein & Kreuzer'06]

$L_d = \mathbb{T}[X]_{\leq d}$: set of terms up to degree-*d*

$V = \{v_1, \ldots, v_s\}$: basis of the vector space $\langle V \rangle_K \subset \langle L_d \rangle_K$

Expand the vector space to $\langle V \cup x_1 V \cup \cdots \cup x_n V \rangle_K \cap \langle L_d \rangle_K$

- **(Need basis extension)** → Extend V and expand again.
- **(Otherwise)** → Terminate or enlarge L_d to L_{d+1} and expand again.

Oracle Border Basis Algorithm

Border basis algorithm (sketch) [Kehrein & Kreuzer'06]

$L_d = \mathbb{T}[X]_{\leq d}$: set of terms up to degree-*d*

$V = \{v_1, \ldots, v_s\}$: basis of the vector space $\langle V \rangle_K \subset \langle L_d \rangle_K$

Expand the vector space to $\langle V \cup x_1 V \cup \cdots \cup x_n V \rangle_K \cap \langle L_d \rangle_K$

- **(Need basis extension)** → Extend V and expand again.
- **(Otherwise)** → Terminate or enlarge L_d to L_{d+1} and expand again.

The **full expansion** leads to many wastful Gaussian elimination!

Oracle Border Basis Algorithm

Border basis algorithm (sketch) [Kehrein & Kreuzer'06]

$L_d = \mathbb{T}[X]_{\leq d}$: set of terms up to degree-*d*

$V = \{v_1, \ldots, v_s\}$: basis of the vector space $\langle V \rangle_K \subset \langle L_d \rangle_K$

Expand the vector space to $\langle V \cup x_1 V \cup \cdots \cup x_n V \rangle_K \cap \langle L_d \rangle_K$

- **(Need basis extension)** → Extend V and expand again.
- **(Otherwise)** → Terminate or enlarge L_d to L_{d+1} and expand again.

The **full expansion** leads to many wastful Gaussian elimination!

Oracle

$\textsc{Oracle}(V, L_d) = \{(x_j, v_k)\} \subset X \times V$ (suggesting expansion pairs)

*Inaccurate prediction can be resolved by ocational full expansion as a fallback.

Observation: An oracle with truncation of $v_k \in V$ to l-leading terms works well:

Each polynomial is truncated to its first l leading terms

Near-perfect prediction on termination

Field	Variables	l	Precision (%)	Recall (%)	F1 Score (%)	No Expansion Acc. (%)
$\mathbb{F}_{31}$	$n = 3$	1	84.4	86.8	85.6	99.7
		3	89.4	90.0	89.7	99.7
		5	91.6	93.2	92.4	99.7
	$n = 4$	1	90.7	91.6	91.1	98.8
		3	92.9	93.7	93.3	98.8
		5	94.2	94.7	94.4	98.8
	$n = 5$	1	92.7	93.1	92.9	99.6
		3	94.3	94.6	94.4	99.6
		5	94.8	95.3	95.1	99.6

Impact of l is moderate

- Restrictions to a few leading terms make the oracle generalizable to “longer” polynomials.

Monomial embedding

$$\{x^2 + y,\ 2y\} \quad \rightarrow \quad [\texttt{C1}, \texttt{E2}, \texttt{E0}, \texttt{+}, \texttt{C1}, \texttt{E0}, \texttt{E1}, \texttt{|}, \texttt{C2}, \texttt{E0}, \texttt{E1}]$$

Monomial embedding

$$\{x^2 + y,\ 2y\} \quad \rightarrow \quad [\texttt{C1}, \texttt{E2}, \texttt{E0}, \texttt{+}, \texttt{C1}, \texttt{E0}, \texttt{E1}, \texttt{|}, \texttt{C2}, \texttt{E0}, \texttt{E1}]$$

↓ Standard input embedding

$$v_{\texttt{C0}}, v_{\texttt{E2}}, v_{\texttt{E0}} \in \mathbb{R}^d$$

Monomial embedding

$$\{x^2 + y,\ 2y\} \quad \rightarrow \quad [\texttt{C1}, \texttt{E2}, \texttt{E0}, \texttt{+}, \texttt{C1}, \texttt{E0}, \texttt{E1}, \texttt{|}, \texttt{C2}, \texttt{E0}, \texttt{E1}]$$

↓ Standard input embedding

$$v_{\texttt{C0}}, v_{\texttt{E2}}, v_{\texttt{E0}} \in \mathbb{R}^d$$

↓ ***Monomial embedding***

$$v_{x^2} := (v_{\texttt{C0}}, v_{\texttt{E2}}, v_{\texttt{E0}})/3$$

Monomial embedding

$$\{x^2 + y,\ 2y\} \quad \rightarrow \quad [\texttt{C1}, \texttt{E2}, \texttt{E0}, \texttt{+}, \texttt{C1}, \texttt{E0}, \texttt{E1}, \texttt{|}, \texttt{C2}, \texttt{E0}, \texttt{E1}]$$

↓ Standard input embedding

$$v_{\texttt{C0}}, v_{\texttt{E2}}, v_{\texttt{E0}} \in \mathbb{R}^d$$

↓ ***Monomial embedding***

$$v_{x^2} := (v_{\texttt{C0}}, v_{\texttt{E2}}, v_{\texttt{E0}})/3$$

Benchmark for learning cummulative polynomial product over $\mathbb{F}_p(X)$

# Variables	Method	Success rate (%)	GPU memory (MB)
$n = 2$	standard	33.9	4,302
	monomial	**39.7**	**1,678**
$n = 3$	standard	37.5	11,296
	monomial	**44.6**	**2,408**
$n = 4$	standard	38.8	23,632
	monomial	**47.3**	**3,260**
$n = 5$	standard	22.0	27,442
	monomial	**53.7**	**3,424**

Higher succes rate & memory efficiency (reduced by $\mathcal{O}(n^2)$***)***

Takeaway

Algorithm + learning-based oracle (with fallback)

- Certified the correctness, empirical accelaration.

Truncation to leading terms

- Heuristics for learning-based oracle.

Monomial embedding

- Prior knowledge on tokenization & embedding improves training & learning efficiency.
- More aggresive verision by [Malhou+'26]

Case Study #2

Utility of Weak Learners

SALSA:
Attacking Lattice Cryptography with Transformers

Emily Wenger, Mingjie Chen, François Charton, Kristin Lauter

NeurIPS 2022

SALSA

Approach

Secret key recovery through behavioral analysis of a weak learner

Learning with Error (LWE): given samples $\{(\boldsymbol{a}_i, b_i)\}_i$, recover the secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ from

$$b_i = \boldsymbol{a}_i \cdot \boldsymbol{s} + \epsilon_i \bmod q, \quad \boldsymbol{a}_i \text{ uniform in } \mathbb{Z}_q^n, \quad \epsilon_i \text{ small noise}$$

- **The noise ϵ_i makes the recovery very hard.**

$\boldsymbol{s}$

SALSA

Approach

Secret key recovery through behavioral analysis of a weak learner

Learning with Error (LWE): given samples $\{(\boldsymbol{a}_i, b_i)\}_i$, recover the secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ from

$$b_i = \boldsymbol{a}_i \cdot \boldsymbol{s} + \epsilon_i \bmod q, \quad \boldsymbol{a}_i \text{ uniform in } \mathbb{Z}_q^n, \quad \epsilon_i \text{ small noise}$$

- **The noise ϵ_i makes the recovery very hard.**

The *SALSA* attack trains a weak model $\mathcal{M} : \boldsymbol{a} \mapsto b$ with $\{(\boldsymbol{a}_i, b_i)\}_i$ (with a fixed, unknown $\boldsymbol{s}$)

SALSA attack

For a large $K > 0$, we should have $b = (K\boldsymbol{e}_i) \cdot \boldsymbol{s} + \epsilon = K s_i + \epsilon$.

