TabularMath: Evaluating Computational Extrapolation in Tabular Learning via Program-Verified Synthesis

Zerui Cheng [1 * †] **Jiashuo Liu** [2 *] **Jianzhu Yao** [1] **Pramod Viswanath** [1] **Ge Zhang** [2] **Wenhao Huang** [2]

Abstract

Standard tabular benchmarks mainly focus on the evaluation of a model's capability to interpolate values inside a data manifold, where models good at performing local statistical smoothing are rewarded. However, there exists a very large category of high-value tabular data, including financial modeling and physical simulations, which are generated based upon deterministic computational processes, as opposed to stochastic and noisy relationships. Therefore, we investigate if tabular models can provide an extension from statistical interpolation to *computational extrapolation*.

We propose TABULARMATH, a diagnostic benchmark of 114 deterministic problems (233,472 rows) generated from verified programs based on GSM8K and AIME. We evaluate 9 tabular architectures and in-context learning (ICL) with `GPT-OSS-120B`. On standard regression metrics, TabPFN v2.5 performs remarkably well, achieving $R^2 = 0.998$ in-distribution and maintaining positive R^2 even under distribution shift—unique among the tabular models we tested. When we measure *rounded consistency* (exact integer match), a different picture emerges: TabPFN v2.5 drops below 10% on out-of-distribution data, while ICL maintains around 40%. This gap between R^2 and exact-match accuracy suggests that tabular models learn smooth function approximations but struggle to recover precise computational outputs under extrapolation. The two paradigms appear complementary: TabPFN scales efficiently with data; ICL achieves exact computation from few examples. We release all code and data at `https://github.com/Marco-Cheng/TabularMath` to support further investigation.

* Equal contribution. † Work done during internship at ByteDance. [1]Princeton University [2]ByteDance. Correspondence to: Zerui Cheng <zerui.cheng@princeton.edu>, Jiashuo Liu <liujiashuo77@gmail.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

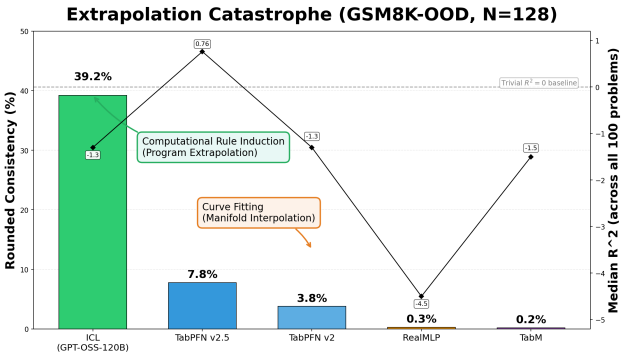


Figure 1. The R^2–consistency gap. On TABULARMATH-GSM8k with out-of-distribution test data, ICL achieves ∼39% exact-match accuracy while tabular models fall below 10%. TabPFN v2.5 nonetheless achieves strong R^2 (0.75 under OOD), indicating that it captures the functional trend even when exact predictions fail. The two approaches exhibit complementary strengths: ICL recovers precise computations from few examples; TabPFN approximates functions efficiently and improves with more data.

Can modern tabular models learn to *compute*?

Given a table of input–output pairs generated by a deterministic arithmetic rule, can a tabular model recover the rule and make exact predictions on inputs outside the training range?

Our experiments suggest a nuanced answer. Tabular models—particularly TabPFN v2.5—achieve excellent regression fits, but exact-match accuracy under extrapolation remains low. This gap between R^2 and precise computation has implications for applications where approximate answers are insufficient.

1.1. Motivation for a New Benchmark

Existing tabular benchmarks primarily test statistical interpolation: whether models can smooth predictions across a data distribution and generalize to new points from the same

distribution. This focus has favored tree ensembles (Grinsztajn et al., 2022) and prior-fitted networks like TabPFN (Hollmann et al., 2023), which are effective at pattern recognition in noisy real-world data.

However, many high-value tabular tasks involve deterministic computation rather than statistical relationships—pricing formulas, physics simulations, business logic. For these tasks, the model must recover the underlying computation, not merely approximate the distribution. Whether tabular models can do this has not been systematically tested.

A Motivating Example. Consider: "Item cost = \$125. Discount = 20%. Tax on discounted price = 8%. Final price?" The answer is \$108 exactly. We can generate thousands of variants by changing the parameters. When train and test data are i.i.d., models achieve near-perfect R^2. But when test outputs exceed the training range (extrapolation), exact-match accuracy drops substantially—even when R^2 remains reasonable. This gap between regression fit and exact computation is systematic across our tested problems.

In this work, we propose a generic pipeline to *synthesizing verified tabular benchmarks from computational tasks* (see Appendix A). Starting from a formal mathematical specification of a task, we enable automatic generation of very large tabular datasets by means of coupled generator-verifier programs that provide a *provable correct label* for every row of the dataset. This construction yields tabular datasets with *zero irreducible error* and controlled distribution shifts, enabling precise evaluation of whether models learn computational structure rather than merely fitting training distributions.

1.2. Transforming Math Problems to Tabular Computation

To transform this property into something measurable and systematic, we introduce TABULARMATH, which transforms math word problems into tabular regression problems with *known, correct answers*. Each problem defines a deterministic computation: the numeric quantities in the text become the input columns, and the solution is a fixed, deterministic function of these inputs.

More concretely, for each seed problem we have (i) extract the numeric quantities from the text to form a template, (ii) synthesize a *generator* that samples valid numeric quantities to create a row, and (iii) synthesize a *verifier* that exactly computes the math logic of the problem. The verifier is used to compute the label of each row and is only accepted when it produces the same answer as the original problem.

This generator-verifier construction guarantees that TABULARMATH will produce tabular datasets with *no irreducible errors*. Therefore, any generalization failure reflects an inability to infer or execute the underlying computation, rather than noise, annotation error, or ambiguity.

Evaluation Framework. This construction enables a clear distinction between interpolation and extrapolation. We perform evaluations on our model(s) in both of the following ways: (i) standard i.i.d. splits, and (ii) out-of-distribution (OOD) splits, where the model is trained on rows with the smallest 80% of target values and evaluated on the largest 20%.

Since the output is deterministic, the OOD performance directly measures *algorithmic extrapolation*: the ability of a model to apply what it has learned outside of the ranges of the training data.

We compare 10 model types including tree ensembles, a variety of neural tabular architectures, and in-context learning (ICL) using `GPT-OSS-120B`. ICL plays a crucial experimental role here since it is an *existence proof* that the underlying computational logic can be learned from samples of the data, serving as the baseline.

Main Results. Our experiments reveal a stark gap that standard evaluations miss:

1. **A $5\times$ extrapolation gap.** Under OOD evaluation, ICL maintains $\sim$40% exact-match accuracy while TabPFN v2.5 drops to 8% and tree ensembles fall below 1%. This gap is invisible on i.i.d. data.
2. **Remarkable sample efficiency of ICL.** With just 32 examples, ICL achieves 55% accuracy; TabPFN needs roughly 500 examples to match this level.
3. **Complementary strengths.** Given sufficient in-distribution data (2,048 rows), TabPFN v2.5 surpasses ICL at 62% accuracy. Tabular models excel at interpolation; ICL excels at rule induction.

Why This Matters.

Existing tabular benchmarks test interpolation, not rule recovery. As a result, a critical failure mode—catastrophic degradation under extrapolation in deterministic domains—remains hidden from standard evaluations.

TABULARMATH makes this blind spot visible by linking tabular learning to verified computation. Our findings suggest that while current tabular architectures excel on traditional benchmarks, they lack the inductive bias for systematic computation. In domains where correctness under extrapolation is non-negotiable, this limitation has practical consequences.

1.3. Contributions

This paper makes the following contributions:

1. **A synthesis framework for verified tabular benchmarks.** We introduce a methodology that transforms

computational tasks into tabular datasets through coupled generator-verifier programs. Every generated row has a provably correct label, and distribution shifts can be precisely controlled.

2. **The TABULARMATH benchmark.** We apply this framework to 114 mathematical problems from GSM8K and AIME, producing 233,472 rows with guaranteed correctness. This separates algorithmic reasoning from statistical noise.

3. **Systematic evaluation of interpolation versus extrapolation.** Using deterministic targets and output-based OOD splits, we document how leading tabular models fail under extrapolation. We also demonstrate that ICL can recover the underlying computations from examples alone, serving as an existence proof.

4. **Practical guidance.** We analyze why different model families exhibit different failure modes, characterize the complementary strengths of tabular models and ICL, and discuss implications for practitioners choosing between approaches.

Scope. TABULARMATH is a diagnostic tool for probing specific limitations of tabular models, not a general-purpose evaluation framework. Standard tabular models are conservative interpolators by design—a desirable property in noisy real-world settings. Our goal is to isolate when this inductive bias becomes a liability: namely, in deterministic applications requiring precise extrapolation. We hope this work motivates architectures that combine statistical robustness with computational precision.

