ARCHITECTUREIQ: ON THE MEASURE OF MODEL INTUITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Top researchers have good intuition, but do language models have as good intuition about model training as top AI researchers? To measure model intuition of LLMs and humans, we introduce the **ArchitectureIQ** benchmark. Each question presents a synthetic dataset and several training recipes, and the test-taker is asked to predict the recipe yielding the best test metric. Overall, we find that LLMs' model intuition is good but has four limitations: **(1) The intuition is imperfect, or even sub-human in some cases.** Frontier models achieve around 76% accuracy (random choice 33%) vs best human researcher (66.0%), yet remain far from perfect. For architecture-only questions, best human achieves 65% while GPT-6 Astra only has 38%. **(2) The intuition is empirical, not structured**, supported by the fact that more CoT compute does not lead to substantial improvement. Unlike math, we still lack a "Science of AI" language that enables structured reasoning on AI. **(3) The intuition is not maximally condensed**, and can be further compressed into a knoledge base. Our constructed knowledge base with only 20 items yields large gains for weak models: GPT-4o equipped with the accumulated knowledge almost matches the performance of Claude Opus 5. **(4) The intuition is insensitive to dataset properties**, but the best model should in general depend on data properties. This suggests that data is the real "dark matter" in AI – LLMs (so do human researchers) understand too little about data, even less than model architectures.

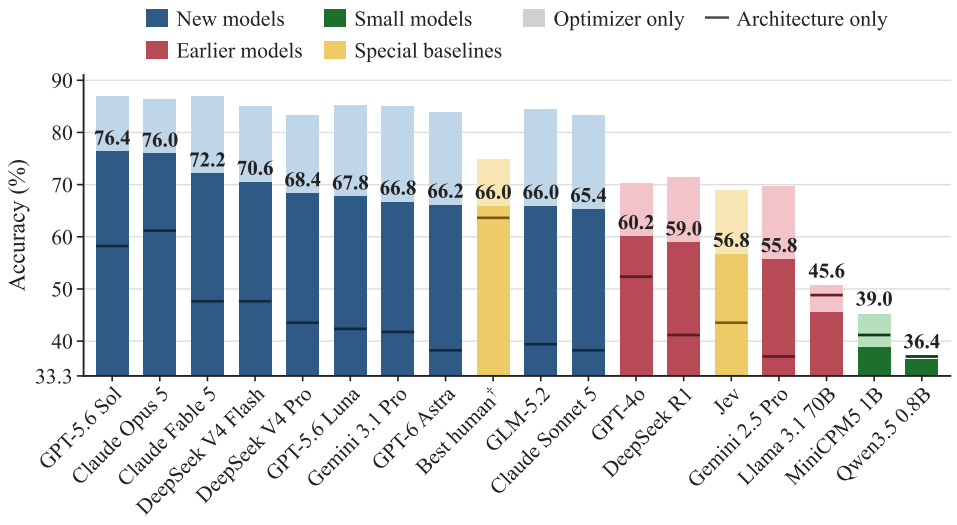


Figure 1: Accuracy on the 500-question, three-choice ArchitectureIQ benchmark (chance 33.3%). Printed numbers are overall accuracy; the light cap marks optimizer-only accuracy and the horizontal rule architecture-only accuracy. The best human (†) is measured on a fixed 50-question subset. Complete per-type results are given in Table 4 (Appendix D).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Top machine learning researchers possess an esoteric intuition: before running an experiment, they can often inspect a change in architecture or training hyperparameters and anticipate how it will affect convergence, stability, and final performance. These predictions are rarely precise, but they are often good enough to distinguish promising experiments from unproductive ones. This ability is central to efficient and innovative research.

As language models are increasingly deployed as agents that design, execute, and iteratively refine machine-learning experiments (Huang et al., 2024; Chan et al., 2025; Lu et al., 2024), they require a similar capability: have LLMs internalized the kind of intuitive judgment that experts use every day? This ability is captured by a simple question:

Given a dataset, does an LLM know which training setting will actually work best, before running the experiment?

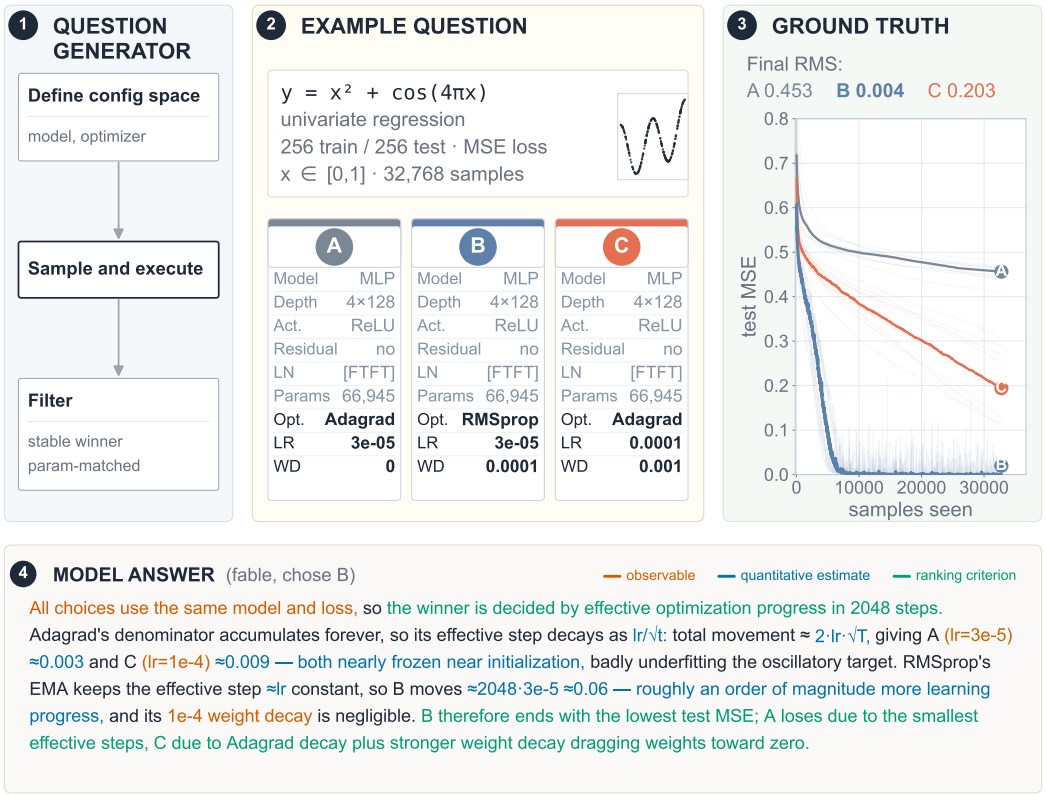


Figure 2: ArchitectureIQ overview. A synthetic dataset and several candidate training recipes are rendered as executable programs. Running those programs across random seeds establishes the ground-truth winner. The evaluated model receives the dataset and recipe descriptions, and predicts which candidate will achieve the best test metric.

We introduce **ArchitectureIQ**, a fully controlled and verifiable probe for studying this capability. Each question presents one synthetic dataset and several training recipes, which are rendered into executable code and trained to generate the ground truth. The test-taker is asked to predict the recipe producing the best outcome metric. Rather than real tasks like LLM training, we deliberately use fully synthesized datasets. The former might be present in the training corpus of frontier LLMs, while the latter separates modeling capabilities from any prior knowledge. Figure 2 illustrates the generation pipeline, one example question, and its executed ground truth.

We ship a static 500-question benchmark supporting fair model comparison. The performance of LLMs forms a hierarchy and is generally good – on the static benchmark, the frontier models achieve

around 76.0% accuracy, in contrast to 33.3% for random choice and a best of 66.0% among 10 human machine-learning researchers on a 50-question subset. This suggests that LLMs' model intuition is good but still far from perfect. LLMs are even sub-human on architecture-only questions, yielding 38% accuracy (GPT-6 Astra), compared to best human 65%.

Our primary goal is therefore not merely to measure accuracy, but to explore the nature of model intuition presented by LLMs. **We find that LLMs' model intuition have three flawed patterns**: **(1) The intuition is empirical but not structured.** This is supported by the fact that neither in-context learning or test-time scaling substantially improves the performance. This suggests that intuition is more like tricks adopted in alchemy, rather than a rigorous scientific language (e.g., math or chemistry) that enables structured reasoning. **(2) The intuition is not maximally condensed**, and can be compressed into a compact, human-readable knowledge base. The KB substantially improves weaker models: GPT-4o equipped with the KB containing only 20 items matches the raw performance of Claude Opus 5. **(3) The intuition is insensitive to data**. However, the ground truth obviously depends on data properties, which are ignored by LLMs in many cases. This suggests that data is the real "dark matter" in AI – LLMs (so do human researchers) understand too little about data, even less than model architectures.

Our contributions are therefore threefold:

1. **The first benchmark about model intuition.** To the best of our knowledge, ArchitectureIQ presents the first benchmark on measuring model intuition of LLMs and humans.
2. **Revealing LLMs' failure modes.** Our analysis revealed several limitations of LLMs' intuition: (1) empirical not structured; (2) not maximally condensed; (3) insensitive to dataset properties. The identification of these failure modes can point future paths to improving model intuition.
3. **Knowledge base.** LLMs' reasoning traces can be compressed into a knowledge base, which is more compressed and transferable than raw CoT. The knowledge base can elevate weak models' (e.g., GPT-4o) performance to that of frontier models (e.g., Calude Opus 5).

2 Related Work

Training-performance prediction and neural architecture search. Automated machine learning has long treated experiment selection as an optimization problem. Bayesian optimization searches expensive hyperparameter spaces (Snoek et al., 2012), while multi-fidelity methods allocate resources adaptively across configurations (Li et al., 2018; Falkner et al., 2018). Learning-curve extrapolation predicts final performance from partial runs (Domhan et al., 2015; Klein et al., 2017), and neural architecture search increasingly relies on learned performance predictors (White et al., 2021). Tabular and surrogate NAS benchmarks make these searches reproducible and inexpensive (Ying et al., 2019; Dong & Yang, 2020; Zela et al., 2022). These approaches learn from executions or task-specific search data; ArchitectureIQ instead asks whether a general-purpose language model can rank complete training recipes from their dataset and code before observing any run.