- **If $\mathcal{M}(K\boldsymbol{e}_i)$ is consistently small for several Ks, then maybe $s_i = 0$.**

Learning modular addition

Learning the parity function is hard[Shalev-Shwartz+'17, Hahn+'24].

$$[x_1, \ldots, x_N] \in \{0, \ldots, q-1\}^N \quad \mapsto \quad \left(\sum_{i=1}^{N} x_i \right) \mod q$$

Learning modular addition

Learning the parity function is hard[Shalev-Shwartz+'17, Hahn+'24].

$$[x_1, \ldots, x_N] \in \{0, \ldots, q-1\}^N \quad \mapsto \quad \left(\sum_{i=1}^{N} x_i \right) \mod q$$

[Saxena+'25]*

Train on instances with many 0s!

Test on instances
$x \sim \mathbb{U}[0, q-1]^N$

- Distirubutional gap between train/test sets.

Learning modular addition

Learning the parity function is hard[Shalev-Shwartz+'17, Hahn+'24].

$$[x_1, \ldots, x_N] \in \{0, \ldots, q-1\}^N \quad \mapsto \quad \left(\sum_{i=1}^{N} x_i \right) \mod q$$

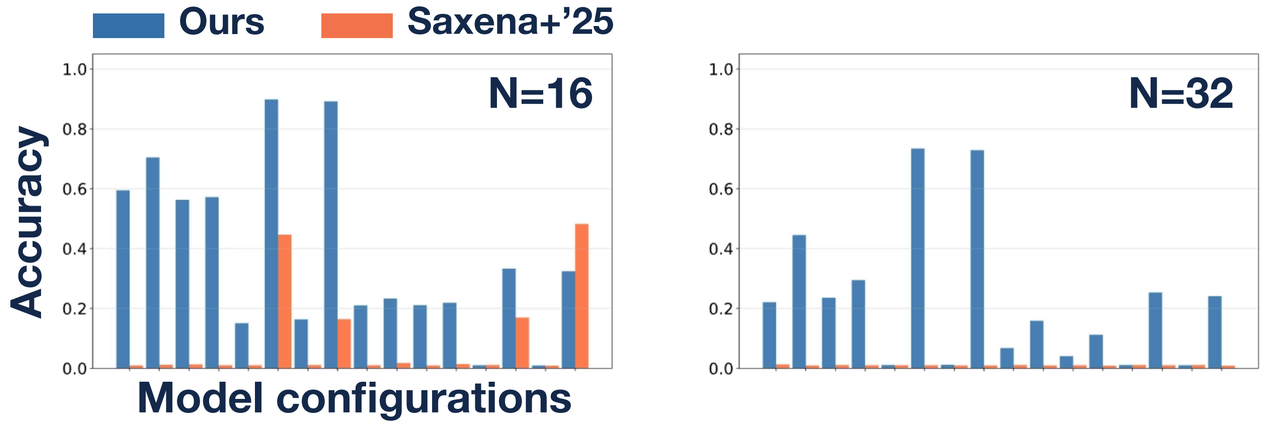
[Saxena+'25]*

Train on instances with many 0s!

- Distirubutional gap between train/test sets.

[Kikuchi+'26.]

Train with additional modulo $Kq,\ (K > 1)$!



- No gap; learning better with 1/10 training samples.

Autoregressive Generation

[Kim & Suzuki'25]

$$[x_1, \ldots, x_N] \quad \mapsto \quad [y_1, \ldots, y_N] \quad \text{with } y_k = \left(\sum_{i=1}^{k} x_i \right) \mod 2$$

- Learning of the parity (by gradient descent) is theoretically hard[Shalev-Shwartz+'17, Hahn+'24].
- Learning with autoregressive generation makes it easy[Kim & Suzuki'25].

Autoregressive Generation *in Order*

Autoregressive generation of an integer product (digit-wise)[Shen+'23]

```
3 1 4  x  1 5 9  =  1 6 2 9 6 6
```

Autoregressive Generation *in Order*

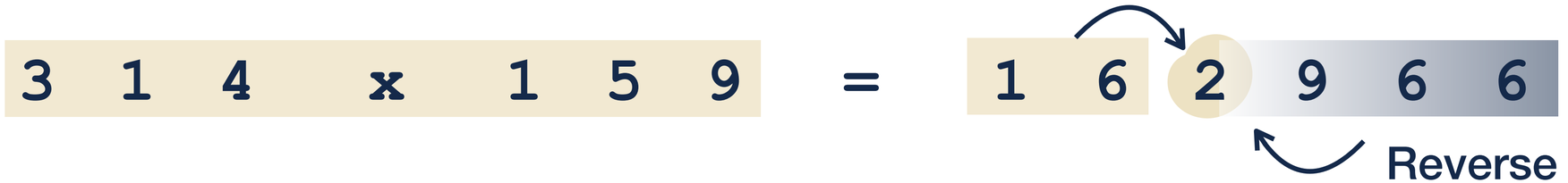
Autoregressive generation of an integer product (digit-wise)[Shen+'23]

# Digits	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	0.99	0.93	0.87	0.86
3	1.00	0.93	0.67	0.32	0.25
4	0.99	0.86	0.34	0.08	0.00
5	0.99	0.82	0.26	0.04	0.01

(a) most-to-least significant digits

Autoregressive Generation *in Order*

Autoregressive generation of an integer product (digit-wise)[Shen+'23]



# Digits	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	0.99	0.93	0.87	0.86
3	1.00	0.93	0.67	0.32	0.25
4	0.99	0.86	0.34	0.08	0.00
5	0.99	0.82	0.26	0.04	0.01