2. Methodology: Program-Verified Tabular Synthesis

TABULARMATH is designed to isolate *computational generalization* in tabular learning. Each problem induces a deterministic mapping from structured numeric inputs to an exact output, and we construct datasets by sampling inputs and computing labels using an executable verifier program. This removes label noise and makes distribution shifts explicit.

2.1. Latent program view

We adopt a simple perspective: behind each seed math problem is a latent program that deterministically maps a set of numeric quantities to an answer.

Hypothesis 1 (Latent Program)**.** *Each* TABULARMATH *task corresponds to a deterministic verifier* $\mathcal{V} : \mathcal{X} \to \mathcal{Y}$ *such that* $y = \mathcal{V}(\mathbf{x})$ *for all valid inputs. The verifier is an executable (Turing-complete) program implementing the task's computational logic.*

The hypothesis is true by construction: we only include a task once we have validated a verifier that reproduces the original seed answer.

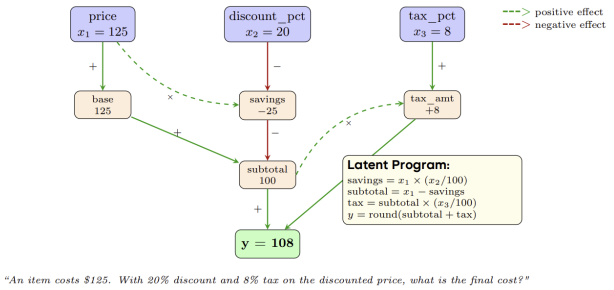


Figure 2. **From computation graphs to more general latent programs.** Many synthetic tabular benchmarks are generated from computation graphs. In TABULARMATH, each dataset is generated by a latent operator (right): an executable Turing-complete Python program that encode the underlying math task. Latent programs can characterize both smooth and discrete relationships and is a generalization on computation graphs.

2.2. Verified synthesis pipeline

To scale beyond hand-written generators, we use an LLM as a *compiler*: it converts a seed word problem into a coupled generator–verifier pair. We then validate the verifier on the base instance from the seed problem and reject any pair that fails the check. Finally, we sample inputs using the generator and compute exact labels using the verifier.

Example: a seed problem becomes a data engine

Seed (simplified). "Item cost = \$125. Discount = 20%. Tax on discounted price = 8%. Final price?"
Template. Replace the constants with slots (a, b, c) corresponding to cost, discount, and tax.
Verifier. $y = \mathrm{round}\big((a \cdot (1 - b/100)) \cdot (1 + c/100)\big)$.
Sample rows. $(a, b, c) \mapsto y$: $(125, 20, 8) \mapsto 108$, $(200, 15, 10) \mapsto 187$, $(90, 25, 5) \mapsto 71$.

2.3. Design goals: controllable extrapolation with zero label noise

TABULARMATH is intended to isolate *computational* generalization in a setting where labels are unambiguous and can be generated at scale. Our construction therefore follows three design principles.

Zero label noise via executable verification. Every row is labeled by running a verifier program $\mathcal{V}$ derived from the seed problem. This lets us treat deviations in evaluation as genuine model failures (rather than annotation noise), and it enables exact-match metrics that are meaningful for deterministic tasks.

Controllable sample complexity via row caps. For

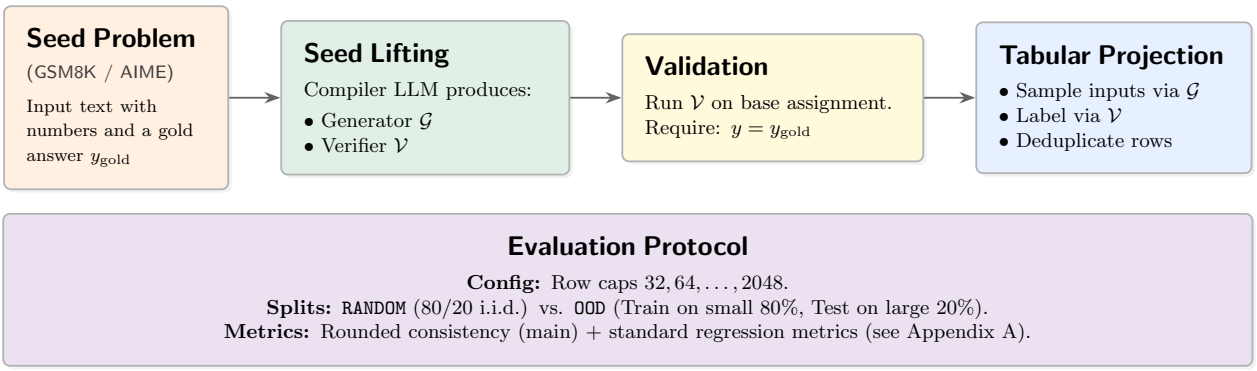


Figure 3. End-to-end construction and evaluation pipeline for TABULARMATH.

each seed, we generate tables at multiple *row caps* $n \in \{32, 64, \ldots, 2048\}$. This produces learning curves in a regime where the underlying function is fixed but the number of demonstrations varies, allowing us to ask whether a method *approaches* correct computation as n increases.

Output extrapolation via sorted-target splits. To probe extrapolation beyond the observed numeric range, we define an OOD split by sorting rows by the target y and training on the smallest 80% while testing on the largest 20%. Unlike random splits, this construction explicitly withholds a region of the output space, turning otherwise easy interpolation into a controlled extrapolation test. Importantly, because the targets are generated by programs, the OOD split does not introduce label noise; it only changes which parts of the function are observed.

Benchmark formulation (notation)

A seed problem induces a deterministic mapping $f : \mathcal{X} \to \mathbb{R}$ (implemented by $\mathcal{V}$) and a sampling distribution over inputs (implemented by $\mathcal{G}$). For a row cap n, we sample $x_i \sim \mathcal{G}$ and set $y_i = f(x_i)$, forming a table $D_n = \{(x_i, y_i)\}_{i=1}^{n}$. We evaluate predictors on RANDOM (i.i.d. 80/20) and OOD (sorted-target) splits, and report *rounded consistency* (Eq. 1) as an exactness metric.

We use generic, model-agnostic feature transforms (e.g., log and modular features) to provide tabular learners with common arithmetic cues; details are deferred to Appendix B.

2.4. Feature engineering for arithmetic cues

One might wonder whether the poor extrapolation performance of tabular models simply reflects a lack of appropriate input features. To address this, we provide all models with a fixed set of generic arithmetic transforms computed from each primary slot. These include: (i) log-transformed features $\log(1 + |x|)$ to capture multiplicative relationships, (ii) sign and parity indicators to expose discrete structure, (iii) modular features $x \bmod k$ for small values of k that might reveal periodic patterns, and (iv) standard encodings for categorical slots where present.

This feature engineering is deliberately model-agnostic—we apply the same transforms to all methods rather than hand-tuning per problem or per model. The intent is to give tabular learners access to common arithmetic cues without introducing unfair advantages. Despite these augmentations, the extrapolation gap persists, suggesting that the limitation lies in the model architectures and training objectives rather than in feature availability.

2.5. Validation and rejection criteria

The synthesis pipeline is designed to be conservative: we only accept a seed if the generated verifier is correct on the base instance and the resulting data looks well-formed. Concretely, a synthesized generator–verifier pair must satisfy:

- **Base-instance agreement:** the verifier reproduces the published seed answer exactly;
- **Determinism:** repeated runs on the same input produce the same output;
- **Type and range checks:** generated slots respect simple constraints (e.g., non-negativity for counts) and avoid invalid states such as division by zero;
- **Executable labeling:** the verifier runs to completion under a strict timeout and returns a numeric label.

These checks ensure that TABULARMATH has no label noise by construction and that any errors in downstream models reflect learning failures rather than annotation artifacts.

2.6. Benchmark curation

We instantiate the pipeline on two sets of seed problems.

Statistic	*GSM8K*	*AIME*
Number of problems	100	14
Rows per problem	2,048	2,048
Total rows	204,800	28,672
Avg. features/problem	45.3	40.6
Min features	25	18
Max features	98	88
Combined total	*233,472 rows*	

Table 1. Dataset statistics for TABULARMATH.

GSM8K (Grade School Math). 100 word problems requiring multi-step arithmetic (addition, subtraction, multiplication, division, percentages, rates).

AIME 2024 (Competition Math). 14 problems covering number theory, algebra, combinatorics, and geometry, often involving modular arithmetic or counting.

For GSM8K, we sample 100 problems uniformly at random. The synthesis tool produces a valid generator–verifier pair for each sampled seed within 20 attempts, and we include all successfully validated problems. For each problem, we generate 2,048 rows after deduplication.