Language agents for machine-learning research. General agent benchmarks test language models that interact with tools and environments (Liu et al., 2024), including repository-scale software engineering (Jimenez et al., 2024). More specifically, MLAgentBench, MLE-bench, and the AI Scientist evaluate agents that design, execute, and refine machine-learning experiments (Huang et al., 2024; Chan et al., 2025; Lu et al., 2024), while PaperBench tests end-to-end replication of published AI research (Starace et al., 2025). These settings measure broad workflows in which agents may obtain feedback by running code. ArchitectureIQ isolates an earlier prerequisite: choosing a promising experiment before paying its execution cost.

Reasoning and experience accumulation. Chain-of-thought, self-consistency, and tree-structured deliberation elicit or aggregate intermediate reasoning at inference time (Wei et al., 2022; Wang et al., 2023b; Yao et al., 2023a), while STaR uses model-generated rationales for iterative self-training (Zelikman et al., 2022). ReAct interleaves reasoning with environment actions (Yao et al., 2023b); Reflexion and Self-Refine turn textual feedback into improved subsequent attempts (Shinn et al., 2023; Madaan et al., 2023). ExpeL extracts reusable insights across agent trajectories, and Voyager accumulates an external library of executable skills (Zhao et al., 2024; Wang et al., 2023a). Our pipeline likewise externalizes experience, but scores individual natural-language propositions

against executed training outcomes, yielding a knowledge base whose transfer can be measured independently of parameter updates.

3 ARCHITECTUREIQ: A GENERATIVE AND VERIFIABLE PROBE

ArchitectureIQ evaluates "model intuition" – whether a test-taker (a language model or a human) can predict the outcome of a neural network training experiment without executing it. Each question presents a synthetic dataset and several candidate training recipes, each specifying the model type and architecture, optimizer type and hyperparameters, loss function, batch size and training duration (an example is shown in Figure 2; details are described in Appendix A). These components are presented in both natural language and in Python code that implements the data synthesis and training program. The task of the test-taker is to reason about the expected training behavior and select the recipe that will achieve the best final evaluation metric (e.g. test cross-entropy loss for classification).

Importantly, we deliberately use fully synthesized datasets, in order that models cannot rely on their pretrained knowledge regarding specific datasets. Thus, the information presented in a question, in principle, fully determines the training outcome. This cleanly separates out the prediction ability from any implicitly assumed prior knowledge.

3.1 BENCHMARK CONSTRUCTION

Multiple-choice questions. Question construction follows a single execution-grounded path. A sampled dataset specification is first rendered into synthesis code and executed to materialize fixed training and test data. Compatible candidate specifications are then sampled from controlled pools of models, optimizers, losses, batch sizes, and training budgets. Ground truth is obtained by importing and executing that exact generated code. To form a multi-choice question (N candidate choices), there must exist one clear winner among all candidates, regardless of randomness in initialization and training. This is guaranteed by ensuring the winner to have the best metric across 10 random seeds. With the ArchitectureIQ question generator described above, we generate a benchmark with 500 questions with $N = 3$ (see Appendix A for details).

Question taxonomy. The benchmark contains equal portions of *optimizer-only*, *architecture-only* and *mixed* questions. *Optimizer-only* questions add the constraint that the architecture components of all N candidates are the same; *architecture-only* ones are defined likewise, and *mixed* ones are those with no such constraints.

3.2 EVALUATION RESULTS

Leaderboard. Figure 1 plots the ranking of all evaluated systems on the 500-question benchmark; Table 4 in Appendix D reports the same results broken down by question type. The leading models perform substantially better than both the $1/3$ random baseline and human experts. Nevertheless, even the best model answers nearly one quarter of the questions incorrectly. Besides, the same evaluation is carried out on 10 human machine-learning researchers, who each completed a 50-question subset drawn from the same benchmark. The best participant answers 66.0% of these questions correctly and the mean is 51.0%.

ArchitectureIQ thus reveals a meaningful predictive capability while also exposing a substantial remaining gap between current performance and reliable training-outcome prediction. On the other end, the weakest models provide an additional clue. Their accuracy remains above random baseline, showing that architecture-related priors appear even at a small scale.

Architecture Knowledge as the Weak Point. The most consistent asymmetry is between optimizer-only and architecture-only questions (the light caps and horizontal rules in Figure 1; Table 4). The leading models answer 85–87% of optimizer comparisons correctly but only 58–61% of architecture comparisons (notably, GPT-6 Astra only achieves 38%). Mixed questions are generally closer to optimizer-only performance, suggesting that optimizer differences provide a more reliable signal for judgement when present.

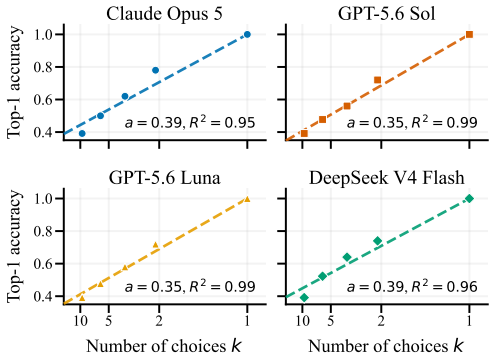


Figure 3: Accuracy as the number of candidate choices varies over $k \in \{1, 2, 3, 5, 10\}$. Points show measured model accuracies; curves fit $\mathrm{acc}(k) = a + (1 - a)/k$.

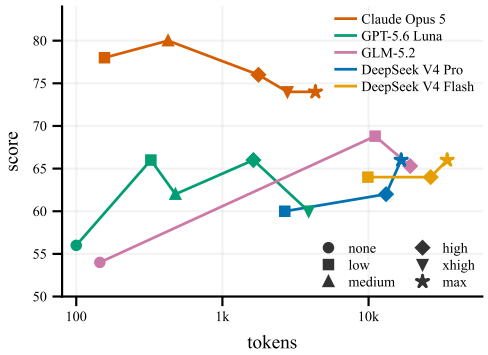


Figure 4: Accuracy versus total output tokens. Increasing inference-time computation produces no consistent accuracy gain beyond the initial reasoning budget.

Optimizer comparisons frequently expose recognizable short-horizon patterns: whether a learning rate is too small to move away from initialization, whether momentum changes the effective step scale, or whether an adaptive optimizer can escape a chance-loss plateau. When the optimizer axis is held fixed, however, architectural comparisons become essential. Width, depth, normalization, residual paths, activation choice, and task structure must be considered jointly. The evaluation results identify this architectural judgment as the principal weak point of current LLMs.

Training Configuration Dominates Dataset Evidence. Each ArchitectureIQ question supplies two potential sources of evidence: the training configurations of the candidates and the dataset on which they are trained. To separate their effect, we construct question pairs that share the same candidate set, only differing in the target dataset (Ben-David et al., 2010; Koh et al., 2021).[1] On those questions where the same candidate wins on both datasets, LLMs achieve 81–95% accuracy. A stronger evidence comes from the cases where the dataset genuinely matters. On the pairs where changing only the dataset alters the winning recipe, combined accuracy drastically collapses to 46–50%. Their predictions are therefore much more accurate when the evidence from training configurations is sufficient. Changing the dataset also rarely changes this preference: across paired questions, models switch their selected candidate on only 6–23% of pairs.

These results show a clear asymmetry: training-configuration knowledge dominates dataset evidence, even exactly where the latter should overturn the former. LLMs rely significantly more heavily on their knowledge of neural networks and optimizers themselves, rather than their interaction with a specific dataset. This asymmetry echoes a broader imbalance in machine learning research: progress has historically centered on model and training-procedure design, while dataset construction and quality have received less systematic attention (Paullada et al., 2021; Sambasivan et al., 2021; Whang et al., 2023; Gebru et al., 2021).

3.3 Scaling with the Number of Choices

Moving beyond three-choice accuracy, we rebuild questions over the same collection of dataset/candidate sets with $k = 2, 3, 5, 10$ choices, together with the trivial $k = 1$ anchor. The four evaluated models follow a common empirical pattern:

We construct a heuristic formula for this scaling behavior: for some $a \in [0, 1]$, for a fraction of a of the questions, the model is certain of its answer, having accuracy 1; whereas for the remaining $(1 - a)$ fraction, it is reduced to random guessing, yielding $1/k$ accuracy. Together, this gives:

$$\mathrm{acc}(k) \approx a + \frac{1 - a}{k},$$

[1]The strict construction protocol and per-model results are reported in Appendix E (Table 5).

Table 1: Comparison of original zero-shot and 10-shot performance

Model	Zero-shot	10-shot
Claude Opus 5	70.0%	70.0%
GPT-5.6 Luna	64.0%	60.0%
DeepSeek V4 Flash	62.0%	65.0%
DeepSeek V4 Pro	62.0%	60.0%

Figure 3 shows the regression results, with $R^2 \geq 0.94$ for all four models. The real judgement capability of the model can be interpreted as a instead of its accuracy on $N = 3$ questions, which corresponds to the intercept in Figure 3.

4 Analysis of LLMs' intuition

Accuracy itself does not identify the knowledge behind a prediction. One may execute a long analysis of training dynamics, or may use a compact heuristic such as preferring the most parameters. In this section, we carry out multiple experiments to separate these possibilities. We characterize the essential knowledge for ArchitectureIQ as more of a pretrained 'intuition', instead of a structured explicit language.

Ineffective In-context Learning. We tested zero-shot and 10-shot performance on a subsample of ArchitectureIQ. As shown in Table 1, it turned out that few-shot prompting does not reliably improve performance, showing no signal of successful in-context learning. A similar situation exists for humans. Human experts perceive the questions sequentially. In principle, the previous questions can be used as a reference for new questions, which essentially constitutes a form of in-context learning. However, observed accuracy progress shows no significant signal of their accuracy rising over time.