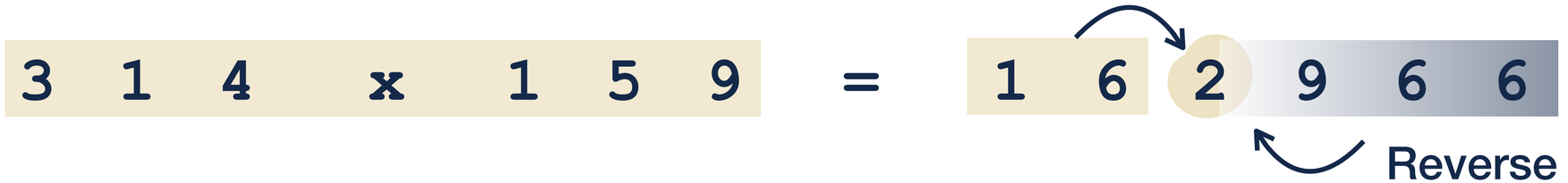
(a) most-to-least significant digits

# Digits	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00

(b) least-to-most significant digits

Autoregressive Generation *in Order*

Autoregressive generation of an integer product (digit-wise)[Shen+'23]



# Digits	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	0.99	0.93	0.87	0.86
3	1.00	0.93	0.67	0.32	0.25
4	0.99	0.86	0.34	0.08	0.00
5	0.99	0.82	0.26	0.04	0.01

(a) most-to-least significant digits

# Digits	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00

(b) least-to-most significant digits

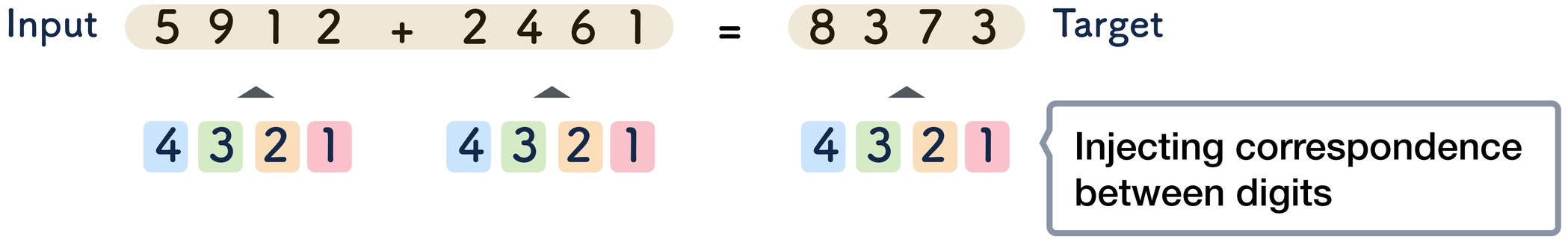
- Learning to predict in a good *ordering* is important.
- [Sato+'25] proposes an exploration method of learning-friendly order.

Correspondence Injection

[McLeish+'24]

Injecting correspondence leads to strong length-generalization.

Integer addition task



Correspondence Injection

[McLeish+'24]

Injecting correspondence leads to strong length-generalization.

Integer addition task

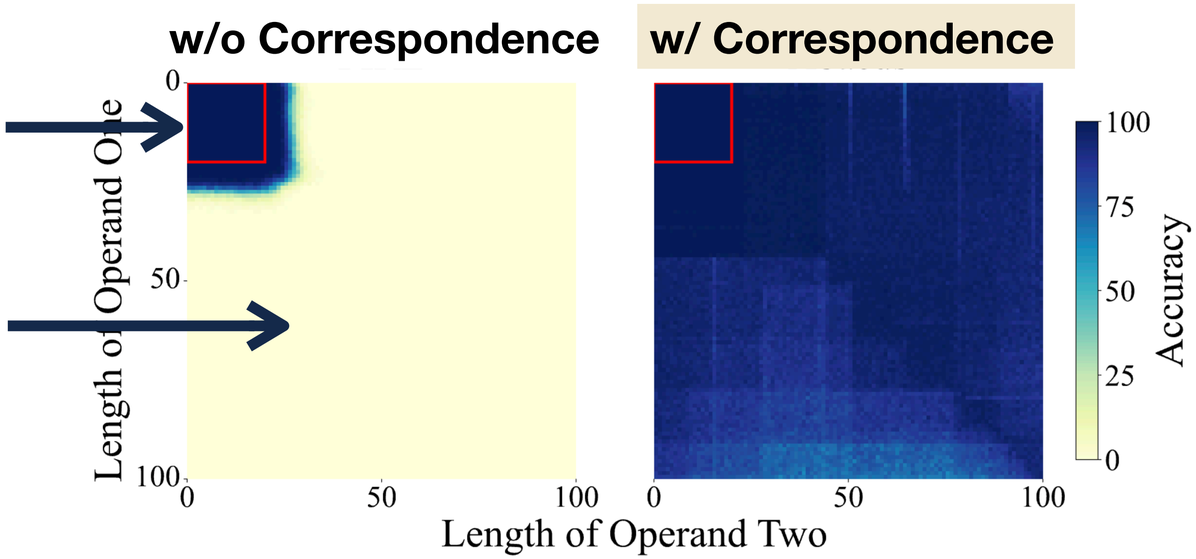
Input 5 9 1 2 + 2 4 6 1 = 8 3 7 3 Target

4 3 2 1 4 3 2 1 4 3 2 1

Injecting correspondence between digits

Training instances up to 20 digits

Test instances up to 100 digits



Takeaway

SALSA

- Under-trained models can still be useful!

Modular arithmetic

- Hard task; several interesting, empirical learning tricks are known.

Auto-regressive generation

- Make hard tasks easier to learn; good task breakdown and orders are important
- Trade-off between prediction accuracy and training cost

Case Study #3

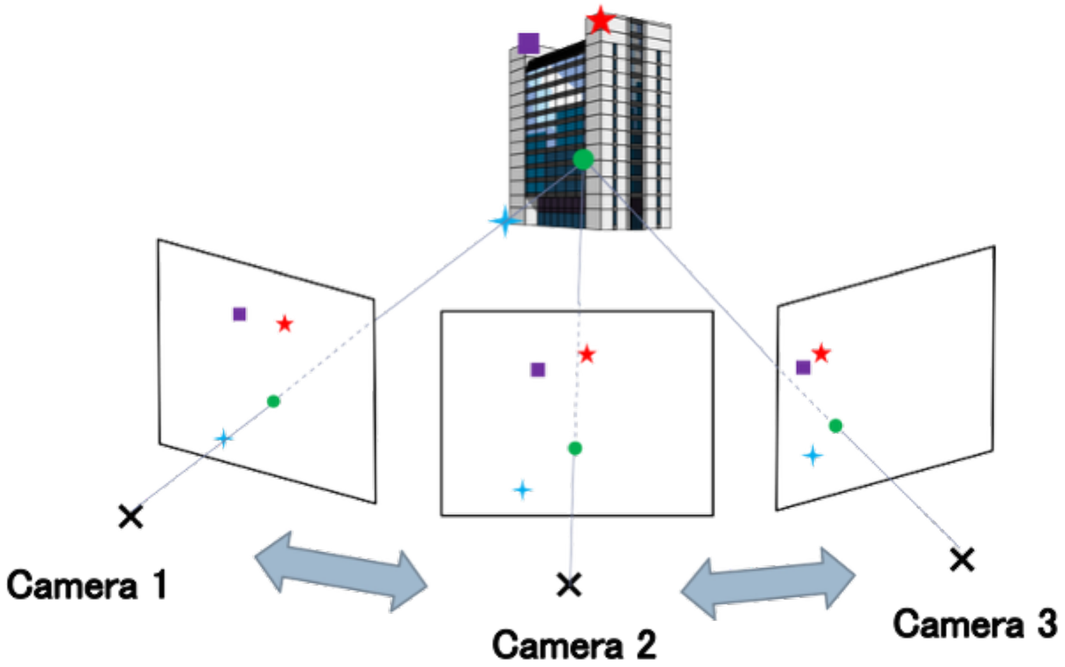
Reduction to Classification, not Generation