3. Experimental setup

3.1. Evaluation protocol

We evaluate all models under two split regimes to distinguish interpolation from extrapolation.

RANDOM Split. A standard 80/20 random split where training and test data are drawn i.i.d. from the same distribution. The 80% forms the *context* (training) set; the remaining 20% forms the *query* (test) set.

OOD Split (Out-of-Distribution). Rows are sorted by their target values y. The bottom 80% (smallest outputs) become the context set; the top 20% (largest outputs) become the query set. This forces the model to predict outputs that exceed any value observed during training.

Extrapolation scope. It is worth distinguishing *input extrapolation*—where test inputs lie outside the training range—from *output extrapolation*—where test outputs exceed observed training outputs. Our OOD split targets output extrapolation specifically. Although input and output extrapolation often co-occur, isolating output extrapolation reveals whether a model's learned function generalizes beyond the values it has seen.

Practical relevance. This evaluation mirrors deployment scenarios where models trained on typical cases must handle extreme or unusual inputs—a common requirement in financial modeling, engineering simulations, and risk assessment.

We vary the size of the dataset (*row cap*) over $\{32, 64, 128, 256, 512, 1024, 2048\}$ and apply splits after subsampling. All preprocessing (standardization, imputation) is fit on the context set only and then applied to the query set.

3.2. Models evaluated

We evaluate ten representative methods spanning four families: (1) *Tree ensembles*—XGBoost (Chen and Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), LightGBM (Ke et al., 2017), and Random Forest (Breiman, 2001)—represent the conventional baselines that dominate tabular benchmarks. (2) *Prior-fitted networks*—TabPFN v2 and v2.5 (Hollmann et al., 2023; 2025)—are foundation models pretrained on synthetic tabular tasks, designed for few-shot generalization. (3) *Deep tabular models trained per dataset*—RealMLP-TD-S, TabM (Gorishniy et al., 2024), and xRFM—represent recent neural architectures with improved representation learning. Finally, (4) *in-context learning (ICL)* using `GPT-OSS-120B` (Agarwal et al., 2025) serves as a reference point for what is achievable through example-based reasoning.

For tree ensembles, we use publicly available implementations with typical depth and learning-rate configurations. Deep tabular models follow their default training recipes without extensive hyperparameter tuning, as our goal is to evaluate standard practice rather than heavily optimized setups. TabPFN v2 is evaluated using the `TabPFNRegressor` interface; TabPFN v2.5 uses the Prior Labs API with configuration `model_path="v2.5_default"` and eight estimators. All preprocessing—standardization and imputation—is fit on the context split only and then applied to the query split, following standard practice for preventing data leakage.

For ICL, we serialize each row as key–value text and request a JSON list of predictions. The serialization follows a straightforward format:

```
CONTEXT: x1=5.2, x2=3.1, ..., y=42
CONTEXT: x1=2.8, x2=7.0, ..., y=31
...
QUERY 0: x1=4.5, x2=6.2, ...
QUERY 1: x1=1.9, x2=8.3, ...
```

We parse the model's response into a JSON list of predicted y values, with retry logic (up to 10 attempts) for parsing failures. Due to context-length constraints, ICL is evaluated only for row caps $\{32, 64, 128\}$ with temperature 0.

3.3. Metrics

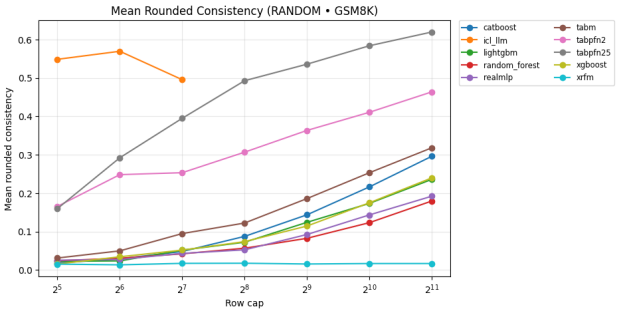
Our primary metric in the paper is *rounded consistency*, defined as exact-match accuracy after rounding predictions

Table 2. Rounded consistency (%) on GSM8K (mean across 100 problems). ICL is limited to $\leq$ 128 rows due to context length.

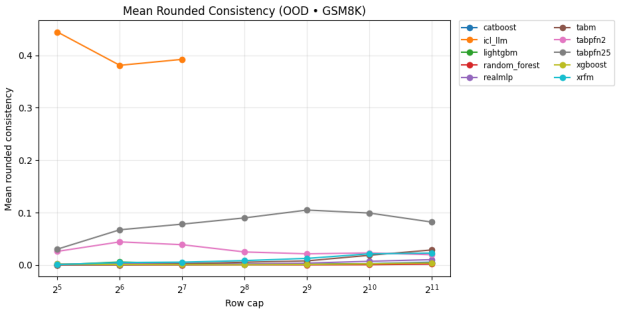
	RANDOM *split*			OOD *split*		
Model	32	128	2048	32	128	2048
ICL (LLM)	54.9	49.6	–	44.4	39.2	–
TabPFN v2.5	16.0	39.5	**62.0**	**3.0**	**7.8**	**8.2**
TabPFN v2	16.6	25.3	46.4	2.6	3.8	1.9
TabM	3.3	9.8	31.6	0.0	0.2	3.8
CatBoost	2.0	4.9	29.6	0.0	0.0	0.5
XGBoost	1.6	5.2	24.0	0.1	0.0	0.4

Table 3. Median R^2 on GSM8K across 100 problems. TabPFN v2.5 achieves near-perfect fit in-distribution and remains positive under distribution shift, while other tabular models go negative.

	RANDOM *split*			OOD *split*		
Model	128	512	2048	128	512	2048
TabPFN v2.5	.93	.98	**.998**	**.75**	**.59**	.28
TabPFN v2	.92	.96	.98	−1.4	−1.4	−1.4
RealMLP-TD-S	.87	.97	.998	−2.2	−1.7	−1.5
TabM	.86	.96	.99	−0.27	.03	.17
XGBoost	.89	.96	.99	−2.6	−2.0	−1.5
CatBoost	.94	.98	**.998**	−2.8	−2.0	−1.5



(a) GSM8K—RANDOM



(b) GSM8K—OOD

Figure 4. Scaling with more rows on GSM8K. Under OOD, ICL and tabular models diverge sharply; under RANDOM, TabPFN v2.5 continues to improve with additional data.

to integer, reflecting whether or not the underlying math problem is correctly solved by the model's prediction:

$$\text{Accuracy} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \mathcal{D}_{\text{test}}} \mathbf{1}[\text{round}(\hat{y}_i) = \text{round}(y_i)]. \quad (1)$$

The *rounded consistency* metric reflects the discrete correctness requirement of mathematical problems. We also report standard tabular metrics (e.g., RMSE, MAE, and R^2) in the appendix for completeness.

4. Results

4.1. GSM8K: interpolation vs. extrapolation

Table 2 reports rounded consistency (exact match after rounding) on 100 GSM8K-derived tasks. For brevity we show three row caps; full results across all row caps and models are in Appendix F.

Interpolation. Under i.i.d. evaluation, TabPFN v2.5 improves steadily with more data and surpasses ICL at 2,048 rows (62.0% vs. 50–57% for ICL at 32–128 rows). This is consistent with a strong inductive prior plus efficient scaling with additional in-distribution samples.

Extrapolation. Under sorted-target OOD evaluation, the picture changes: ICL maintains ∼40% consistency, while all tabular models fall below 10%. TabPFN v2.5 remains the strongest tabular baseline, yet its exact-match accuracy stays in the single digits.

The R^2–consistency gap. The regression objective can hide a failure to compute exact outputs. For example, TabPFN v2.5 reaches median $R^2 = 0.75$ on GSM8K-OOD at 128 rows (Table 3) while achieving only 7.8% consistency (Table 2). ICL exhibits the inverse behavior: it often yields negative R^2 yet matches the exact integer answer on a subset of problems. Appendix D reports additional regression metrics.

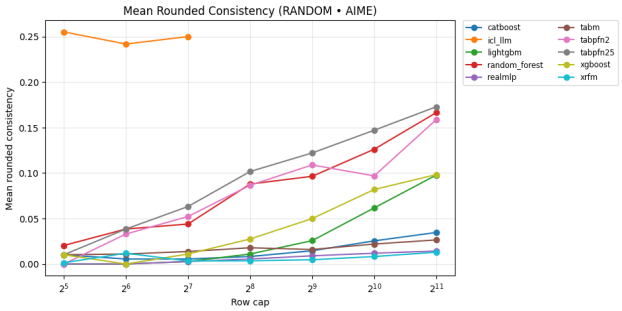
4.2. AIME 2024: a harder regime

AIME tasks substantially reduce performance for all methods. ICL averages 25.5% consistency on AIME-RANDOM at 32 rows and 14.3% on AIME-OOD, while tabular models are near zero under OOD across row caps. Figure 5 shows that AIME amplifies the same qualitative pattern as GSM8K: tabular models can improve steadily under i.i.d. splits, but they struggle to preserve exactness when the evaluation requires extrapolating beyond the observed target range. Full AIME results are in Appendix F.