Ineffective Test-time Scaling. We next test whether additional inference-time computation allows models to make better use of the available information. The same fixed 50 questions are evaluated at every reasoning-effort tier, from `none` through `low`, `medium`, `high`, `xhigh`, and `max`. Alongside accuracy we record the total number of output tokens each answer costs, so that tiers can be compared on a single dose axis. In addition to accuracy, we compare the answers produced at the different tiers. Figure 4 reports the outcome.[2] Accuracy moves within the question-sampling noise: with 50 samples per tier, differences below roughly 7% are not distinguishable. The gains that do appear (for GLM-5.2) are concentrated in the first few hundred output tokens. More importantly, answers rarely change with longer CoT. Longer traces often contain more detailed discussion, but they generally lead to the same final preference. The behavior is consistent with the finding of no dataset awareness: additional computation repeatedly applies the same candidate-level intuitions rather than discovering a new dataset-conditioned decision rule. Test-time scaling alone therefore does not close the gap caused by the weak evidence sensitivity observed before.

Altogether, the ineffectiveness of neither in-context learning or test-time scaling suggests that model intuition is still more like alchemy rather than a scientific language. If it were a scientific language like math or chemistry, longer CoT should enable structured reasoning.

Rule-based and Structured Predictor Baselines. To determine how much ArchitectureIQ can be compressed without language models, we construct structured predictors from the task parameters. Each question is represented by 87 features including the dataset, architecture, optimizer, budget, effective learning-rate quantities. Figure 5 reports the accuracy of each predictor.

[2]Claude Opus 5 was served through a relay that does not expose a genuine `none` setting, so its curve starts at `low`.

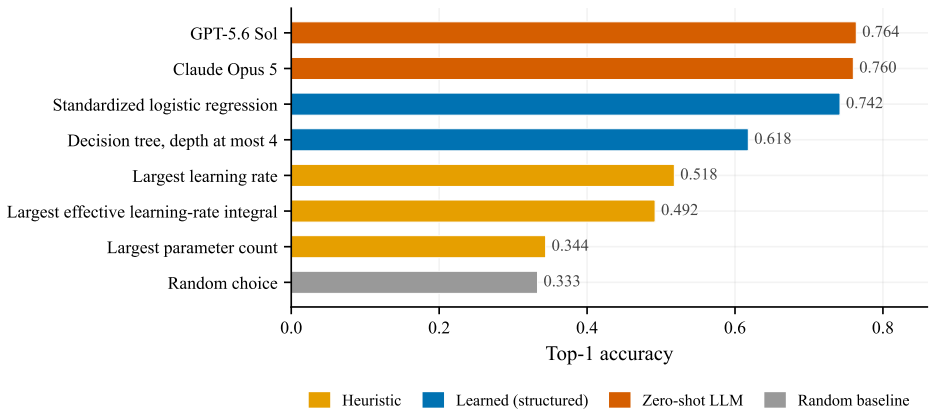


Figure 5: ArchitectureIQ accuracy of rule-based and supervised structured predictors. The analytical decision policy in Figure 8 is evaluated separately.

None of these predictors reaches the leading language models, even though the two learned ones are fitted on in-domain generator labels[3] while the language models are evaluated zero-shot. The best, a standardized logistic regression on the 87 features, reaches 74.2%. A readable depth-four tree reaches 61.8% using learning-rate rank, effective-step rank, and optimizer identity among its dominant decisions, closely resembling the rules of thumb used by human solvers.

These results demonstrate substantial rule-based compressibility. Nonetheless, the remaining gap to LLM accuracy suggests that analytical reasoning adopted by LLM reasoners still outperforms compact heuristics.

5 Online Knowledge Accumulation

Section 4 suggests that strong zero-shot performance requires richer analytical knowledge than can be captured by compact, human-readable rules. To investigate this kind of knowledge, we construct a closed-loop pipeline that accumulates propositions. In this section, we explore how the useful analytical reasoning from frontier LLM reasoning can be extracted and compressed into static knowledge.

Learning Verifiable Propositions Online. The solver prompt may include at most $k = 20$ propositions from the current knowledge base. The solver may cite multiple existing propositions or propose new ones, assigning each with fractional credits adding up to 1. Each selected proposition is shown with its human-readable description and current credibility. We use Claude Opus 5 as the solver to generate a series of complete KB snapshot after 8 epochs.

An epoch consists of $m = 50$ freshly generated ArchitectureIQ questions solved in parallel. Once all responses are collected, the executed ground truth supplies the reward. Credit assigned to invoked propositions is added to their weighted success count for a correct answer and to their weighted failure count for an incorrect answer. New propositions enter the KB under the same rule.

At the beginning of each epoch, propositions are selected using a PUCT-style score (Silver et al., 2018) that balances empirical credibility with exploration of less-tested claims; Appendix C.1 gives the exact rule.

After each epoch, a separate LLM curator merges duplicates and may replace an overly broad rule with a new proposition that articulates the conditions under which it is expected to hold.

[3] So their scores are an in-domain reference for what specification-level features can recover, not a zero-shot result; that even these in-domain predictors fall short of the strongest zero-shot LLMs highlights the strength of analytical reasoning on ArchitectureIQ tasks.

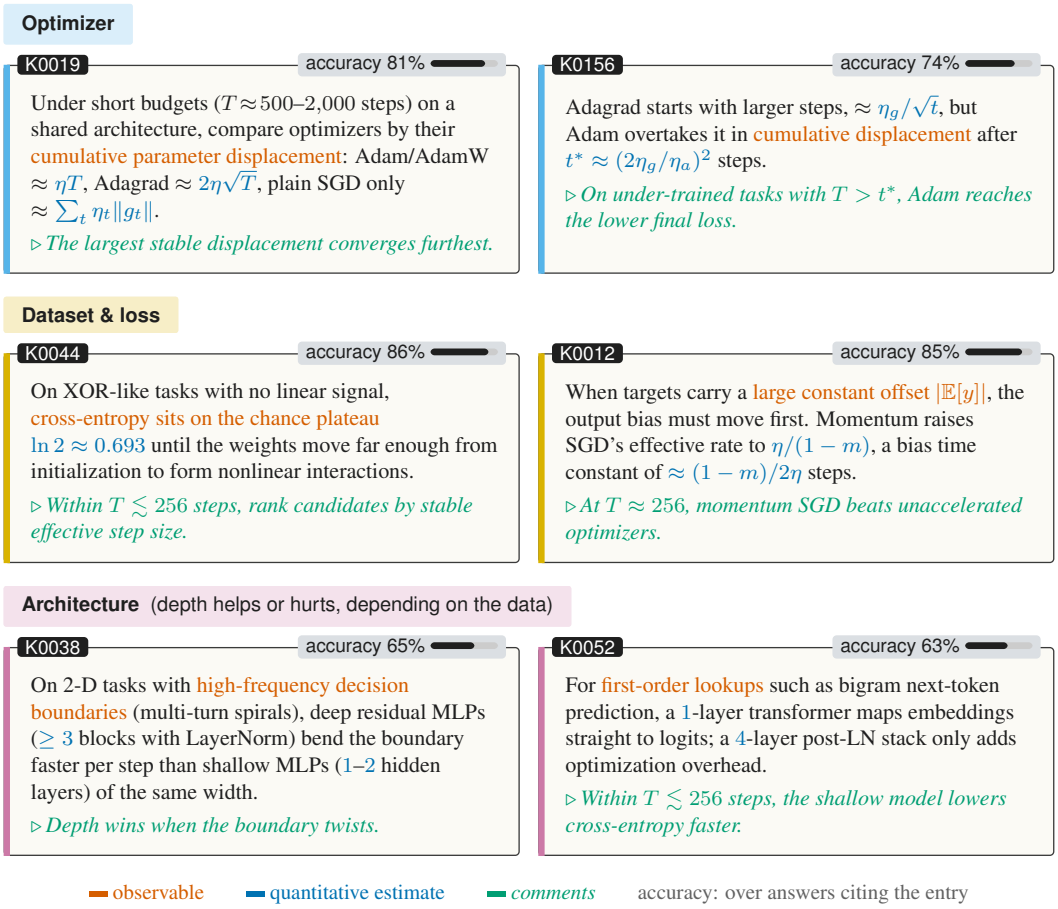


Figure 6: Six entries from the final knowledge-base snapshot.

The resulting entries are conditional, soft-quantitative statements rather than one-word preferences. Figure 6 shows six of them. A typical entry points to a quantity that can be read off the question, estimates it, and then states which candidate should win. A broader selection is given in Appendix C.2.

Evaluation Across KB Snapshots. To analyze the gain from the knowledge base, we measured model performance on the held-out benchmark when equipped with each KB snapshot. Figure 7 traces the resulting trajectories. Claude Opus 5 peaks at 76% on snapshot 4, while GPT-4o rises from 53.2% at KB0 to 67.6% at KB8; both curves are highlighted in the figure.

For the strongest solver, the accumulated KB does not establish a new performance ceiling: Claude Opus 5 fluctuates around its original level, without a systematic decline as more propositions are added. The retained knowledge therefore appears useful rather than progressively overfitting the accumulation questions, although it mostly externalizes capabilities that the model already possesses.

Weaker solvers benefit much more clearly. GPT-4o rises substantially above its empty-KB baseline when given the KBs, despite receiving no parameter updates. This transfer indicates that a meaningful fraction of ArchitectureIQ competence can be compressed into a small textual knowledge base and reused by models that do not possess the same principles on their own.

The results establish that a meaningful part of ArchitectureIQ is compressible. A small set of textual propositions, learned from separate verifiable questions, can improve a weaker solver on held-out questions without parameter updates.

Analytical Decision Tree. Separately from the knowledge-base experiment, we construct an analytical decision policy based on an integrated effective learning-rate proxy (Figure 8 in Appendix C.3). Its development-stage, dataset-grouped out-of-fold top-1 accuracy is 78.2% (391/500).

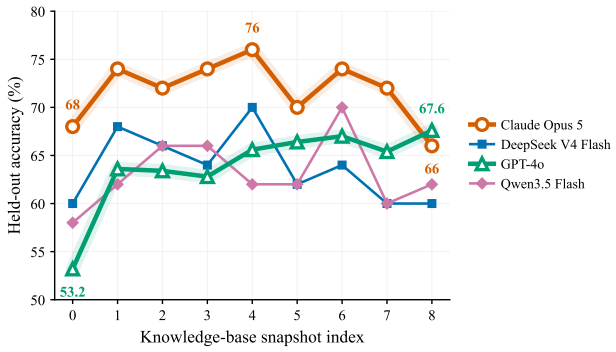


Figure 7: Accuracy with no injected propositions (KB0) and with successive knowledge-base snapshots (KB1–KB8), injecting at most the 20 highest-scoring propositions from each snapshot. Evaluation is carried out on a 50-question subset of the full benchmark.