Learning to Solve Hard Minimal Problems

Petr Hruby, Timothy Duff, Anton Leykin, Tomas Pajdla

CVPR 2022

Four-points-in-three-views problem



Input: 4 corresponding image points

$$\{(\boldsymbol{v}_{k,1}, \boldsymbol{v}_{k,2}, \boldsymbol{v}_{k,3})\}_{k=1}^{4} \quad \text{(from 3 camera views)}$$

Output: Relative camera positions from Cam. 1

$$(R_2, \boldsymbol{t}_2),\ (R_3, \boldsymbol{t}_3) \quad \text{(rotations and translations)}$$

Task: Solving polynomial systems induced from geometric constraints.

Homotopy continuation

Homotopy continuation: computation of the roots of a polynomial system $F(\boldsymbol{x}) = \mathbf{0}$

$$H(\boldsymbol{x}, t) = (1 - t)G(\boldsymbol{x}) + tF(\boldsymbol{x})$$

A system with known roots

- Tracking the roots of $H(\boldsymbol{x}, t)$ by changing $t = 0 \to 1$.

Homotopy continuation

Homotopy continuation: computation of the roots of a polynomial system $F(\boldsymbol{x}) = \mathbf{0}$

$$H(\boldsymbol{x}, t) = (1 - t)G(\boldsymbol{x}) + tF(\boldsymbol{x})$$

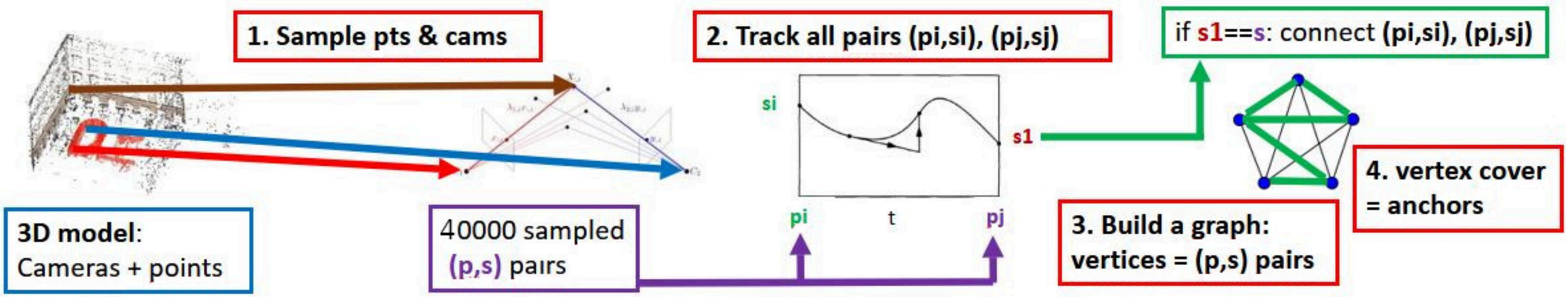
A system with known roots

- Tracking the roots of $H(\boldsymbol{x}, t)$ by changing $t = 0 \to 1$.

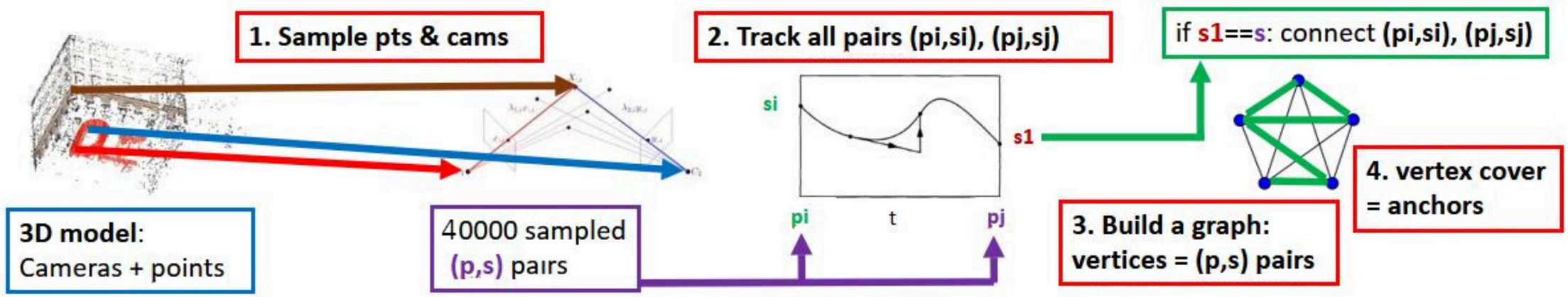
The key is **choosing a good start system** avoiding singularities along the homotopy path.

Approach by [Hruby'22]

Homotopy continuation with a start-system classifier

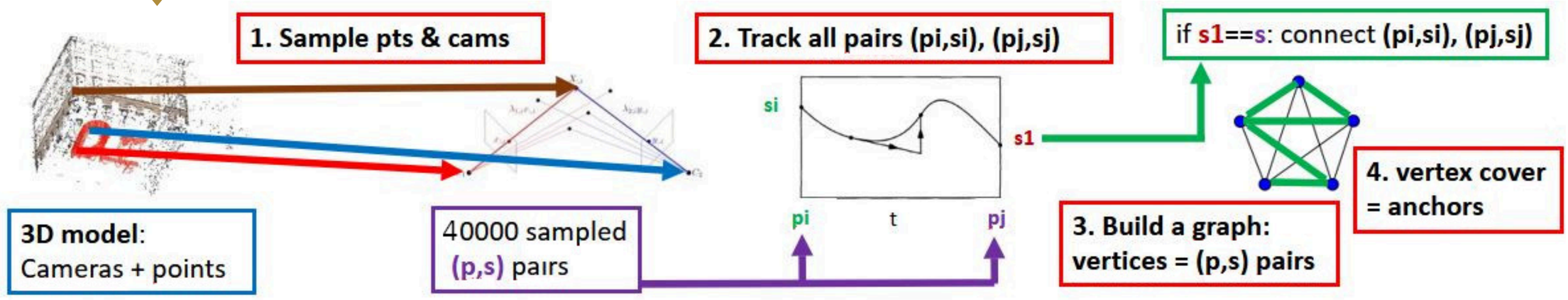

1. Sample pts & cams
2. Track all pairs (pi,si), (pj,sj)
if s1==s: connect (pi,si), (pj,sj)
si
s1
pi
t
pj
3D model:
Cameras + points
40000 sampled
(p,s) pairs
3. Build a graph:
vertices = (p,s) pairs
4. vertex cover
= anchors

- **Step 1:** Correct *anchors*, our start systems, through sampling and selection.