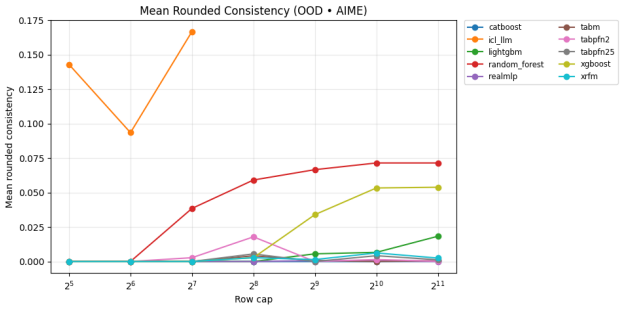
4.3. Understanding failures

The consistent gap between regression fit and exactness suggests that many models capture broad functional trends without inducing the underlying arithmetic rule. Our qualitative inspection reveals three recurring failure modes.

Range-limited predictors. Tree ensembles predict within



(a) AIME—RANDOM



(b) AIME—OOD

Figure 5. Scaling with more rows on AIME 2024. The OOD split isolates output extrapolation and exposes a sharp drop in exact-match performance for tabular predictors, while ICL retains non-trivial consistency in the low-data regime.

the convex hull of training leaf values, making them brittle under sorted-target OOD splits where test targets exceed the observed range. This manifests as near-zero rounded consistency even when in-distribution performance is reasonable. The piecewise-constant nature of tree-based methods fundamentally limits their extrapolation capacity—they cannot extend learned patterns beyond the training support.

Smooth-but-inexact approximations. TabPFN v2.5 emerges as the strongest tabular baseline under both splits and is the only evaluated tabular model with positive median R^2 under OOD (Table 3). Its predictions, however, often fall between neighboring integers around the true answer, producing good R^2 but poor consistency once rounded. This "close but wrong" pattern—high regression fit combined with low exact-match accuracy—indicates that the model approximates the functional form without recovering the discrete computational structure.

Sparse large errors in ICL. ICL achieves substantially higher consistency under OOD, but its errors are heavy-tailed: when it fails to infer the latent rule, predictions can deviate substantially from the target scale, lowering R^2 even when a subset of problems are solved exactly. This pattern reflects the all-or-nothing nature of rule induction—either the model recognizes the underlying computation or it does not.

Model-specific observations. Beyond these general patterns, we observe several model-specific behaviors worth noting. TabPFN v2.5 exhibits the best scaling among all tabular methods on GSM8K-RANDOM, reaching 62.0% at 2,048 rows, while TabPFN v2 plateaus earlier (46.4%). RealMLP-TD-S scales moderately but reaches only 19.3% at 2,048 rows, suggesting that prior-fitted methods hold an advantage on reasoning-intensive tasks where sample efficiency matters. Among tree ensembles, Random Forest shows surprisingly robust performance on AIME-OOD; we suspect that ensemble averaging serves as regularization that limits extreme predictions. xRFM underperforms across all conditions, likely due to the internal validation split consuming training data that is already severely limited at small row caps.

These behaviors collectively suggest that evaluating deterministic tabular tasks benefits from reporting both trend metrics (e.g., R^2) and exactness metrics (rounded consistency). Success on one does not guarantee success on the other, and different model families exhibit distinct failure signatures.

5. Discussion

Across problems, we observe a stable separation between two modes of generalization. Tabular models (including TabPFN) behave like function approximators: they fit the trend extremely well in-distribution and can remain well-calibrated under moderate shift, but they rarely hit the exact integer outputs demanded by the underlying computation. ICL behaves more like example-based program induction: when it recognizes the latent rule from a small number of demonstrations, it extrapolates that rule beyond the observed output range.

Where the split matters. In many real tabular settings, approximate predictions are acceptable—a sales forecast within 5% of the true value is often good enough. In contrast, deterministic domains (pricing formulas, physics simulations, business rules) often have an implicit verifier: a prediction that is numerically close but wrong is still wrong. Rounded consistency makes this distinction explicit and helps separate trend learning from rule recovery. Our results suggest that practitioners working in such domains should be cautious about deploying standard tabular models without verification, even when regression metrics appear strong.

Sample efficiency vs. scaling. ICL is dramatically more sample-efficient on TABULARMATH: at 32 rows it already exceeds the strongest tabular baseline at 128 rows (Table 2). TabPFN v2.5, however, continues to improve with additional in-distribution data and becomes stronger at 2,048 rows. This pattern suggests a trade-off between a fast learner that

can infer rules from few examples and a compact model that benefits from more tabular supervision. The crossover point—around 500 rows for GSM8K—may serve as a rough guideline for practitioners choosing between approaches.

The role of statistical priors. TabPFN's positive OOD R^2 deserves attention. Unlike tree ensembles, which collapse to trivial predictions under extrapolation, TabPFN maintains meaningful functional structure even when exact outputs fail. This suggests that prior-fitted networks learn something transferable about data relationships—perhaps capturing the general shape of arithmetic functions—even if they cannot recover the precise computational rule. The gap between $R^2 = 0.75$ and consistency $< 10\%$ indicates that the statistical prior captures trend but not the discrete structure needed for exactness.

Implications and hybrids. Prior-fitted networks can be strong when throughput matters and evaluation is i.i.d., while ICL offers robustness under extrapolation at far higher inference cost. The complementarity suggests several hybrid designs worth exploring: routing hard cases to an LLM when extrapolation is detected, using ICL to propose a candidate rule that can then be verified programmatically, or distilling verified ICL behaviors into efficient tabular architectures. None of these approaches are straightforward, but the performance gap we document provides clear motivation.

Limitations. TABULARMATH focuses on deterministic arithmetic with integer-valued targets; many real tabular problems are noisy and heterogeneous. Our benchmark is diagnostic rather than comprehensive—it isolates a specific failure mode rather than claiming to represent all tabular tasks. For ICL, we evaluate a single model (`GPT-OSS-120B`) with a fixed prompt template, and we restrict to ≤ 128 rows due to context limits. Performance with other LLMs, longer contexts, or chain-of-thought prompting remains an open question. Finally, our feature engineering is deliberately generic; problem-specific features might improve tabular model performance on individual tasks, though this would undermine the goal of evaluating general-purpose learning.

6. Related Work

TABULARMATH sits at the intersection of tabular learning and algorithmic reasoning. A large body of work evaluates tabular models on i.i.d. data drawn from real-world distributions; our focus instead is on *deterministic* tabular tasks where correctness under extrapolation is the central concern.

Tabular architectures and foundation models. Tree ensembles remain strong baselines on conventional benchmarks (Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022), and recent neural architectures—FT-Transformer (Gorishniy et al., 2021), TabTransformer (Huang et al., 2020), and TabM (Gorishniy et al., 2024)—aim to close this gap through improved representation learning and inductive biases. TabPFN (Hollmann et al., 2023; 2025) takes a different approach, pretraining on millions of synthetic tabular tasks generated from structural causal models to achieve strong few-shot performance. These methods assume that data arises from smooth statistical distributions; TABULARMATH probes what happens when the underlying process is instead a discrete computation.

In-context learning as program induction. Large language models can perform few-shot function learning through in-context learning (Brown et al., 2020; Garg et al., 2022), with theoretical connections to gradient descent (von Oswald et al., 2023) and Bayesian inference (Xie et al., 2022). We use ICL not as a practical system but as an existence proof that the latent computations in TABULARMATH can be recovered from demonstrations alone. This counterpoint to tabular predictors helps isolate what is fundamentally difficult about computational extrapolation.

Verified and program-generated benchmarks. Our synthesis pipeline draws inspiration from program-aided reasoning (Gao et al., 2023; Chen et al., 2022) and recent contamination-resistant evaluations (Zheng et al., 2025). The key distinction is that we use executable verifiers to generate *tabular* datasets with zero label noise, enabling controlled distribution shifts and exact-match evaluation that would be impossible with noisy real-world data.

The full discussion of related work is in Appendix H.

7. Conclusion

We introduce TABULARMATH, a benchmark designed to probe computational generalization in tabular learning. Through our experiments, we find out that, tabular models (e.g., TabPFN) are good estimators, scale gracefully with data, but lack exactness when there are high requirements on precision. ICL achieves precise computation from minimal demonstrations, outperforms tabular models on exact-match by a surprisingly large margin, but can make outrageous predictions in some cases. This distinction has practical implications on which models are more reliable under different scenarios with different precision expectations.