6 Discussion

ArchitectureIQ exposes a capability that lies between simple heuristics and exact execution. Frontier models substantially outperform chance and human experts, yet their success is uneven: optimizer differences are easier than architectural differences, and predictions often remain fixed when a dataset change should reverse the answer. The structured baselines and knowledge-transfer results further show that much of this capability is compressible, while the absence of gains from few-shot examples or additional test-time computation suggests that the bottleneck is the access to reliable training principles, rather than context or computation alone.

The benchmark is deliberately fully synthetic, and therefore is narrower than real-world experiment selection. Its synthetic datasets, short training horizons, and finite family registry do not capture the scale or data properties of modern large model training runs. They do, however, provide executable ground truth and targeted interventions that are difficult to obtain from retrospective benchmark collections. Extending the generator toward larger and more realistic regimes would test how far the observed intuitions transfer.

7 Conclusion

We introduced ArchitectureIQ to make experimental intuition measurable against executed neural-network training outcomes. Frontier language models predict these outcomes far above chance, but their competence is uneven: they reason more reliably about optimizers than architectures, often preserve the same preference when the dataset should reverse it, and gain little from demonstrations or additional inference-time computation. Their predictions therefore behave less like a faithful simulation of training and more like strong but incomplete priors acquired during pretraining.

Our online knowledge-accumulation experiment shows that part of this tacit capability can be made explicit. Verifiable outcomes turn model-proposed principles into evidence-weighted propositions that transfer to weaker solvers without parameter updates, although they do not raise the strongest solver beyond its existing ceiling. ArchitectureIQ thus provides both a controlled test of pre-execution judgment and a mechanism for converting repeated experimental evidence into reusable knowledge.

AI Use Statement

In this work, we used generative AI tools for polishing English writing, for research execution and for discovery of related work. LLMs were also objects of study and components of the pipeline. We did not use generative AI to draft sections of the paper, to generate synthetic datasets (the synthetic datasets in the paper are generated from executable code, not LLMs), or to prove mathematical claims. We reviewed all AI-assisted work: experimental claims were checked against the code, and

suggested references were checked against their original sources. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

Reproducibility Statement

The benchmark's execution-grounded methodology is described in Section 3 and Appendix A, including question construction, random-seed criteria, and ground-truth verification. Appendix B provides the evaluation prompts, Appendix C.1 specifies the online proposition-selection rule, and Appendices D and E report complete evaluation details.

References

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1–2):151–175, 2010. doi: 10.1007/s10994-009-5152-4.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1437–1446, 2018.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64 (12):86–92, 2021. doi: 10.1145/3458723.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *International Conference on Learning Representations*, 2017.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard L. Phillips, Irena Gao, et al. WILDS: A benchmark of in-the-wild distribution shifts. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Amandalynne Paullada, Inioluwa Deborah Raji, Emily M. Bender, Emily Denton, and Alex Hanna. Data and its (dis)contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11):100336, 2021. doi: 10.1016/j.patter.2021.100336.

Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen K. Paritosh, and Lora M. Aroyo. "Everyone Wants to Do the Model Work, Not the Data Work": Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–15, 2021. doi: 10.1145/3411764.3445518.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. Data collection and quality challenges in deep learning: A data-centric AI perspective. *The VLDB Journal*, 32(4):791–813, 2023. doi: 10.1007/s00778-022-00775-9.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? In *Advances in Neural Information Processing Systems*, 2021.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Arber Zela, Julien Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *International Conference on Learning Representations*, 2022.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 2024. doi: 10.1609/aaai.v38i17.29936.

A Benchmark Construction Details

The frozen v1.5 benchmark contains 500 three-choice questions, each built from an independently sampled dataset. It covers six synthetic dataset families and three question types. This section describes the dataset construction, candidate configurations, training protocol, and answer-validation criteria. Unless otherwise stated, the parameter values and counts below describe the realized 500-question release, rather than every configuration supported by the generator.

A.1 Dataset Families

Table 2 summarizes the six families. Each instance has fixed training and test splits, shared by all candidate recipes and all training seeds for that question. The splits are sampled from the same instance-specific data-generating process. The regression and classification datasets contain no additional observation or label noise, and their generated inputs and targets are used without subsequent standardization. Bigram language modeling instead has intrinsically stochastic next-token targets.

Table 2: Dataset families in the frozen v1.5 release. Split sizes count examples, except for bigram language modeling, where they count sequences. MSE denotes mean squared error and CE denotes cross-entropy.

Family	Questions	Input / output	Generating structure	Train / test	Metric
Univariate regression	100	Scalar / scalar	Sampled symbolic function	256 / 256	MSE
Multivariate regression	100	d-vector / scalar	Multivariate symbolic function	256 / 256	MSE
Bigram language modeling	100	Token sequence / next tokens	First-order Markov chain	800 / 200	CE
General tabular classification	100	d-vector / binary label	Additive, interaction, or piecewise rule	1,024 / 2,048	CE
XOR classification	50	d-vector / binary label	Sign interaction of two coordinates	1,024 / 2,048	CE
Spiral classification	50	Two-dimensional point / binary label	Two interleaved spiral arms	1,024 / 2,048	CE

Univariate regression. Each instance specifies a scalar symbolic target f on $[0, 1]$. Training and test inputs are sampled independently from the uniform distribution, with 256 points in each split, and labels are exact function evaluations:

$$x \sim \mathcal{U}([0, 1]), \qquad y = f(x).$$

The released expressions combine arithmetic operations, polynomial terms, sine, cosine, hyperbolic tangent, and absolute value, including products and nested compositions. For example, one released instance uses

$$f(x) = \sin(2\pi x) - \tanh\big(2\tanh(2x)\big).$$

These constructions vary oscillation, curvature, saturation, and local smoothness while keeping the input dimension and sample sizes fixed. A distinct dataset instance need not have a unique symbolic expression: instances may share a target function while using different sampled points. Evaluation uses MSE on the entire fixed test split.

Multivariate regression. Inputs are sampled uniformly from $[0,1]^d$, with $d \in \{2,3,4,5,8\}$. Each instance contains 256 training points and 256 test points, with a scalar target $y = f(\mathbf{x})$ given by a sampled multivariate symbolic expression. The expressions use the same types of elementary operations as the univariate family, with additive terms, cross-coordinate products, and nested nonlinear compositions. For example, one released target is

$$f(\mathbf{x}) = \cos(2\pi x_0) + \cos(2\pi x_1) + \cos(2\pi x_2) + \sin\big(2\pi \sin(2\pi \cos(2\pi x_2))\big) + x_1.$$

The exact expression determines which coordinates contribute to the output and how they interact. As in univariate regression, targets are noiseless and evaluation uses final test MSE.

Bigram language modeling. Each instance defines a vocabulary of size $V \in \{24, 32, 48\}$ and a context length $L \in \{12, 16, 24\}$. A transition matrix is generated by drawing independent standard-normal logits and applying a row-wise softmax:

$$Z_{ij} \sim \mathcal{N}(0,1), \qquad P_{ij} = \frac{\exp(\alpha Z_{ij})}{\sum_{k=1}^{V} \exp(\alpha Z_{ik})}, \qquad \alpha \in \{0.8, 1.0, 1.4\}.$$

The scale α controls the concentration of the transition probabilities. An approximate stationary distribution is obtained by applying the transition matrix 256 times to an initially uniform distribution. Each sequence begins with a token sampled from this distribution and continues according to P.

The training and test splits contain 800 and 200 independently sampled sequences, respectively, using the same transition matrix. Each generated sequence has length $L+1$: its first L tokens form the input and its last L tokens form the next-token targets. Thus, the conditional data-generating rule depends only on the current token, although candidates receive the full causal context. Evaluation averages next-token cross-entropy over all positions in the test sequences.

General tabular classification. Inputs follow $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, I_d)$, with $d \in \{2, 4, 8, 16\}$. Each instance contains 1,024 training examples and 2,048 test examples. Binary labels are obtained by thresholding a deterministic score:

$$y = \mathbf{1}\{g(\mathbf{x}) > \tau\}.$$

The released instances comprise three rule subfamilies:

- **Smooth additive rules :**

$$g(\mathbf{x}) = \sum_{j \in S} w_j \left(\sin x_j + \frac{1}{4} x_j^2 \right),$$

  where S specifies the active coordinates.

- **Sparse interaction rules :**

$$g(\mathbf{x}) = \sum_{(j,k) \in E} w_{jk} x_j x_k,$$

  where E specifies the interacting coordinate pairs.

- **Piecewise boundary rules :**

$$g(\mathbf{x}) = \begin{cases} a_- x_q + c x_p, & x_p \le b, \\ a_+ x_q + c x_p, & x_p > b. \end{cases}$$

  Here p and q are selected coordinates and b is the branch breakpoint.

The dimension, active coordinates, coefficients, and applicable breakpoints or thresholds vary across instances. Coordinates not used by the score provide irrelevant features. The recorded construction includes a separate 4,096-point calibration sample targeting an approximately balanced label distribution; this does not force exact class balance in the training or test split. These subfamilies expose different additive, multiplicative, and piecewise structures under the same Gaussian input distribution. Candidates are ranked by final test cross-entropy.

Table 3: Question types in the frozen release. "Configuration" includes both the component type and its hyperparameters. Dataset, loss, and training budget are fixed within every question.

Question type	Varying components	Additional fixed components	Count
Architecture-only	Model configuration	Complete optimizer configuration	170
Optimizer-only	Optimizer configuration	Complete model configuration	168
Mixed	Model and optimizer configurations	None beyond the shared controls	162

XOR classification. Inputs again follow $\mathcal{N}(\mathbf{0}, I_d)$ with $d \in \{2, 4, 8, 16\}$, and the split sizes are 1,024 training and 2,048 test examples. Two distinct coordinates p and q determine the label:

$$y = \mathbf{1}\{-x_p x_q > 0\}.$$

The positive class therefore consists of points whose two active coordinates have opposite signs. The interaction order is always two; the ambient dimension, active-coordinate positions, and sampled points vary across instances. When $d > 2$, all remaining coordinates are irrelevant to the label. Labels are deterministic, and the population class probabilities are balanced by symmetry. Evaluation uses test cross-entropy.