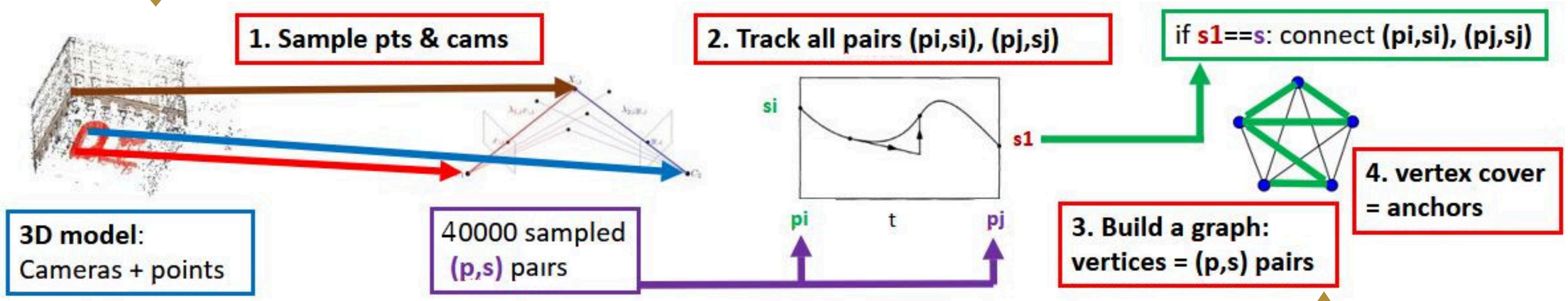
- **Step 1:** Correct *anchors*, our start systems, through sampling and selection.

A 3D dataset offers polynomial systems with solutions



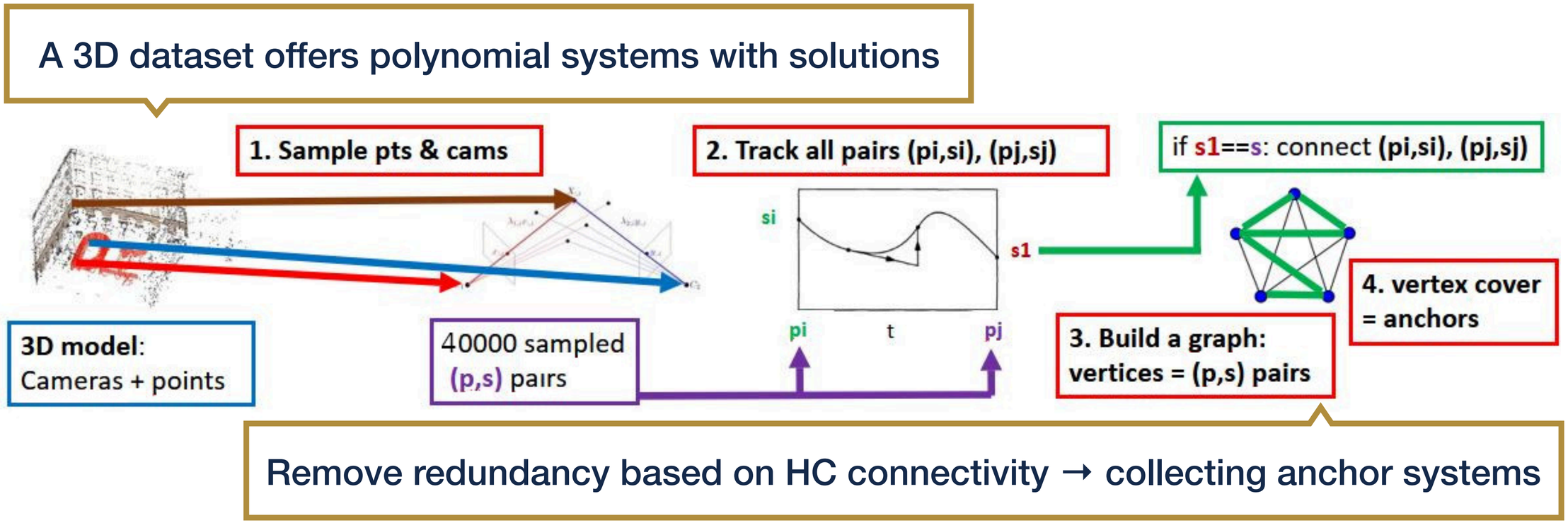
- **Step 1:** Correct *anchors*, our start systems, through sampling and selection.

A 3D dataset offers polynomial systems with solutions



Remove redundancy based on HC connectivity → collecting anchor systems

- **Step 1:** Correct *anchors*, our start systems, through sampling and selection.



- **Step 2:** Sample new systems, and label each system by running HC from each anchor

Takeaway

Reduction to Classification

- Classification is often easier, more stable, and generalizable than generation.

  **but, recall that direct parity computation (binary classification) is hard.*

Task Framing

- Limiting to four-points-in-three-views problem eases the OOD problem.
- This was also the case with SALSA

Closing

Where to start?

Summary and Open Problems

Summary

Case study #1 [Kera-Pelleriti+'25]

Computational Algebra with Attention: Transformer Oracles for Border Basis Algorithms

- Algorithm + learning-based oracle
- Monomial-level embedding

Case study #2 [Wenger+'22]

SALSA: Attacking Lattice Cryptography with Transformers

- Utility of weak learners
- Learning techniques of arithmetic tasks
- Autoregressive generation is powerful given good task decomposition.

Case study #3 [Hruby+'22]

Learning to Solve Hard Minimal Problems

- Classification to initial systems (not generation of them)
- Good task framing

Open problems (SC)

Open problems (SC)

Sampling ideals and Gröbner bases

Learning Gröbner basis computation needs many (F, G) instances

$$\begin{pmatrix} f_1 & \cdots & f_s \end{pmatrix}^\top = A \begin{pmatrix} g_1 & \cdots & g_t \end{pmatrix}^\top$$

Open problems (SC)

Sampling ideals and Gröbner bases

Learning Gröbner basis computation needs many (F, G) instances

$$\begin{pmatrix} f_1 & \cdots & f_s \end{pmatrix}^\top = A \begin{pmatrix} g_1 & \cdots & g_t \end{pmatrix}^\top$$

Gröbner bases? — how to sample them?

- 0-dim cases: ✓ Ideals in shape position[Kera+'24]
  ✓ Ideals of points[Kera-Pelleriti+'25]
- Positive-dimentional cases?

Open problems (SC)

Sampling ideals and Gröbner bases

Learning Gröbner basis computation needs many (F, G) instances

$$\begin{pmatrix} f_1 & \cdots & f_s \end{pmatrix}^\top = A \begin{pmatrix} g_1 & \cdots & g_t \end{pmatrix}^\top$$

What A keeps the ideal unchanged, $\langle F \rangle = \langle G \rangle$?