Future directions. TabPFN's positive OOD R^2 suggests that prior-fitted networks capture meaningful structure beyond the training distribution, even if they miss the exact computational rule. Bridging the remaining gap may require more efforts on tabular model architecture, data, etc. TABULARMATH serves as a useful testbed for these directions. To stimulate future research, all code and datasets of TabularMath have been released at `https://github.com/Marco-Cheng/TabularMath`.

Impact Statement

This paper presents a diagnostic benchmark for evaluating tabular machine learning models on deterministic computational tasks. We see several potential impacts worth discussing.

On the positive side, TABULARMATH exposes a previously hidden failure mode of widely-deployed tabular models: catastrophic performance degradation under extrapolation in deterministic domains. Making this limitation visible can help practitioners make more informed decisions about when to trust model predictions, particularly in high-stakes applications involving pricing, financial modeling, or engineering calculations where approximate answers may be unacceptable.

We do not foresee significant negative societal consequences from this work. The benchmark itself consists of mathematical problems and does not involve sensitive data. Our findings highlight limitations rather than capabilities, which we believe contributes to responsible AI development by promoting accurate understanding of model behavior.

One consideration is that our results should not be interpreted as a blanket criticism of tabular models. These methods remain highly effective for the statistical interpolation tasks they were designed for. TABULARMATH targets a specific capability—computational extrapolation—that lies outside the intended use case of most tabular architectures. We encourage practitioners to use our benchmark as one diagnostic tool among many, rather than as a definitive evaluation.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023.

Sercan Ö. Arik and Tomas Pfister. TabNet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

Leo Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, Yejin Choi, et al. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.

Shivam Garg, Dimitris Tsipras, Percy S. Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 24991–25004, 2022.

Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. TabM: Advancing tabular deep learning with parameter-efficient ensembling. *arXiv preprint arXiv:2410.24210*, 2024.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *Advances in Neural Information Processing Systems*, 35, 2022.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637 (8044):319–326, 2025.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR, 2018.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems*, 36:76336–76369, 2023.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31, pages 6638–6648, 2018.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81: 84–90, 2022.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2): 49–60, 2014.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022.

Jianzhu Yao, Kevin Wang, Ryan Hsieh, Haisu Zhou, Tianqing Zou, Zerui Cheng, Zhangyang Wang, and Pramod Viswanath. SPIN-Bench: How well do LLMs plan strategically and reason socially? In *Conference on Language Modeling (COLM)*, 2025. arXiv:2503.12349.

Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hangyi Hao, Jianzhu Yao, Peiyao Sheng, Zixuan Wang, Wenhao Chai, Aleksandra Korolova, Peter Henderson, Sanjeev Arora, Pramod Viswanath, Jingbo Shang, and Saining Xie. LiveCodeBench Pro: How do olympiad medalists judge LLMs in competitive programming? *arXiv preprint arXiv:2506.11928*, 2025.

A. Additional pipeline details

Figure 6 provides a more explicit view of the synthesis workflow used to construct TABULARMATH. In brief, each seed problem is compiled into a generator–verifier pair, validated on the seed instance, and then used to produce a table of inputs and exact outputs.

B. Feature engineering

To provide tabular learners with common arithmetic cues without hand-tuning per problem, we generate a fixed set of generic transforms from each primary slot: (i) log features $\log(1 + |x|)$, (ii) sign and parity indicators, (iii) modular features $x \bmod k$ for small k, and (iv) simple encodings for categorical slots when present. The full feature list is released with the dataset.

C. Implementation details

Unless stated otherwise, we use standard, publicly available implementations of each model. Tree ensembles use typical depth/learning-rate configurations; deep tabular models follow their default training recipes. TabPFN v2 is evaluated using `TabPFNRegressor`; TabPFN v2.5 uses the Prior Labs API configuration `model_path=v̈2.5_default¨` with eight estimators. For all baselines, preprocessing (standardization and imputation) is fit on the context split only and applied to the query split.

D. Additional metrics

We report RMSE, MAE, and R^2 in addition to rounded consistency. In particular, we use $R^2 = 1 - frac\sum_i(y_i - haty_i)^2\sum_i(y_i - bary)^2$ computed over the query set for each problem and then aggregate across problems.

E. Technical Details

E.1. Seed Lifting Prompt

The prompt instructs the compiler LLM to parameterize a mathematical problem into a generalized template with generator and verifier components. The prompt template is provided as follows:

System Prompt for Seed Lifting

You are an expert at parameterizing math word problems in GSM8K. You need to generalize a problem by parameterizing as many parts of it as possible without changing the core calculation logic. Return ONE JSON OBJECT only. No code fences, no prose.

① Seed Math Problems from Existing Datasets

Original Problem from GSM8K Dataset

"Mark has a garden with flowers. He planted plants of three different colors in it. 10 of them are yellow, and there are 80% more of those in purple. There are only 25% as many green flowers as there are yellow and purple flowers. How many flowers does Mark have in his garden?"

② Parameterization & Program Synthesis

Parameterized Template with G-V (Generator-Verifier) pair

"... {a} of them are yellow, and there are {b}% more of those in purple. There are only {c}% as many green flowers as there are yellow and purple flowers..."

G: generator.py (samples valid parameters)

```
def generator():
    while True:
        a = random.randint(1, 100)
        b = random.randint(0, 100)
        c = random.randint(0, 100)
        # Ensure integer constraints
        purple = (1 + b/100) * a
        total_yp = a + purple
        green = (c/100) * total_yp
        if purple == int(purple) and \
           green == int(green):
            return {'a': a, 'b': b, 'c': c}
```

V:verifier.py (computes exact answer)

```
def verifier(a, b, c):
    num_yellow = a
    num_purple = (1 + b/100) * a
    num_green  = (c/100) * (a + num_purple)
    return int(num_yellow +
               num_purple +
               num_green)
```

verifier.py implements the exact mathematical logic, guaranteeing correct labels for all generated rows.

③ TabularMath Dataset Generation with G-V Pairs

yellow (a)	*purple_pct (b)*	*green_pct (c)*	*expected answer (y)*
20	50	40	70
15	100	20	54
40	50	12	112
8	25	50	27
10	80	25	?
⋮	*(a simplified version with no further feature engineering, 2,048 rows per problem)*		

④ Model Evaluation with TabularMath

Split Strategies:

- Test with $2^5, 2^6, ...2^{11}$ rows for each model.
- *Random (i.i.d.):* 80% train / 20% test
- *OOD (Extrapolation):* Train on rows with smallest 80% of y; test on largest 20%

OOD tests whether models extrapolate beyond training distribution, evaluating the capability of discovering the underlying logic pattern.

Metrics:

- Standard regression: R^2, RMSE, MAE
- *Rounded Consistency* (*primary metric*):

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\big[\text{round}(\hat{y}_i) = \text{round}(y_i)\big]$$

Exact-match accuracy after rounding to integers (i.e. the ability of solving real math tasks).

Figure 6. Pipeline used to construct and evaluate TabularMath.

User Prompt Structure

SEED
`question_id`: *¡example_id¿*
`question`: *¡original question text¿*
`answer`: *¡ground truth answer¿*

GOAL
Output a SINGLE JSON object implementing a minimal, coherent augmentation spec:

- `text_templates`: list of paraphrased/translated versions using `[slot_name]` placeholders. Allowed changes include: paraphrasing, translation to other languages (Chinese, French, Spanish, etc.), changing background story or entities while preserving slot semantics. Do NOT leak the final answer inside templates.
- `slots`: dict or list with fields: `name`, `kind` ∈ {`int`, `float`, `choice`, `str`, `entity`, `unit`}, optional `interval`/`map`, `weight`, `base_value`, `meta`.
- `verifier`: `{"type":"python", "code":"def verifier(assign): ... return True, y"}` — validates whether the input assignment is valid, returns the answer for valid assignments (if invalid, returns `False, None`).
- `generator`: `{"type":"python", "code":"def generator(rng): ... return assign"}` — randomly generates an assignment coherent with the verifier.
- `base_assignment`: assignment corresponding to the original seed question; MUST pass the verifier.

RULES

- Use ONLY `[slot_name]` tokens in text_templates (no {braces}). Slot names must be ASCII snake_case.
- The set of slot names in text_templates, slots, base_assignment, verifier inputs, and generator outputs MUST MATCH exactly.
- Code restrictions: pure Python 3; no I/O; no imports; we provide `math` and `random` at runtime (refer to them as `math.gcd`, `math.lcm`, etc.).
- Output `y` should be numeric when the problem is numeric (usual GSM8K).
- The output of generator MUST PASS the verifier and contribute to a valid new augmented task.