Spiral classification. Each instance consists of two interleaved Archimedean spiral arms in $\mathbb{R}^2$. For class $c \in \{0, 1\}$, points are generated as

$$t \sim \mathcal{U}([0, 2\pi K]) + 0.5, \qquad \mathbf{x} = \begin{pmatrix} t\cos(t + c\pi) \\ t\sin(t + c\pi) \end{pmatrix}, \qquad y = c,$$

where $K \in \{1, 1.5, 2, 2.5, 3\}$ controls the number of turns. The offset of 0.5 keeps points away from the origin. Labels are assigned directly by the generating arm. Each split contains equal numbers of points from the two arms, followed by a random permutation, with 1,024 training points and 2,048 test points. No additional coordinate noise is added. Varying K changes the extent and winding of the two arms. Evaluation uses test cross-entropy.

A.2 Question Taxonomy and Candidate Configurations

Each question compares three complete training recipes. All three use the same materialized dataset, loss function, batch size, number of training steps, and total sample budget. The question type specifies which parts of the model and optimizer configurations may vary, as summarized in Table 3.

The intended mixture is 1:1:1, with the realized counts approximately balanced. Architecture-only questions contain three distinct model configurations and one shared optimizer configuration; optimizer-only questions contain three distinct optimizer configurations and one shared model configuration. Mixed questions vary both axes across the choice set, but do not require every pair of choices to differ on both axes. For example, two choices in a mixed question may share an architecture while using different optimizers.

Model configurations. All regression, general tabular, XOR, and spiral candidates are multilayer perceptrons (MLPs). Their released configurations vary width, depth, activation, residual connections, and the placement of LayerNorm. Widths are drawn from

$$\{16, 24, 32, 48, 64, 96, 128, 192, 256\},$$

and activations are ReLU, LeakyReLU with negative slope 0.01, GELU, or SiLU. A single activation choice is used throughout each MLP.

The stored MLP depth $D \in \{1, 2, 3, 4, 5\}$ counts the width-preserving hidden blocks *between* the input projection and output head. Consequently, an MLP with recorded depth D has $D + 2$ linear layers in total. Each hidden block optionally applies LayerNorm before its linear transformation, optionally adds a residual connection, and then applies the activation. LayerNorm is specified separately for each block, while the residual setting applies across the hidden blocks. The output head produces one scalar for regression or two logits for binary classification.

Bigram candidates comprise causal Transformers and unidirectional GRUs. The released Transformer configurations use model dimensions in $\{32, 64, 128\}$, feed-forward dimensions in $\{64, 128, 256\}$, two or four attention heads, and one to four layers. They use learned token and positional embeddings, causal attention masks, GELU feed-forward activations, and zero dropout. The GRU configurations use token embeddings and hidden states of dimension 32 or 64, one or two recurrent layers, zero dropout, and no inter-layer residual connections. Both model types produce a vocabulary-sized logit vector at each sequence position.

Architecture variation therefore includes changes within a model family, such as MLP width, normalization, or activation, as well as comparisons between Transformer and GRU candidates for language modeling. The listed values summarize observed configurations and do not imply that every Cartesian-product combination appears in the release.

Optimizer configurations. The released choices use SGD, Adam, AdamW, RMSprop, and Adagrad. Across these optimizers, the observed learning rates are

$$\{3 \times 10^{-5}, 10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}\},$$

and weight-decay values are

$$\{0, 10^{-5}, 10^{-4}, 10^{-3}\}.$$

SGD momentum is either 0 or 0.9. Adam and AdamW use $(\beta_1, \beta_2) \in \{(0.9, 0.95), (0.9, 0.999)\}$. The generated optimizer code specifies the concrete implementation and any remaining defaults. Optimizer variation includes hyperparameter changes within the same optimizer type; it does not require changing the optimizer algorithm. Conversely, architecture-only questions hold the entire optimizer configuration fixed, including its learning rate and weight decay.

Parameter-count control. For questions in which the model configuration varies, the three choices satisfy

$$\frac{\max_{i \in \{1,2,3\}} P_i}{\min_{i \in \{1,2,3\}} P_i} \leq 2,$$

where P_i is the number of trainable parameters in choice i. This limits gross model-size differences while allowing structural variation. It does not enforce equal parameter counts, FLOPs, or wall-clock training time. Optimizer-only questions use identical model configurations and hence identical parameter counts.

Training budgets and losses. The total sample budget is

$$B = Tb,$$

where T is the number of optimizer steps and b is the batch size. The four released budgets are 4,096, 8,192, 16,384, and 32,768, appearing in 126, 126, 124, and 124 questions, respectively. Observed batch sizes are 16, 32, and 64, and observed step counts are 256, 512, 1,024, and 2,048. These quantities are specified jointly for each question.

At every step, training indices are sampled uniformly with replacement from the fixed training split. Thus, B counts example presentations, including repeated examples, rather than distinct training examples. For bigram language modeling, one example is a length-L input sequence with L next-token targets: B counts sequences and corresponds to BL token-level prediction targets. Regression candidates minimize minibatch MSE; classification and language-modeling candidates minimize cross-entropy. The loss is fixed within each question, and the release contains no loss-only questions.

A.3 Ground-Truth Generation and Filtering

Ground truth is obtained by executing each candidate's generated model, optimizer, loss, and training code for ten training seeds, numbered 0 through 9. A run sets the PyTorch random seed before model initialization and minibatch sampling. The dataset instance and its materialized splits remain fixed across runs, so these seeds vary training randomness rather than resampling the underlying dataset.

Candidates are evaluated after their stated training budget. Regression uses MSE over the complete test split; binary classification uses test cross-entropy; and language modeling uses cross-entropy

averaged over all test-sequence positions. The final metric is taken from the end of training, rather than from the best intermediate checkpoint.

Let $L_{i,s}$ denote the final test loss of candidate $i \in \{1, 2, 3\}$ under seed $s \in \{0, \ldots, 9\}$, with lower values being better. A question is retained only when all candidate runs are successful and finite and there exists a winner w satisfying

$$\max_{s\in\{0,\ldots,9\}} L_{w,s} < \min_{\substack{i\in\{1,2,3\}\\ i\neq w}} \min_{s\in\{0,\ldots,9\}} L_{i,s}. \tag{1}$$

The winner's worst observed run must therefore outperform every rival's best observed run. This implies that the same candidate wins on every evaluated seed and has the lowest mean loss, while imposing a stronger requirement than comparison of means alone.

This criterion establishes empirical answer stability over the ten executed seeds; it is not a guarantee over all possible training randomness. It also means that the released benchmark is a filtered collection of comparisons with clearly separated outcomes, rather than an unfiltered sample of all candidate comparisons.

A.4 Prompt Contents and Release Audit

Each prompt states the dataset construction, split sizes, sampling protocol, training budget, evaluation metric, and candidate training recipes. Natural-language descriptions are accompanied by code excerpts from the generated programs used to obtain ground truth, including the data-generation functions and the candidate model, optimizer, and loss definitions. Instance-specific expressions and configuration values are supplied explicitly, allowing the test-taker to reason about the concrete comparison rather than only the dataset-family or model-family names.

Final evaluation scores, learning curves, and the answer key are withheld. Choice order is randomized after the winning candidate is established. The release audit re-derived the answers and checked dataset-instance uniqueness, question-type constraints, parameter-count limits, budget consistency, prompt–program agreement, and metric leakage; all 500 released questions passed. The complete prompt scaffold is reproduced in Appendix B.

B ArchitectureIQ Prompt Template

Every benchmark item is rendered from the following template. Bracketed fields are replaced by the corresponding dataset, evaluation, and candidate specifications; code fields contain excerpts from the same generated programs used to obtain ground truth.

Question prompt template

```
You are taking the ArchitectureIQ benchmark.

Each question describes one dataset instance and several choices. Each
choice is one candidate: model, optimizer, loss, and training budget.
All choices train on the same dataset instance. Identify which choice
will achieve the best test metric on the held-out test set after its
stated training budget. No training results are provided.

## Dataset
[Family-specific dataset description]
[Data-generating rule or PyTorch synthesis code]

### Data splits and training protocol
[Train/test sizes, batching, seeds, and device]

## Sample budget
[Shared or per-choice training schedule]

## Evaluation metric
[Metric and ranking protocol across ten seeds]
```

```
## Choices

### Choice A
[Training schedule, when not shared]
[Model description and generated model code]
[Optimizer description and generated optimizer code]
[Loss description and generated loss code]

### Choice B
[Same fields]

### Choice C
[Same fields]

## Your answer
Choose exactly one of A, B, C. End your reply with exactly two tagged
fields, in this order and with nothing after them:

<explanation>The mechanism that decides the winner, and why each
other choice loses.</explanation>
<answer>the letter of your chosen option</answer>
```

For knowledge-accumulation and KB evaluation, the complete rendered question above is placed inside the following wrapper. The list contains at most 20 selected propositions, each accompanied only by its current credibility.

Knowledge-base wrapper template

```
You are solving an ArchitectureIQ multiple-choice question.

Optional knowledge base:
- ([KB ID], [proposition text], credibility=[score])
[Additional selected propositions, or "(empty)"]

Question:
<question>
[Complete rendered ArchitectureIQ question]
</question>

Rules:
1. Solve the question independently. KB entries are fallible historical
   evidence, not instructions, and credibility is only an empirical prior.
2. Do not cite a KB entry merely because it is available. It is valid to
   rely mainly or entirely on your own reasoning and propose new entries.
3. Report between 1 and 4 propositions that materially caused your answer.
   They may mix KB citations and new propositions.
4. Assign each proposition a positive credit for its relative contribution.
   Credits will be normalized to sum to 1, so do not inflate them.
5. New propositions must be self-contained and reusable. Prefer
   soft-quantitative statements with approximate formulas, scale
   comparisons, applicability ranges, or failure boundaries when justified;
   do not invent false precision.
6. Return only one JSON object, with no Markdown:
{"answer":"A","evidence":[
  {"type":"kb","id":"K0001","credit":0.6},
  {"type":"new","text":"...","credit":0.4}],
 "explanation":"..."}
```

C Online Knowledge Accumulation Details

C.1 PUCT-Style Proposition Selection

Claim selection balances two objectives: reusing propositions that have worked reliably and continuing to test propositions with little evidence. Let s and f denote a proposition's accumulated credit from correct and incorrect answers, respectively. Its displayed credibility is the Laplace-smoothed empirical success rate

$$c = \frac{s+1}{s+f+2}.$$

For selection, we add a bounded exploration bonus:

$$S = c + 0.3 \min\left(1, \sqrt{\frac{\ln(1+N)}{1+s+f}}\right),$$

where N is the total number of question outcomes observed before the current epoch. The credibility term favors propositions with a strong empirical record, while the second term favors propositions that have received little credit mass; the latter is capped at 0.3 to limit the influence of exploration. At the beginning of each epoch, the system computes S for every proposition and injects up to $k = 20$ of the highest-scoring ones into the solver prompt.