Gröbner bases? — how to sample them?

- "Generic $A \in K[X]^{s \times t}$ from Bruhat decomp. works" [Kambe+'25]
- "Generic $A \in K[X]^{s \times t}$ with $s > n$ works" [Kera-Pelleriti+'25]

- 0-dim cases: ✓ Ideals in shape position[Kera+'24]
  ✓ Ideals of points[Kera-Pelleriti+'25]
- Positive-dimentional cases?

Open problems (SC)

Sampling ideals and Gröbner bases

Learning Gröbner basis computation needs many (F, G) instances

$$\begin{pmatrix} f_1 & \cdots & f_s \end{pmatrix}^\top = A \begin{pmatrix} g_1 & \cdots & g_t \end{pmatrix}^\top$$

What A keeps the ideal unchanged, $\langle F \rangle = \langle G \rangle$?

Gröbner bases? — how to sample them?

- "Generic $A \in K[X]^{s \times t}$ from Bruhat decomp. works" [Kambe+'25]
- "Generic $A \in K[X]^{s \times t}$ with $s > n$ works" [Kera-Pelleriti+'25]

- 0-dim cases: ✓ Ideals in shape position[Kera+'24]
  ✓ Ideals of points[Kera-Pelleriti+'25]
- Positive-dimentional cases?

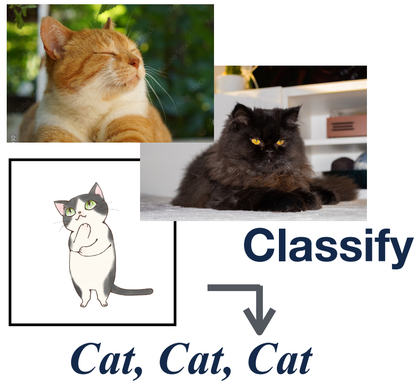
Extending the tricks in learning arithmtic to symbolic tasks

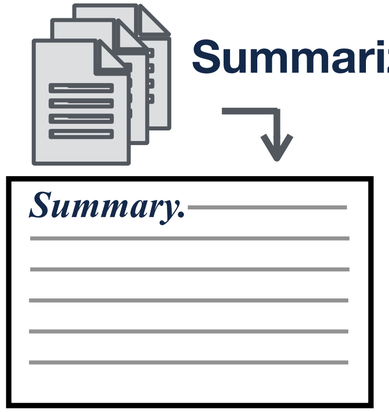
- Good ordering for autoregressively generating Gröbner bases?
- Informative correspondence between tokens to accelarate learning?

Open problems (ML)

Open problems (ML)

Learning functions with input-sensitivity — Unified learning methodology and theory?





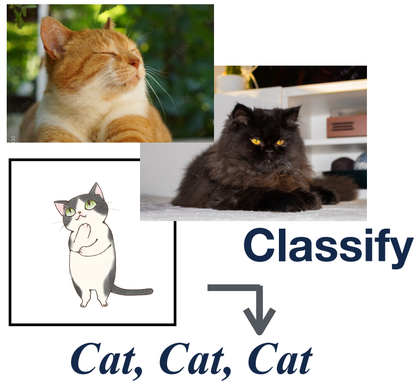
vs.

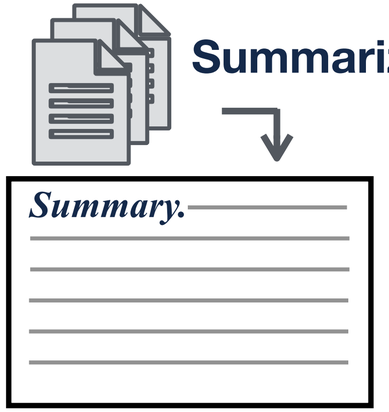
$$\begin{cases} f_1 = & x^3 - 3x^2 - y + 1 \\ f_2 = & -x^2 + y^2 - 1 \end{cases}$$

$$\begin{cases} g_1 = & x^2 - y^2 + 1 \\ g_2 = & xy - x - y^4 + 11y^2 + 3y - 13 \\ g_3 = & y^5 + y^4 - 11y^3 - 17y^2 + 9y + 17 \end{cases}$$

Open problems (ML)

Learning functions with input-sensitivity — Unified learning methodology and theory?





vs.

$$\begin{cases} f_1 = & x^3 - 3x^2 - y + 1 \\ f_2 = & -x^2 + y^2 - 1 \end{cases}$$

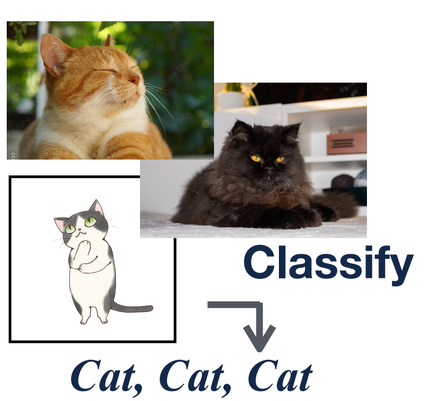
$$\begin{cases} g_1 = & x^2 - y^2 + 1 \\ g_2 = & xy - x - y^4 + 11y^2 + 3y - 13 \\ g_3 = & y^5 + y^4 - 11y^3 - 17y^2 + 9y + 17 \end{cases}$$

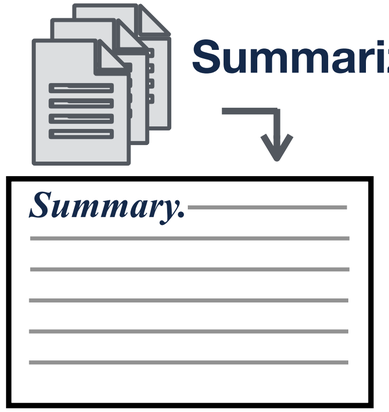
End-to-end approach

- End-to-end learning is fishy; no certification of the output, weak OOD generalization
- Yet, that is an advantage of Transformers, and speed-up in literature often drastic.

Open problems (ML)

Learning functions with input-sensitivity — Unified learning methodology and theory?





vs.

$$\begin{cases} f_1 = & x^3 - 3x^2 - y + 1 \\ f_2 = & -x^2 + y^2 - 1 \end{cases}$$

$$\begin{cases} g_1 = & x^2 - y^2 + 1 \\ g_2 = & xy - x - y^4 + 11y^2 + 3y - 13 \\ g_3 = & y^5 + y^4 - 11y^3 - 17y^2 + 9y + 17 \end{cases}$$

End-to-end approach

- End-to-end learning is fishy; no certification of the output, weak OOD generalization
- Yet, that is an advantage of Transformers, and speed-up in literature often drastic.

Algorithms with Predictions [Mitzenmacher and Vassilvitskii'20]

- AwP handles uncertainty of ML models into algorithm's output quality (errors).
- However, most symbolic algorithms doesn't come with 1D smooth error metrics.