Retry Mechanism. When validation fails (e.g., generator output fails verifier, or verifier produces incorrect answer on base assignment), the prompt includes diagnostic feedback:

Retry Feedback (appended on failure)

RETRY_FEEDBACK
The previous attempt failed. Diagnose and fix the issues below. Ensure generator yields VALID assignments passing the verifier, and that verifier compiles and returns `(bool, y)`.
Issues to fix:
¡specific error messages from validation pipeline¿

```
{
  "text_templates": [
    "If [name1] has [a] apples and [name2] has [b],
     how many more does [name1] have?",
    "[name1] possesses [a] apples while [name2] has [b];
     compute the difference.",
    "[name1] has [a] marbles; [name2] has [b]. Find [a] - [b].",
    "Suppose [name1] bought [a] stickers and [name2] bought [b].
     How many more did [name1] buy?",
    // Chinese/French/Spanish translations...
  ],
  "slots": {
    "a": {"kind":"int", "interval":[0,1],
          "map":{"kind":"int_range","lo":5,"hi":50,"step":1},
          "base_value": 12},
    "b": {"kind":"int", "interval":[0,1],
          "map":{"kind":"int_range","lo":1,"hi":50,"step":1},
          "base_value": 3},
    "name1": {"kind":"entity",
              "meta":{"names":["Alice","Xiao Ming","Jean","Lucia"]},
              "base_value":"Alice"},
    "name2": {"kind":"entity",
              "meta":{"names":["Bob","Xiao Hong","Marie","Diego"]},
              "base_value":"Bob"}
  },
```

```
  "verifier": {
    "type": "python",
    "code": "def verifier(assign):\n
        a = int(assign['a']); b = int(assign['b'])\n
        if a<=b or a<0 or b<0: return False, None\n
        return True, a - b"
  },
  "generator": {
    "type": "python",
    "code": "def generator(rng):\n
        a = rng.randint(5,50)\n
        b = rng.randint(1,a-1)\n
        name1 = 'Alice'\n
        name2 = 'Bob'\n
        return {'a':a,'b':b,'name1':name1,'name2':name2}"
  },
  "base_assignment": {"a":12, "b":3, "name1":"Alice", "name2":"Bob"},
  "meta": {"source":"gsm8k", "example_id":"<EXAMPLE_ID>"}
}
```

Listing 1. Output Schema Example (JSON)

The key innovation is that templates support multiple languages and paraphrases while maintaining consistent slot semantics. The verifier both validates the mandatory constraints (e.g., $a > b > 0$) and computes the ground-truth answer, ensuring 100% label accuracy for all generated instances.

E.2. Example: Distance-Rate-Time Problem

Consider a seed problem: "A person runs 15 km at an effective speed (base speed minus 3 km/h headwind). If the running time is 5 hours, find the base speed".

```python
def generator():
    """Sample valid inputs for the problem. Package random imported in our execution environment."""
    base_speed = random.uniform(6, 10)
    headwind = random.uniform(1, 5)
    effective_speed = base_speed - headwind
    total_time = random.uniform(1, 10)
    distance_km = effective_speed * total_time

    return {
        "distance_km": distance_km,
        "headwind_kmh": headwind,
        "total_time_hours": total_time
    }

def verifier(inputs):
    """Compute the answer given inputs."""
    d = inputs["distance_km"]
    h = inputs["headwind_kmh"]
    t = inputs["total_time_hours"]
    # effective_speed = base_speed - headwind
    # time = distance / effective_speed
    # Therefore: base_speed = distance/time + headwind
    base_speed = d / t + h
    return base_speed
```

Listing 2. Generator and Verifier Example

E.3. Details About the Evaluation Protocol

Row Caps. We evaluate our models at row caps $32, 64, 128, 256, 512, 1024, 2048$. For subsampling we use `pandas.DataFrame.sample(n=rowcap, random_state=2025)` before splitting.

Construction of OOD Split. We sort rows by their target value y. We then form our context or training set from the lower 80% (the smallest output) and our query or test set from the upper 20% (the largest output). In doing so, this ensures that

there is no overlap in the range of y-values of the test distribution with the training support.

Construction of RANDOM Split. We construct an 80/20 random split as standard using `train_test_split` with `random_state=2025` (42 for TabPFN v1 compatibility).

ICL Serialization Format.

```
CONTEXT: x1=5.2, x2=3.1, x3=0.5, ..., y=42
CONTEXT: x1=2.8, x2=7.0, x3=1.2, ..., y=31
...
QUERY 0: x1=4.5, x2=6.2, x3=0.8
QUERY 1: x1=1.9, x2=8.3, x3=2.1
```

We parse the LLM's response into a JSON list of predicted y values. If parsing fails, we will retry up to 10 times with logging.

F. All Results

F.1. Comparison of Complete Models

In Table 4, we provide the full results for all models across all configurations. All the numbers represent mean rounded consistency (%) averaged over the number of problems in each family.

F.1.1. Model-Specific Insights

TabPFN. Both TabPFN v2 and v2.5 perform strongly on Random splits across small to medium-sized scales as expected given their design focus on a few-shot regime. TabPFN v2.5 exhibits the best scaling of all table-based models on GSM8K-Random, achieving 62.0% at 2,048 rows in this case. However, both TabPFN v2 and v2.5 have very poor performance on OOD splits, which indicates that their statistical priors fail to represent the computational structure.

RealMLP. Scales moderately well on this benchmark, attaining only 19.3% at 2,048 rows on GSM8K-Random. This lower-than-TabPFN performance is indicative that prior-fitted methods may provide superior performance on reasoning intense tasks where sample efficiency matters.

Tree Models. XGBoost (Chen and Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), LightGBM (Ke et al., 2017), and Random Forest (Breiman, 2001) demonstrate limited but reasonable scalability, though they yield extremely weak absolute performance. There is an inherent limitation of tree-based models in terms of being able to extrapolate beyond their piecewise-constant representation of the underlying function. In addition to this, Random Forest has somewhat surprisingly good robustness on AIME-OOD; we suspect that ensemble averaging serves as regularization to limit extreme predictions.

xRFM. Is severely underperforming in all cases, and therefore, we believe that the reason for this is largely the fact that there is very little hyperparameter optimization, and the internal validation split used during training reduces what is already a severely limited amount of training data. Therefore, we hypothesize that if sufficient tuning were performed, xRFM could potentially become a viable alternative.

F.2. Additional Measures: R^2, RMSE, and MAE

While rounding consistency measures exact match correctness, it is also useful to consider traditional regression metrics in order to understand the quality of model fit. Tables 5–8 list the median R^2, RMSE, and MAE on normalized targets (i.e., zero mean, unit variance per task) for all conditions beginning at 128 rows. We use the median instead of the mean in order to avoid the distortion effects of outliers and differences in scales.

Important Insight. The difference between the high values of R^2 and low consistency illustrate that while models can generate smooth approximations of the target function, they do not accurately represent the discrete logical structure required to produce exact answers. For instance, using TabPFN v2.5 on GSM8K-Random at 2048 rows yields median $R^2 = 1.00$ but only 62.0% consistency—it is incorrect ("close but wrong") on the remaining 38%.

Negative R^2 Interpreted. Most models (other than TabPFN v2.5) achieve a negative median R^2 under OOD conditions; i.e., predictions are no better than simply predicting the training mean. Since we use the median instead of the mean for

Table 4. Complete rounded consistency results (%).