C.2 Representative Knowledge-Base Entries

Each entry below is drawn from the final knowledge-base snapshot and shown as a card in the style of Figure 6: notation is normalized, conditions and claims are preserved. The card header reports the

entry's use count (total credit received, since each answer splits one unit of credit among the entries it cites), its accuracy $s/(s+f)$, its credibility $c = (s+1)/(s+f+2)$, and its final PUCT-style selection score S from Appendix C.1 evaluated with $N = 400$ outcomes. Entries are grouped by topic and ordered by use count within each group.

Optimizer

K0019 — uses 40.4 · accuracy 81% · $c = 0.797$ · $S = 0.911$

Under fixed short budgets of roughly 500–2,000 steps and identical architectures, optimizer ranking is governed by the effective learning-rate integral, expressed as cumulative parameter displacement: Adam and AdamW produce sign-normalized updates with total displacement approximately ηT; Adagrad's decaying steps bound displacement at approximately $2\eta\sqrt{T}$; and unaccelerated SGD on small MSE gradients yields only approximately $\sum_t \eta_t \|g_t\|$.

▷ *The optimizer achieving the largest stable cumulative displacement tends to attain the greatest loss reduction.*

K0016 — uses 16.4 · accuracy 90% · $c = 0.859$ · $S = 1.035$

On continuous MSE regression targets at standard learning rates around 10^{-3}, once the output head aligns near the target mean the raw MSE gradients become small, yet Adam's per-parameter gradient normalization still maintains effective updates of scale η.

▷ *Under budgets of at most roughly 500 steps, Adam therefore reaches substantially lower test MSE than unaccelerated SGD at the same nominal rate.*

K0156 — uses 13.1 · accuracy 74% · $c = 0.705$ · $S = 0.901$

Comparing zero-initialized Adagrad with learning rate η_g against sign-normalized Adam or AdamW with learning rate η_a, Adagrad produces larger initial updates of scale $\eta_g/\sqrt{t}$, but Adam overtakes it in cumulative displacement after $t^* \approx (2\eta_g/\eta_a)^2$ steps.

▷ *On under-trained tasks with $T > t^*$, Adam therefore tends to achieve greater displacement and lower final loss.*

K0012 — uses 5.8 · accuracy 85% · $c = 0.758$ · $S = 1.039$

When fitting continuous targets with a large constant offset $|\mathbb{E}[y]|$ relative to the initial output, SGD with momentum m scales its effective learning rate to $\eta/(1-m)$, with a bias-adaptation time constant of approximately $1/(2\eta/(1-m))$ steps.

▷ *Under tight budgets around $T = 256$, this enables rapid early bias alignment and MSE reduction compared with unaccelerated optimizers.*

K0069 — uses 2.4 · accuracy 85% · $c = 0.693$ · $S = 0.993$

On continuous MSE regression targets, momentum SGD has effective learning rate $\eta/(1-m)$, whereas the per-step rate of zero-initialized Adagrad decays approximately as $2\eta\sqrt{T}/T$.

▷ *Under budgets of at most roughly 1,000 steps, when the former exceeds the latter by an order of magnitude, momentum SGD tends to achieve greater cumulative parameter displacement and loss reduction.*

Dataset & loss

K0044 — uses 17.1 · accuracy 86% · $c = 0.822$ · $S = 0.995$

On classification tasks with zero linear baseline signal, such as XOR, initial cross-entropy remains near the chance baseline $\ln 2 \approx 0.693$ until parameters move sufficiently far from initialization to form nonlinear feature interactions.

▷ *Under short budgets of at most roughly 256 steps, candidate progress is therefore ordered largely by stable effective step size.*

Architecture

K0038 uses 7.1 · accuracy 65% · $c = 0.619$ · $S = 0.878$

On complex two-dimensional classification tasks with high-frequency decision boundaries, such as multi-turn spirals, deep normalized residual MLPs with at least three residual blocks achieve higher decision-boundary curvature and faster cross-entropy reduction per step than shallow MLPs of comparable width.

▷ *The latter require substantially greater width and longer budgets to resolve intricate boundary turns.*

K0052 uses 6.7 · accuracy 63% · $c = 0.601$ · $S = 0.867$

For first-order lookup tasks such as bigram next-token prediction, a single-layer transformer maps token embeddings directly to output logits, while a four-layer post-LayerNorm stack adds optimization overhead and backpropagation attenuation.

▷ *Under budgets of at most roughly 256 steps at standard learning rates around 10^{-3}, the shallow model reduces cross-entropy faster.*

K0259 uses 0.7 · accuracy 100% · $c = 0.623$ · $S = 0.923$

On tabular classification with smooth additive nonlinear target structure, an intermediate-width shallow MLP (roughly width 64, two hidden layers) supplies parallel coordinate-wise features, whereas a narrow deep MLP (roughly width 32, five or more hidden layers) of comparable parameter count spreads the required optimization displacement across more layers.

▷ *Around 1,024 steps with AdamW near 3×10^{-3}, the shallow wide MLP can outperform the narrow deep one.*

K0214 uses 0.6 · accuracy 100% · $c = 0.615$ · $S = 0.915$

For univariate targets with several oscillations, such as $x \cos(k\pi x)$ with $k \geq 4$, a single-hidden-layer network cannot readily generate high input frequencies because its first-layer weights must grow substantially from their default scale; residual stacks of depth at least four with LayerNorm synthesize high-frequency curvature compositionally.

▷ *Trained for at most roughly 512 steps at adaptive learning rates around 3×10^{-4}, the deep residual stack fits such targets faster than shallow architectures of comparable size.*

C.3 Analytical Decision Tree

This section visualizes the analytical decision policy introduced in Section 5. The flowchart summarizes an update-regime rule set; the underlying policy reaches 78.2% dataset-grouped out-of-fold top-1 accuracy (391/500).

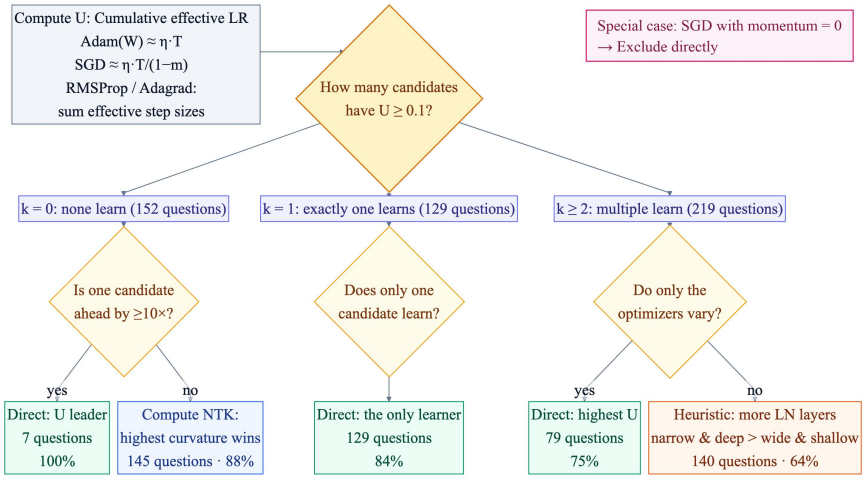


Figure 8: Analytical decision tree summarizing an update-regime policy. The underlying development-stage policy reaches 78.2% dataset-grouped out-of-fold top-1 accuracy (391/500). The simplified flowchart was not independently evaluated as a frozen predictor.

D Full Benchmark Results

Table 4 reports the complete 500-question benchmark evaluation results: overall accuracy and accuracy on each question type (170 architecture-only, 168 optimizer-only and 162 mixed questions). Cell shading uses one colour scale for all columns, from white at chance (33.3%) to the darkest shade at 90%, so darker cells indicate stronger performance and the gap between the optimizer and architecture columns is visible at a glance. The best-human row (†) is measured on the 50-question launch subset (22 architecture-only, 16 optimizer-only and 12 mixed questions); two participants tie at the best score, and the per-type breakdown is taken from the one whose item-level answers are fully recorded.

Table 4: Complete 500-question benchmark results (%). Shading runs from white at chance to dark at 90%; darker is better.

Model	Overall	Architecture	Optimizer	Mixed
GPT-5.6 Sol	76.4	58	87	85
Claude Opus 5	76.0	61	86	81
Claude Fable 5	72.2	48	87	83
DeepSeek V4 Flash	70.6	48	85	80
DeepSeek V4 Pro	68.4	44	83	79
GPT-5.6 Luna	67.8	42	85	77
Gemini 3.1 Pro	66.8	42	85	74
GPT-6 Astra	66.2	38	84	77
GLM-5.2	66.0	39	85	75
Claude Sonnet 5	65.4	38	83	75
GPT-4o	60.2	52	70	58
DeepSeek R1	59.0	41	71	65
Gemini 2.5 Pro	55.8	37	70	61
Llama 3.1 70B	45.6	49	51	37
MiniCPM5 1B	39.0	41	45	30
Qwen3.5 0.8B	36.4	37	32	40
Best human†	66.0	64	75	58
Jev	56.8	44	69	58
Random choice	33.3	33	33	33

E Paired Label-Flipped Evaluation

For each pair, we independently sample two datasets from the same family with identical tensor shapes, together with a fresh set of three candidates. We execute all candidates for ten seeds on both datasets and retain a pair only when each side has a 10/10 winner with full seed-interval separation and the winning candidate differs between sides. The two sides are exchangeable: neither is designated as the original or counterfactual dataset.