CALT: Computer Algebra with Transformers

[Kera-Arakawa-Chamberlan-Bonaccorsi-Sato'25]

```
> pip install calt-x
```

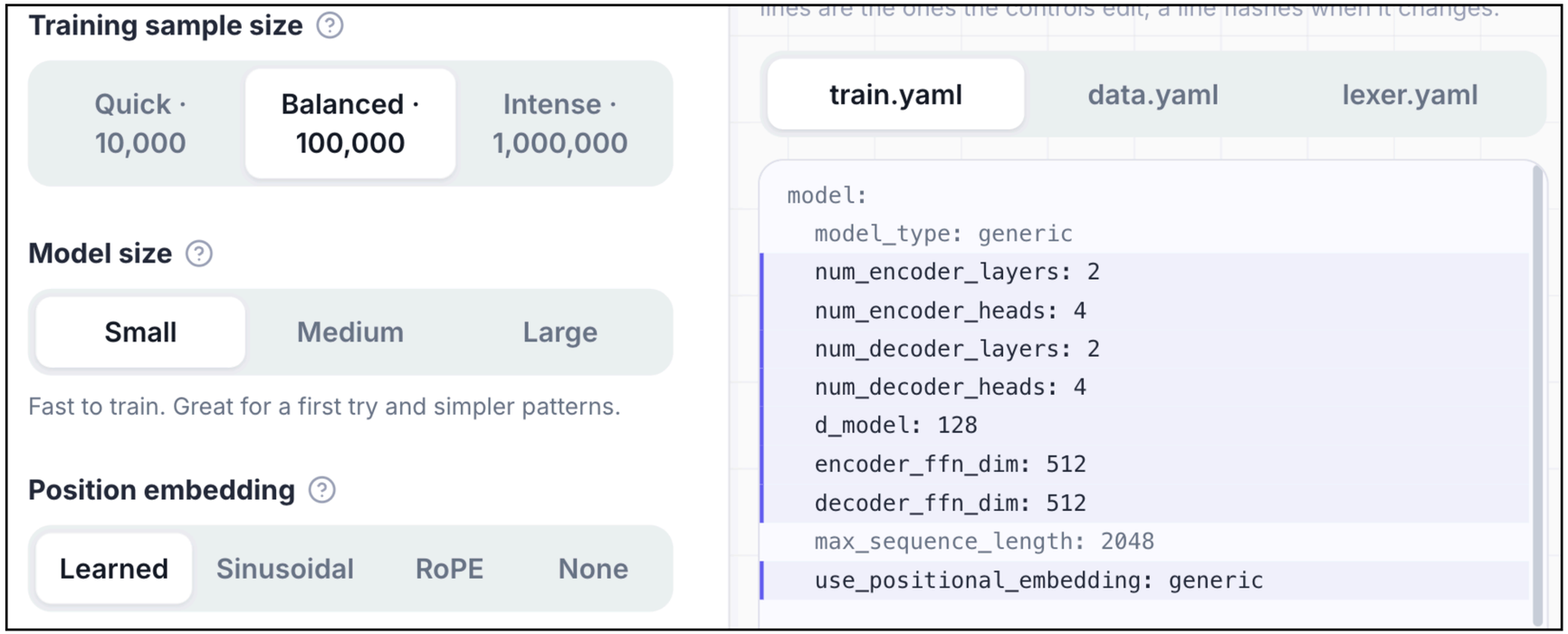
Open in Colab

Training your Transformers for arithmetic and symbolic tasks with minimal coding.

- **conda-forge** for installation
- **Codebase** for experiment template and examples
- **Code Builder** for your easy entry.

CALT Code Builder

1. Define or choose your task



2. Set up config files

References

Further reading

Case study 0 — Learning Gröbner basis computation

1. Kera, H., Ishihara, Y., Kambe, Y., Vaccon, T., and Yokoyama, K. **Learning to Compute Gröbner Bases.** *NeurIPS*, 2024.
   End-to-end Gröbner learning; poses backward Gröbner basis generation and conversion problems.
2. Kambe, Y., Maeda, Y., and Vaccon, T. **Geometric Generality of Transformer-Based Gröbner Basis Computation.** *Contemporary Mathematics* 835 (arXiv:2504.12465), 2025.
   Proves the backward-generated datasets are sufficiently general—a geometric foundation for learned GB.
3. Malhou, M., Perret, L., and Lauter, K. **HATSolver: Learning Gröbner Bases with Hierarchical Attention Transformers.** *arXiv:2512.14722*, 2025.
   Scales to 13 variables, degree 11; true OOD generalization beyond backward-generated data left open.
4. Lample, G., and Charton, F. **Deep Learning for Symbolic Mathematics.** *ICLR*, 2020.
   Seq2seq integration and ODE solving with FWD/BWD/IBP data and cheap differentiation checks.
5. Ishihara, Y., and Yokoyama, K. **Efficient Dataset Generation for Bases of Zero-Dimensional Ideals.** *SCML*, 2026.
   Fast backward generation of (system, basis) pairs for Gröbner and border basis learning.
6. Kambe, Y. **Zariski-Dense Dataset Generation for Learning to Compute Gröbner Bases.** *SCML*, 2026.
   Training data that is Zariski-dense aims to underwrite OOD generalization for learned GB.

Case study 1 — Border bases and learned oracles

7. Kera, H., Pelleriti, N., Ishihara, Y., Zimmer, M., and Pokutta, S. **Computational Algebra with Attention: Transformer Oracles for Border Basis Algorithms.** *NeurIPS*, 2025.
   Oracle prunes BBA expansions with fallback; border-basis backward sampling and monomial embeddings.

8. Kehrein, A., Kreuzer, M., and Robbiano, L. **An Algebraist's View on Border Bases.** In *Solving Polynomial Equations: Foundations, Algorithms, and Applications*, Springer, 2005.
   Border bases: definitions, order ideals, and their numerical stability advantages.

9. Kehrein, A., and Kreuzer, M. **Computing Border Bases.** *J. Pure and Applied Algebra* 205(2), 2006.
   The border basis algorithm (BBA): expand, reduce, and check termination on order ideals.

10. Pelleriti, N., Spiegel, C., Liu, S., Martínez-Rubio, D., Zimmer, M., and Pokutta, S. **Neural Sum-of-Squares: Certifying the Nonnegativity of Polynomials with Transformers.** *arXiv:2510.13444*, 2025.
    Predict SOS decompositions, then verify and round to exact positivity certificates.

11. Peifer, D., Stillman, M., and Halpern-Leistner, D. **Learning Selection Strategies in Buchberger's Algorithm.** *ICML*, 2020.
    RL learns S-pair selection, cutting polynomial additions by 20–40% on ideal benchmarks.

Case study 2 — SALSA and arithmetic learning (1/2)

12. Wenger, E., Chen, M., Charton, F., and Lauter, K. **SALSA: Attacking Lattice Cryptography with Transformers.** *NeurIPS*, 2022.
    First ML attack on LWE: a half-trained Transformer feeds statistical secret-recovery procedures.