Split	*Model*	*32*	*64*	*128*	*256*	*512*	*1024*	*2048*
				GSM8K				
RANDOM	ICL	54.9	57.0	49.6	—	—	—	—
RANDOM	TabPFN v2.5	16.0	29.2	39.5	49.3	53.6	58.4	62.0
RANDOM	TabPFN v2	16.6	24.8	25.3	30.7	36.3	41.1	46.4
RANDOM	RealMLP-TD-S	2.0	3.4	4.0	5.4	9.2	14.1	19.3
RANDOM	TabM	3.3	5.6	9.8	12.5	18.8	25.2	31.6
RANDOM	Random Forest	2.3	2.4	4.6	5.5	8.2	12.3	17.9
RANDOM	XGBoost	1.6	3.5	5.2	7.5	11.3	17.5	24.0
RANDOM	CatBoost	2.0	2.8	4.9	8.8	14.4	21.6	29.6
RANDOM	LightGBM	2.3	2.5	5.2	7.3	12.4	17.4	23.6
RANDOM	xRFM	1.6	1.4	1.7	1.7	1.6	1.7	1.7
OOD	ICL	44.4	38.1	39.2	—	—	—	—
OOD	TabPFN v2.5	3.0	6.7	7.8	9.0	9.5	9.9	8.2
OOD	TabPFN v2	2.6	4.4	3.8	2.5	2.1	2.3	1.9
OOD	RealMLP-TD-S	0.1	0.2	0.3	0.2	0.3	0.7	1.0
OOD	TabM	0.0	0.2	0.2	0.3	1.0	2.5	3.8
OOD	Random Forest	0.0	0.0	0.0	0.0	0.0	0.0	0.1
OOD	XGBoost	0.1	0.1	0.0	0.0	0.1	0.2	0.4
OOD	CatBoost	0.0	0.0	0.0	0.1	0.1	0.2	0.5
OOD	LightGBM	0.0	0.6	0.2	0.2	0.1	0.2	0.3
OOD	xRFM	0.1	0.4	0.5	0.6	1.0	1.8	2.0
				AIME				
RANDOM	ICL	25.5	24.2	25.0	—	—	—	—
RANDOM	TabPFN v2.5	1.0	3.8	6.3	10.2	12.2	14.7	17.3
RANDOM	TabPFN v2	0.0	3.3	5.2	8.7	10.9	9.7	15.9
RANDOM	RealMLP-TD-S	0.0	0.0	0.3	0.5	0.9	1.2	1.4
RANDOM	TabM	1.0	1.1	1.4	1.8	1.6	2.2	2.6
RANDOM	Random Forest	2.0	3.8	4.4	8.8	9.6	12.6	16.6
RANDOM	XGBoost	1.0	0.0	1.1	2.7	5.0	8.2	9.8
RANDOM	CatBoost	1.0	0.5	0.5	0.8	1.5	2.5	3.5
RANDOM	LightGBM	0.0	0.0	0.3	1.1	2.6	6.2	9.8
RANDOM	xRFM	0.1	1.2	0.3	0.4	0.5	0.8	1.3
OOD	ICL	14.3	9.3	16.7	—	—	—	—
OOD	TabPFN v2.5	0.0	0.0	0.0	0.5	0.0	0.4	0.1
OOD	TabPFN v2	0.0	0.0	0.3	1.8	0.0	0.1	0.0
OOD	RealMLP-TD-S	0.0	0.0	0.0	0.0	0.1	0.0	0.0
OOD	TabM	0.0	0.0	0.0	0.4	0.1	0.0	0.1
OOD	Random Forest	0.0	0.0	3.8	5.9	6.7	7.1	7.1
OOD	XGBoost	0.0	0.0	0.0	0.3	3.4	5.3	5.4
OOD	CatBoost	0.0	0.0	0.0	0.0	0.0	0.0	0.0
OOD	LightGBM	0.0	0.0	0.0	0.0	0.6	0.7	1.8
OOD	xRFM	0.0	0.0	0.0	0.3	0.1	0.6	0.2

this calculation, we eliminate the effects of extreme outliers on scale mismatch problems. TabPFN v2.5 has the highest R^2 (median 0.76 at 128 rows) on GSM8K indicating that there is some generalizable knowledge or structure learned from data even if exact predictions fail.

G. Reproducibility

G.1. Computational Resources

Hardware. We conduct all tabular model experiments using an Apple MacBook Pro (M2 Max, 32GB RAM). For TabPFN v2.5 we employed the Prior Labs API. For ICL, we utilize GPT-OSS-120B with retry logic for parsing failures.

Estimated Run Times.

Table 5. GSM8K-RANDOM regression metrics (standardized scale, *median* across problems). ICL RMSE/MAE are on the original scale and not directly comparable.

	R^2					*RMSE*					*MAE*				
Model	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*
ICL	.84	—	—	—	—	—	—	—	—	—	—	—	—	—	—
TabPFN v2.5	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*.02*	*.01*	*.01*	*.01*	*.01*	*.01*	*.01*	*.00*	*.00*	*.00*
TabPFN v2	.99	*1.00*	*1.00*	*1.00*	*1.00*	.09	.07	.04	.03	.02	.06	.03	.02	.02	.01
RealMLP-TD-S	.77	.88	.95	.98	.99	.46	.32	.23	.14	.08	.32	.23	.15	.09	.05
TabM	.96	.98	.99	*1.00*	*1.00*	.20	.12	.07	.04	.03	.14	.08	.05	.03	.02
xRFM	.90	.96	.99	.99	*1.00*	.32	.19	.12	.07	.04	.20	.13	.07	.04	.03
XGBoost	.82	.89	.96	.98	.99	.41	.31	.19	.13	.09	.30	.21	.13	.09	.06
CatBoost	.81	.94	.98	.99	*1.00*	.39	.23	.13	.07	.05	.29	.16	.09	.05	.02
LightGBM	.81	.93	.97	.98	.99	.38	.26	.17	.12	.08	.29	.18	.11	.08	.05
Rand. Forest	.79	.88	.93	.96	.98	.43	.34	.25	.20	.15	.32	.23	.18	.14	.10

Table 6. GSM8K-OOD regression metrics (standardized scale, *median* across problems). Negative R^2 indicates predictions worse than the mean.

	R^2					*RMSE*					*MAE*				
Model	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*
ICL	-1.3	—	—	—	—	—	—	—	—	—	—	—	—	—	—
TabPFN v2.5	*.76*	*.76*	*.75*	*.59*	*.28*	*.57*	*.46*	*.44*	*.53*	*.66*	*.44*	*.37*	*.33*	*.33*	*.45*
TabPFN v2	-1.3	-1.3	-1.4	-1.4	-1.4	1.67	1.55	1.61	1.56	1.50	1.33	1.22	1.16	1.19	1.09
RealMLP-TD-S	-4.5	-2.9	-2.2	-1.7	-1.5	1.42	1.22	1.12	1.03	.95	1.32	1.09	1.00	.87	.78
TabM	-1.5	-.75	-.27	.03	.17	.97	.81	.70	.58	.52	.89	.71	.57	.48	.42
xRFM	-2.7	-2.3	-2.1	-1.8	-1.5	1.18	1.11	1.03	.96	.92	1.00	.95	.86	.76	.70
XGBoost	-4.0	-3.1	-2.6	-2.0	-1.5	1.43	1.32	1.18	1.07	.98	1.30	1.17	1.04	.91	.80
CatBoost	-5.2	-4.0	-2.8	-2.0	-1.5	1.55	1.37	1.21	1.05	.96	1.38	1.22	1.07	.88	.76
LightGBM	-3.4	-2.4	-1.9	-1.7	-1.6	1.24	1.13	1.04	1.00	.96	1.12	1.00	.90	.85	.81
Rand. Forest	-4.3	-3.6	-3.4	-2.8	-2.7	1.53	1.38	1.29	1.23	1.17	1.37	1.25	1.16	1.06	1.01

Table 7. AIME-RANDOM regression metrics (standardized scale, *median* across problems). Competition-level problems show lower R^2 overall.

	R^2					*RMSE*					*MAE*				
Model	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*
ICL	-.31	—	—	—	—	—	—	—	—	—	—	—	—	—	—
TabPFN v2.5	*.80*	*.77*	.77	*.84*	.81	*.19*	.30	.33	.33	.41	*.10*	*.08*	*.13*	*.10*	*.08*
TabPFN v2	.75	.70	.73	.79	*.83*	.30	.35	.44	.45	.35	.21	.12	.19	.20	.12
RealMLP-TD-S	.33	.56	.70	.78	*.83*	.68	.48	.50	.52	.47	.41	.32	.26	.17	.13
TabM	.59	*.78*	*.88*	.83	.81	.45	*.30*	*.31*	.38	*.36*	.30	.15	.18	.12	.09
xRFM	.54	.62	.71	.82	.80	.59	.42	.46	.42	.39	.38	.22	.24	.15	.19
XGBoost	.38	.54	.72	.74	.78	.56	.51	.49	.48	.43	.37	.31	.26	.17	.16
CatBoost	.56	.60	.75	.77	.79	.54	.42	.45	.44	.38	.33	.27	.22	.15	.13
LightGBM	.47	.53	.75	.74	.78	.64	.52	.45	.48	.42	.48	.37	.27	.21	.20
Rand. Forest	.36	.44	.65	.77	.78	.54	.58	.52	.46	.45	.40	.34	.29	.21	.19

Table 8. AIME-OOD regression metrics (standardized scale, *median* across problems). All models show negative R^2, confirming extrapolation failure.