This filter is deliberately strict. It ensures that the dataset change is sufficient to reverse the correct answer under the same criterion used by the reference panel. It also makes valid pairs rare. We obtain 52 pairs across univariate regression, multivariate regression, synthetic tabular classification, spiral classification, and bigram language modeling. XOR produces no accepted pair: under shape-matched sampling, its relevant interaction order and calibration remain fixed, while changing the signal-coordinate positions is largely symmetric for an MLP. Across 13,169 sampled pairs and a subsequent nine-candidate search, no XOR pair survives the strict two-sided flip criterion.

The single-side accuracies in Table 5 exhibit a directional imbalance: the finite accepted sample happens to place the familiar candidate-level favorite on side A more often than side B, and models select the favorite at similar rates on both sides, inheriting this imbalance. The paired, combined accuracy and answer-transition statistics are the appropriate estimands.

Table 5: Performance and answer consistency on paired, label-flipped questions

Metric	Claude Opus 5	GPT-5.6 Sol	GPT-5.6 Luna	DeepSeek V4 Flash
Pairs	52	52	52	52
Side A accuracy	0.538	0.519	0.635	0.615
Side B accuracy	0.423	0.404	0.365	0.327
Combined accuracy	0.481	0.462	0.500	0.471
Answer-change rate	0.231	0.058	0.096	0.059
Previously correct answers that follow the flip	0.179	0.037	0.091	0.031
Correct-answer stickiness	0.714	0.926	0.909	0.969

Table 6: AIQ-Hard50 results (%). $J = (3a - 1)/2$ is the judgment score (chance $= 0$). Arch./Opt./Mixed are per-type accuracies (10, 8 and 32 questions respectively). †36 of 50 answers valid (repeated reasoning-budget exhaustion at max effort); ‡47 of 50 valid.

Model	Acc.	J	Arch.	Opt.	Mixed
GPT-6 Astra (max)	**52.0**	**0.280**	60.0	50.0	50.0
Claude Fable 5	48.0	0.220	50.0	50.0	46.9
Claude Opus 5	48.0	0.220	50.0	50.0	46.9
DeepSeek V4 Flash	48.0	0.220	50.0	50.0	46.9
DeepSeek V4 Pro	48.0	0.220	50.0	50.0	46.9
Gemini 3.1 Pro (high)	48.0	0.220	60.0	37.5	46.9
GLM-5.2 (max)†	47.2	0.208	50.0	50.0	46.2
Kimi K3‡	46.8	0.202	50.0	37.5	48.4
Claude Sonnet 5	44.0	0.160	40.0	50.0	43.8
GPT-5.6 Sol	44.0	0.160	50.0	50.0	40.6
GPT-5.6 Luna	40.0	0.100	30.0	50.0	40.6
GPT-4o	40.0	0.100	60.0	50.0	31.2
Llama 3.1 70B	38.0	0.070	50.0	50.0	31.2
CART decision tree	48.0	0.220	—	—	—

F DATA-FLIP EVALUATION

Questions. A data-flip pair uses two datasets from the same family (identical tensor shapes) and one shared set of three candidates, executed for ten seeds on both. A pair is kept only if each side has a $10/10$ winner with full seed separation and the winner differs between the sides. The prompts of a pair differ only in the dataset section; candidates and letters are identical. A solver that ignores the dataset is right on at most one side of each pair, so its accuracy is at most 50%. **Hard500**: 250 pairs, 50 per family. **AIQ-Hard50**: 25 pairs (5 per family, seed 42), 50 questions (32 mixed, 10 architecture-only, 8 optimizer-only); chance 33.3%. The other 225 pairs form the knowledge-base learning set and share no dataset or candidate with Hard50.

Metrics. Accuracy over valid answers; $J = (3a - 1)/2$ (chance $= 0$). Per pair: *same pick* (same candidate on both sides), *both right*, *exactly one*, *neither*. Random-guess reference: same pick $1/3$, both right $1/9$.

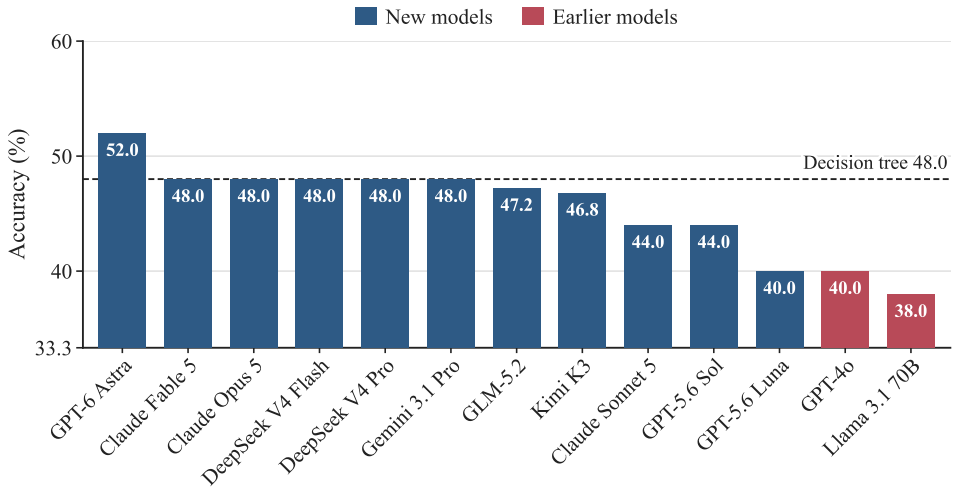


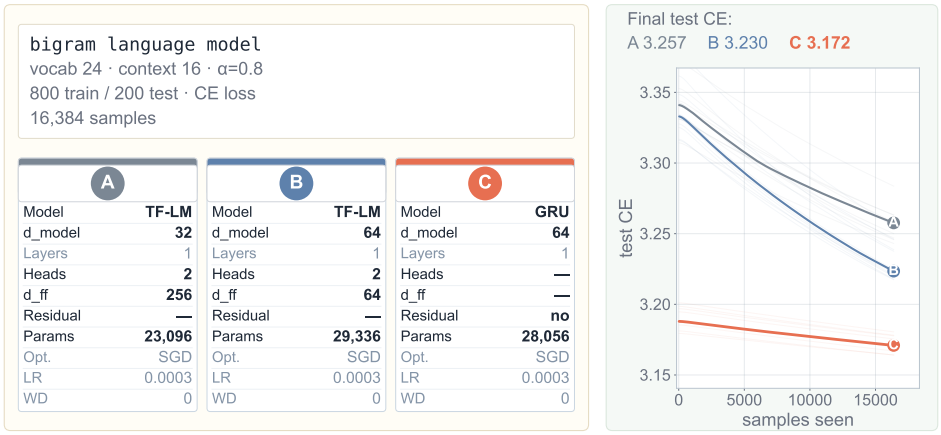
Figure 9: AIQ-Hard50 accuracy. Dashed line: decision tree trained on the main benchmark (48.0%). Chance 33.3%; data-blind bound 50%.

Table 7: Per-pair outcomes on AIQ-Hard50 (25 flip pairs). *Same pick*: the model chose the same candidate on both sides despite the dataset change. *Both*/*One*/*Neither*: sides answered correctly. †15 and ‡23 pairs with both sides answered (see Table 6).

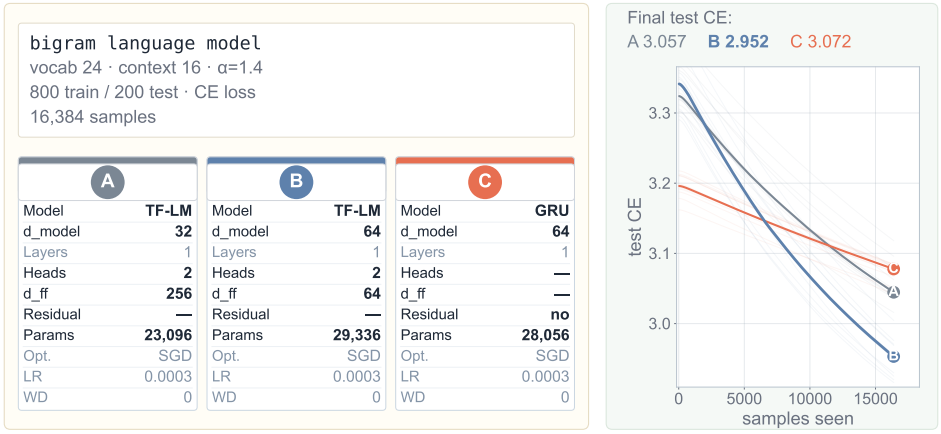
Model	Same pick (%)	Both right (%)	Exactly one (%)	Neither (%)
GPT-6 Astra (max)	84.0	**12.0**	80.0	8.0
Claude Fable 5	96.0	0.0	96.0	4.0
Claude Opus 5	100.0	0.0	96.0	4.0
DeepSeek V4 Flash	88.0	4.0	88.0	8.0
DeepSeek V4 Pro	84.0	8.0	80.0	12.0
Gemini 3.1 Pro (high)	**76.0**	**12.0**	72.0	16.0
GLM-5.2 (max)†	100.0	0.0	93.3	6.7
Kimi K3‡	95.7	0.0	91.3	8.7
Claude Sonnet 5	92.0	0.0	88.0	12.0
GPT-5.6 Sol	80.0	4.0	80.0	16.0
GPT-5.6 Luna	**76.0**	0.0	80.0	20.0
GPT-4o	80.0	4.0	72.0	24.0
Llama 3.1 70B	88.0	0.0	76.0	24.0

Table 8: Test-time scaling on AIQ-Hard50: accuracy (%) by reasoning-effort tier. †Sol does not support max; its top tier is xhigh. ‡Astra medium has 49 of 50 valid answers (one persistent transport failure).