13. Li, C. Y., Sotáková, J., Wenger, E., Malhou, M., Garcelon, E., Charton, F., and Lauter, K. **SALSA PICANTE: A Machine Learning Attack on LWE with Binary Secrets.** *ACM CCS*, 2023.
    Scales SALSA to dimension n ≈ 350 with real-world-ish LWE parameters.

14. Li, C. Y., Wenger, E., Allen-Zhu, Z., Charton, F., and Lauter, K. **SALSA VERDE: A Machine Learning Attack on LWE with Sparse Small Secrets.** *NeurIPS*, 2023.
    Extends SALSA to sparser, smaller secrets via improved preprocessing and training.

15. Stevens, S., Wenger, E., Li, C. Y., Nolte, N., Saxena, E., Charton, F., and Lauter, K. **SALSA FRESCA: Angular Embeddings and Pre-training for ML Attacks on LWE.** *TMLR*, 2025.
    Angular embeddings and pre-training further scale lattice attacks on denser secrets.

16. Shalev-Shwartz, S., Shamir, O., and Shammah, S. **Failures of Gradient-Based Deep Learning.** *ICML*, 2017.
    Parity-like problems provably resist gradient-based learning—the hardness behind modular arithmetic.

17. Hahn, M., and Rofin, M. **Why are Sensitive Functions Hard for Transformers?** *ACL*, 2024.
    High-sensitivity functions like PARITY sit in sharp, isolated minima of the Transformer loss landscape.

18. Kim, J., and Suzuki, T. **Transformers Provably Solve Parity Efficiently with Chain of Thought.** *ICLR*, 2025.
    Recursive prediction of subparities makes parity learnable—theory behind autoregressive task breakdown.

Case study 2 — SALSA and arithmetic learning (2/2)

19. Saxena, E., Alfarano, A., Wenger, E., and Lauter, K. E. **Making Hard Problems Easier with Custom Data Distributions and Loss Regularization: A Case Study in Modular Arithmetic.** *ICML*, 2025.
    Sparse training distributions and a regularized loss scale modular addition to N = 128, q ≤ 974,269.

20. Kikuchi, H., Masuya, R., Kawamoto, K., and Kera, H. **Learning Large-Scale Modular Addition with an Auxiliary Modulus.** *Preprint*, 2026.
    Auxiliary modulus eases learning without the covariate shift of sparse-input training.

21. Sato, Y., Kawamoto, K., and Kera, H. **Discovering Learning-Friendly Generation Orders for Sequential Computation.** *Preprint*, 2026.
    Output generation order (left-to-right vs. right-to-left) strongly affects learnability.

22. Shen, R., Bubeck, S., Eldan, R., Lee, Y. T., Li, Y., and Zhang, Y. **Positional Description Matters for Transformers Arithmetic.** *arXiv:2311.14737*, 2023.
    How positions are described—not just encoded—largely determines arithmetic success.

23. McLeish, S., Bansal, A., Stein, A., Jain, N., Kirchenbauer, J., Bartoldson, B. R., Kailkhura, B., Bhatele, A., Geiping, J., Schwarzschild, A., and Goldstein, T. **Transformers Can Do Arithmetic with the Right Embeddings.** *NeurIPS*, 2024.
    Abacus embeddings encode digit position; 20-digit training generalizes to 120-digit addition.

24. Cho, H., Cha, J., Awasthi, P., Bhojanapalli, S., Gupta, A., and Yun, C. **Position Coupling: Improving Length Generalization of Arithmetic Transformers.** *NeurIPS*, 2024.
    Task structure is baked into position IDs to achieve length generalization on addition.

Case study 3 — Classification, and predict-then-verify

25. Hruby, P., Duff, T., Leykin, A., and Pajdla, T. **Learning to Solve Hard Minimal Problems.** *CVPR*, 2022.
Classify to a precomputed anchor set instead of solving hundreds of spurious solutions.

26. Bernal, E. A., Hauenstein, J. D., Mehta, D., Regan, M. H., and Tang, T. **Machine Learning the Real Discriminant Locus.** *J. Symbolic Computation* 115, 2023.
Learn the real discriminant locus of parametric systems via numerical algebraic geometry labels.

27. Barket, R., Shafiq, U., England, M., and Gerhard, J. **Transformers to Predict the Applicability of Symbolic Integration Routines.** *arXiv:2410.23948*, 2024.
ML replaces Maple integration guards; CAS still guarantees correctness of outputs.

28. John, R., Barket, R., and England, M. **Replacing Heuristic Rule Ordering in Symbolic Integration with Learned Policies.** *SCML*, 2026.
Learned policies replace hand-tuned rule order in symbolic integration solvers.

29. Alfarano, A., Charton, F., and Hayat, A. **Global Lyapunov Functions: A Long-Standing Open Problem in Mathematics, with Symbolic Transformers.** *NeurIPS*, 2024.
Sample Lyapunov functions first, then build dynamics—backward generation beyond SOS reach.

Framework, history & tools

30. Giovini, A., Mora, T., Niesi, G., Robbiano, L., and Traverso, C. **"One sugar cube, please" or Selection Strategies in the Buchberger Algorithm.** *Proc. ISSAC*, 1991.
Sugar strategy for S-pair ordering—the heuristic culture behind practical computer algebra.

31. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. **Attention Is All You Need.** *NeurIPS*, 2017.
The Transformer: the sequence-to-sequence architecture used throughout this talk.

32. Huang, Z., England, M., Wilson, D., Davenport, J. H., Paulson, L. C., and Bridge, J. **Applying Machine Learning to the Problem of Choosing a Heuristic to Select the Variable Ordering for Cylindrical Algebraic Decomposition.** *CICM*, 2014.
The first application of ML inside a computer algebra pipeline: an SVM picks CAD ordering heuristics.

33. Huang, Z., England, M., Wilson, D. J., Bridge, J., Davenport, J. H., and Paulson, L. C. **Using Machine Learning to Improve Cylindrical Algebraic Decomposition.** *Math. Comput. Sci.* 13, 2019.
Journal version: classical ML predicts CAD variable order inside a CAS pipeline.

34. Jing, R.-J., Zhao, Y., and Chen, C. **Breaking the Data Barrier in Learning Symbolic Computation: A Case Study on Variable Ordering for CAD.** *arXiv:2601.13731*, 2026.
Addresses data scarcity for learning CAD heuristics—presented at this conference.

35. Mitzenmacher, M., and Vassilvitskii, S. **Algorithms with Predictions.** In *Beyond the Worst-Case Analysis of Algorithms*, Cambridge University Press, 2020.
ML predictions speed algorithms while preserving worst-case guarantees (consistency vs. robustness).

36. Kera, H., Arakawa, S., Chamberlan, Q., Bonaccorsi, M., and Sato, Y. **CALT: A Library for Computer Algebra with Transformer.** *arXiv:2506.08600*, 2025.
SymPy-based library: one instance generator → dataset, training, and evaluation for SC researchers.