	R^2					*RMSE*					*MAE*				
Model	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*	*128*	*256*	*512*	*1k*	*2k*
ICL	-1.9	—	—	—	—	—	—	—	—	—	—	—	—	—	—
TabPFN v2.5	*-1.3*	*-.69*	*-.77*	*-.53*	-1.2	5.6	4.8	4.5	4.8	6.7	4.8	3.9	3.8	4.0	5.4
TabPFN v2	-1.5	-1.2	-1.6	-1.2	-1.2	7.3	6.5	6.5	6.8	6.3	5.6	5.6	5.0	5.4	4.8
RealMLP-TD-S	-2.0	-2.3	-1.3	-1.3	-1.2	2.10	2.05	2.03	1.98	1.90	1.46	1.38	1.27	1.16	1.16
TabM	*-1.0*	-.90	*-.55*	-.61	*-.46*	*1.99*	*1.93*	*1.70*	*1.64*	*1.64*	*1.31*	*1.16*	*1.07*	*1.04*	*.97*
xRFM	-2.0	-2.5	-1.3	-1.3	-1.4	2.07	2.09	1.90	1.87	1.90	1.31	1.33	1.15	1.14	1.13
XGBoost	-2.1	-2.5	-1.4	-1.7	-1.5	2.08	2.01	2.00	1.88	1.79	1.40	1.27	1.17	1.12	1.12
CatBoost	-2.3	-2.0	-1.5	-1.2	-1.1	2.19	2.14	2.03	1.94	1.82	1.48	1.39	1.23	1.14	1.08
LightGBM	-2.0	-1.6	-1.3	*-1.1*	*-1.1*	2.09	1.95	1.94	1.87	1.80	1.37	1.30	1.17	1.13	1.14
Rand. Forest	-2.2	-2.0	-1.5	-1.5	-1.5	2.12	2.06	2.01	1.97	1.95	1.41	1.39	1.28	1.17	1.13

- **Generating Datasets.** <24 hours using LLM APIs for seed lifting and table synthesis across all 114 problems.
- **Evaluating Classical Models.** Approximately 50 hours on MacBook Pro to run all classical models (XGBoost, LightGBM, CatBoost, Random Forest, TabM, xRFM, RealMLP-TD-S) across all problems, row counts, and splits.

- **Evaluating TabPFN.** Approximately 4 days due to API rate limits; TabPFN itself had very fast inference (approximately 1 second per prediction).
- **Evaluating ICL.** Estimated runtime will depend on API throughput; approximately 2–3 hours for full evaluation with GPT-OSS-120B.

Total Computational Time. The total pipeline time (generation + evaluation) required approximately one week of wall-clock time, limited primarily by API rate limits rather than computational complexity. With local parallelization, we could significantly decrease wall-clock time for tabular model training and evaluation across problems.

G.2. Availability of Code and Data

We provide:

- The 114 generated tables as both CSV and Parquet files.
- Scripts for evaluating all 10 models.
- Raw experimental results (in JSON report format).
- Scripts for aggregating and creating plots of results.
- Scripts for generating data with example prompts.
- Pre-computed summary statistics.

See project page at `https://github.com/Marco-Cheng/TabularMath`.

G.3. Verification Checklist for Reproducibility

To ensure that our results can be reproduced:

1. *Data*: Use the provided Parquet files located in the `datasets/` directory. There should be exactly 2,048 rows in each file.
2. *Splits*: If you are going to create your own split for either `ood_split` or `random_split`, make sure you honor those flags to create the same splits.
3. *Row Caps*: Only include the specified number of rows based upon the `{max_rows` flag that corresponds to the study grid: 32, 64, 128, 256, 512, 1024, 2048.
4. *Standardization*: Include the `{standardize` flag for all classical and TabPFN evaluations.
5. *Logging*: Save all JSON output for compatibility with the aggregation pipeline.

H. Related Work

This appendix contains the full related work discussion from the longer version of the paper. The main paper includes a condensed summary in Section 6.

H.1. Deep Learning for Tabular Data

Numerous innovative architectures have been developed in recent years for deep learning in tabular data. However, despite the development of these new architectures, deep neural networks have not yet replaced tree-based ensemble methods for tabular data. TabNet (Arik and Pfister, 2021) mimics decision tree feature selection through sequential attention. SAINT (Somepalli et al., 2021) applies self-attention across both rows and columns. FT-Transformer (Gorishniy et al., 2021) is an adaptation of transformers for tabular inputs using feature tokenization. NODE (Popov et al., 2020) develops a differentiable version of decision tree ensembles. TabTransformer (Huang et al., 2020) is specifically designed to be used with categorical

features, while other work (Gorishniy et al., 2022; 2024) on numerical embeddings and parameter-efficient ensembling continue to enhance deep tabular methods.

Extensive assessments (Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022; Borisov et al., 2021; McElfresh et al., 2023) have shown that, regardless of the type of assessment, XGBoost and CatBoost consistently produce better results than deep models on datasets with fewer than 10^5 rows. Therefore, the question arises: do neural networks' advantages also exist in structured data? Or are the benchmarks limiting the neural networks' advantages?

H.2. Tabular Foundation Models

TabPFN (Hollmann et al., 2023; 2025) develops a foundation model that is trained on millions of synthetic datasets generated from structural causal models (SCMs). The approach is based upon prior-data fitted networks (Müller et al., 2022) and demonstrates excellent few-shot performance, where it often matches XGBoost with tens of examples. TabPFN v2.5 enhances TabPFN with larger models and improved priors.

These approaches are based upon the assumption that the data is derived from *statistical distributions*—Gaussian mixtures, polynomial relationships, causal graphs—and therefore the prior over functions is smooth and continuous. TABULARMATH evaluates if models can learn *computational structure*—a fundamentally different challenge, which requires discrete computational rule induction.

H.3. In-Context Learning

Large language models demonstrate impressive abilities through in-context learning (ICL), (Brown et al., 2020) and theoretical work has established connections between ICL and gradient descent (von Oswald et al., 2023; Akyürek et al., 2023) and Bayesian inference (Xie et al., 2022). In addition, studies have demonstrated that ICL can enable a wide variety of function classes to be implemented by a transformer given very few demonstrations (Garg et al., 2022). Induction heads (Olsson et al., 2022) have provided mechanistic explanations for the ICL behavior of transformers. The objective of this paper is to investigate whether ICL abilities of transformers can extend to tabular prediction tasks that require computational reasoning.

H.4. Program-Aided Reasoning

Program-aided language models (PAL (Gao et al., 2023)) and Program-of-Thoughts (PoT (Chen et al., 2022)) are two examples of how code execution can enhance reasoning: PAL generates Python code that computes the solution to a math problem, while PoT generates a program that computes the solution. The opposite of this paradigm is evaluated in this paper; namely, instead of using programs to solve problems, programs are used to generate problem variations. This is related to synthetic data approaches, such as Phi-1 (Gunasekar et al., 2023), which extended them to the tabular domain via formal verification.

H.5. Mathematical Reasoning Benchmarks

Grade-school math problems that require multi-step arithmetic are represented in GSM8K (Cobbe et al., 2021). The MATH dataset (Hendrycks et al., 2021) presents problems at a competitive level across multiple domains. A recent example of demonstrating that large language models can achieve high levels of mathematical reasoning through specialized pre-training is Minerva (Lewkowycz et al., 2022). Recent work on process reward models (Lightman et al., 2023) and math-specialized LLMs (Azerbayev et al., 2023; Shao et al., 2024) continues to raise performance levels on these benchmarks. AlphaGeometry (Trinh et al., 2024) achieves Olympiad-level geometry without human demonstrations through neuro-symbolic methods.

In TABULARMATH, we utilize these benchmarks as *seeds* to generate tabular datasets that require the same underlying reasoning capabilities, rather than to evaluate the models directly.

H.6. AI Evaluation and Benchmarking

The evaluation of AI and benchmarking has recently emphasized the necessity of thoroughly evaluating AI, beyond the common benchmarks (Vanschoren et al., 2014). Evaluating code generation benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) evaluate the capability of a program to synthesize. LiveCodeBench Pro (Zheng et al., 2025) evaluate LLMs on competitive programming with contamination-free problems annotated by Olympiad medalists.

SPIN-Bench (Yao et al., 2025) evaluate strategic planning and social reasoning in multi-agent environments. All of the above benchmarks share the motivation behind this paper: current evaluations may fail to capture the full range of model capabilities. TABULARMATH expands this principle to tabular learning, testing computational reasoning rather than language understanding.

H.7. Compositional Generalization

Systematic reasoning requires the ability to generalize compositionally — applying known rules to novel combinations (Lake and Baroni, 2018; Hupkes et al., 2020). Recent work has shown that transformers experience difficulty with compositional tasks (Dziri et al., 2023), especially those that involve performing multi-step symbolic operations. The OOD evaluation regime directly assesses compositional generalization: the model must apply learned computational rules to input ranges that are not seen during training.

I. LLM Usage Disclosure

Large language models were used in two distinct capacities during this work. First, in the *data synthesis pipeline*, we used an LLM as a compiler to convert seed math problems into generator–verifier pairs (Section 2). This usage is part of the methodology itself and is described in detail in the paper. The generated programs were validated against ground-truth answers, so any LLM errors in code generation were caught by our verification pipeline.

Second, during *manuscript preparation*, we used LLM assistance for editing prose, checking consistency, and refining exposition, so that the idea could be better presented and the message could be better conveyed. All research ideas, experimental design, implementation, analysis, and interpre- tation of results were conducted by the human authors. The AI assistants helped refine word choice and clarity but did not contribute to the intellectual content of the work.