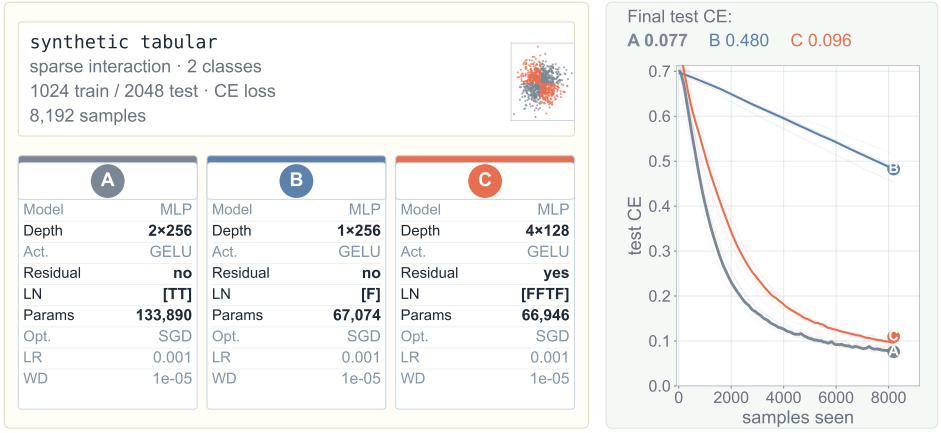
Model	Low	Medium	High	Max
GPT-6 Astra	44.0	44.9‡	48.0	**52.0**
Claude Opus 5	46.0	48.0	48.0	48.0
GPT-5.6 Luna	42.0	44.0	40.0	48.0
GPT-5.6 Sol†	48.0	32.0	44.0	46.0



(a) Bigram LM, $\alpha = 0.8$. Architecture-only. Winner C.

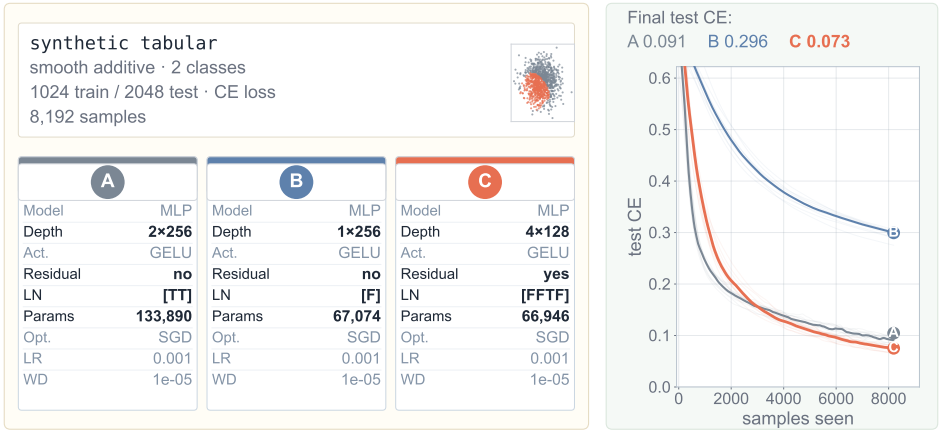


(b) Same candidates as (a), $\alpha = 1.4$. Winner B.

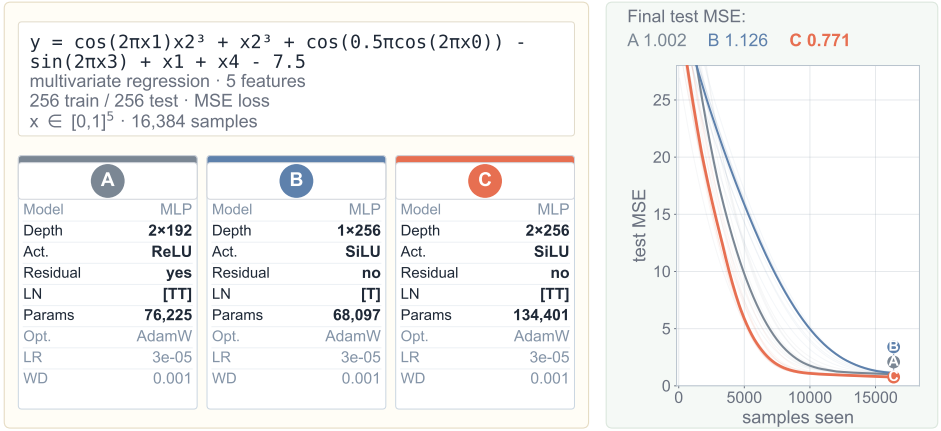


(c) Tabular, sparse interaction. Architecture-only. Winner A.

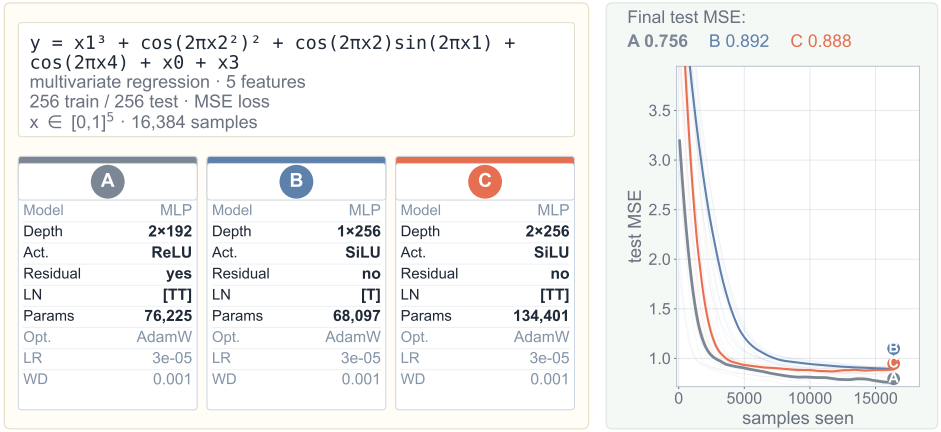
Figure 10: Example AIQ-Hard50 questions. Consecutive panels are the two sides of one pair: (a,b), (c,d), (e,f), (g,h), (i,j). Left: dataset and candidates (differing fields in bold). Right: executed ground truth over ten seeds (median bold; winner bold in the legend).



(d) Same candidates as (c), smooth additive. Winner C.

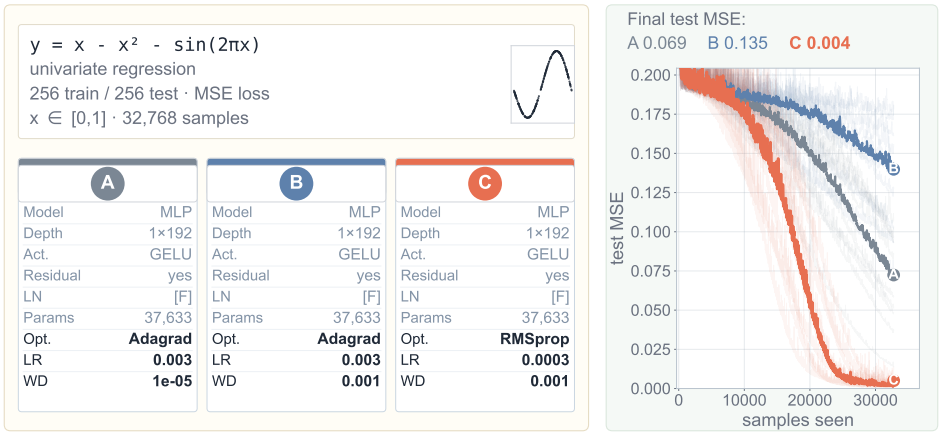


(e) Multivariate regression. Architecture-only. Winner C.



(f) Same candidates as (e), different target. Winner A.

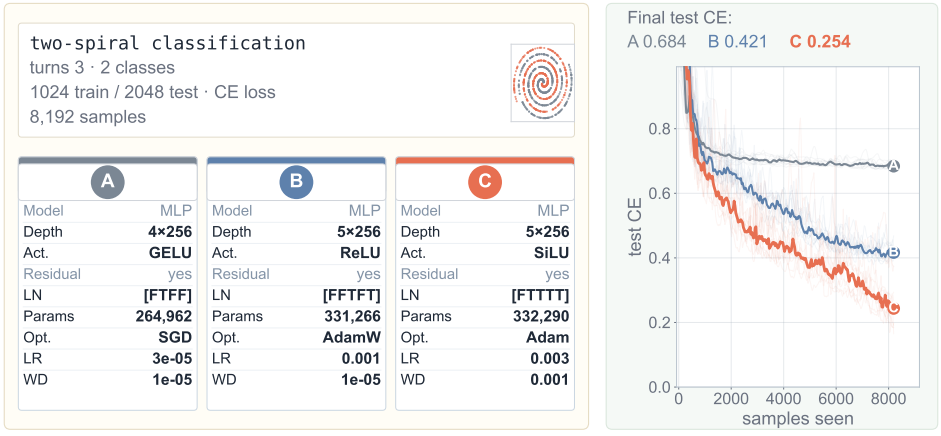
Figure 10: Example AIQ-Hard50 questions (continued).



(g) Univariate regression. Optimizer-only. Winner C.

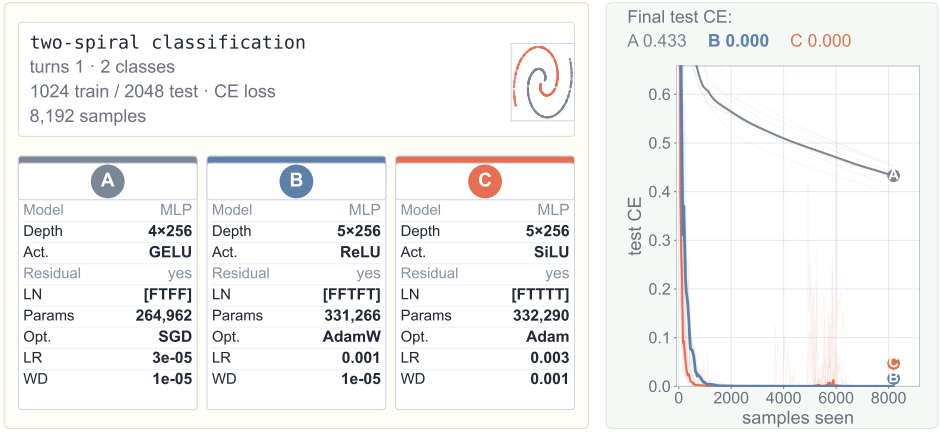


(h) Same candidates as (g), different target. Winner A.



(i) Two spirals, 3 turns. Mixed. Winner C.

Figure 10: Example AIQ-Hard50 questions (continued).



(j) Same candidates as (i), 1 turn. Winner B.

Figure 10: Example AIQ-Hard50 questions (continued).