

Verifier-Guided Adaptive Test-Time Scaling for Cost-Accurate Mathematical Reasoning

Jennifer Scott¹, Tao Chen², Brian Young³, Ling Zhao⁴, Eric Reed⁵

¹Department of Computer Science, University of Southern California, Los Angeles, CA, USA

²Department of Computer Science, University of California, Los Angeles, CA, USA

³Department of Computer Science, Cornell University, Ithaca, NY, USA

⁴Department of Computer Science, Princeton University, Princeton, NJ, USA

⁵Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

jennifer.scott82@yahoo.com

Abstract

Inference-time sampling improves mathematical reasoning only when additional candidates are useful and the selector can identify them. This paper presents Verifier-Guided Adaptive Test-Time Scaling (VATS), a cost-aware framework that allocates a nested candidate budget to each problem, ranks candidate answers with a learned verifier, calibrates checkpoint-level correctness, and stops once the selected answer is stable and sufficiently confident. The empirical study uses all 7,473 GSM8K training problems and all 1,319 test problems. A trace-conditioned program sampler retrieves arithmetic programs from the training rationales, transfers them to new number mentions, adds cue-ranked exact-arithmetic programs, and exposes candidate prefixes at budgets of 1, 2, 4, 8, 16, 32, and 64. A 32-feature histogram gradient-boosting verifier raises exact-match accuracy at the largest budget from 4.78% for proposal-score selection and 6.22% for answer voting to 8.19%. The difference over proposal selection is 3.41 percentage points (95% bootstrap interval 1.74 to 5.00; exact McNemar $p = 4.53e-5$). The development-selected adaptive policy reaches 5.76% accuracy with 6.14 actual candidate evaluations on average, saving 89.59% of the computation used by fixed $N = 64$. A descriptive high-accuracy Pareto point reaches 8.26% with 45.08 evaluations, preserving fixed-budget accuracy while saving 23.63% of measured candidate scoring. Oracle pass@64 is 43.29%, showing that candidate generation and answer selection remain separate bottlenecks. The results establish a transparent accuracy-compute frontier and show that calibrated stopping reduces unnecessary inference without concealing the limits of the underlying reasoning policy.

Keywords - *Mathematical Reasoning; Test-Time Scaling; Verifier; Adaptive Computation; Best-Of-N; Confidence Calibration; GSM8K; Cost-Aware Inference*

1. Introduction

Large language models derive much of their flexibility from transformer architectures and large-scale autoregressive training [19], [20]. Yet mathematical word problems expose a persistent distinction between fluent continuation and reliable multi-step computation. A model can recognize the relevant quantities, produce locally plausible prose, and still make an early semantic or arithmetic error that propagates to the final answer. This distinction makes grade-school mathematics a useful setting for studying inference policies: the final answer has an exact label, the reasoning path contains multiple decision points, and additional computation can be allocated through repeated sampling rather than parameter updates.

Chain-of-thought prompting established that intermediate reasoning steps substantially improve arithmetic, symbolic, and commonsense tasks [2]. Zero-shot prompting showed that a short reasoning instruction can elicit similar behavior

without demonstrations [4], while scratchpad training demonstrated that explicit intermediate state supports computations that are difficult to execute in a single forward pass [9]. Large-scale quantitative language models further show that model scale and mathematical pretraining improve reasoning breadth [5]. These methods convert a one-shot answer into a sequence of commitments. They do not, however, guarantee that the first sampled sequence is correct, so the inference policy must decide whether to accept, resample, compare, or continue.

Test-time scaling addresses this decision by drawing several reasoning paths. Self-consistency marginalizes over sampled chains and returns the most common answer [3]. The original GSM8K work instead trained a verifier to rank sampled solutions, showing that a learned correctness model can use additional candidates more effectively than the generator alone [1]. Process supervision further demonstrated that step-level judgments provide stronger learning signals than final-

answer labels on difficult mathematics [6], while outcome- and process-based feedback studies clarified how verification targets influence performance [7]. Together, these results motivate a separation between candidate generation, verification, and budget allocation. Calibration is also a selection concern: human-uncertainty distillation shows that soft label distributions preserve structured ambiguity that hard targets suppress [27], motivating explicit uncertainty modeling when sampled traces disagree.

A second line of work improves the structure of the candidate itself. Program of Thoughts and Program-Aided Language Models delegate arithmetic execution to an external interpreter [10], [11]. Least-to-most prompting decomposes a difficult request into simpler subproblems [12], and Tree of Thoughts expands reasoning into an explicit search process [13]. Auto-CoT selects diverse demonstrations automatically [16], whereas decomposed prompting organizes complex tasks as modular calls [17]. These methods show that structured traces and external computation reduce calculation errors, but they still require a policy for deciding how many traces to generate.

Iterative feedback provides another route to better inference. STaR bootstraps reasoning from successful self-generated rationales [8]. Self-Refine alternates generation, feedback, and correction [14], and Reflexion stores verbal feedback to improve later attempts [15]. These approaches spend extra computation after an initial response. Their benefit therefore depends on a basic economic question: which problems deserve more attempts, and when is the evidence strong enough to stop? A uniform budget ignores the large variation in problem difficulty and wastes work on cases that stabilize early. An analogous allocation problem appears in risk-aware, budget-constrained auto-bidding, which trades utility against spending and tail risk [28]; VATS substitutes candidate evaluations for monetary spend.

GSM8K was designed to diagnose multi-step grade-school reasoning and supplies answer rationales with executable arithmetic annotations [1]. The MATH benchmark broadens the difficulty range to competition mathematics [18], but GSM8K remains especially suitable for controlled cost studies because exact answers and annotated operations support detailed candidate and error analysis. The central objective here is not to claim competitive language-model accuracy. It is to measure, on the complete GSM8K test set, how verification and adaptive allocation behave when candidate generation, selection, and computation are logged exactly.

Adaptive computation has long treated inference depth as a learned decision. Adaptive Computation Time assigns variable recurrent steps [23], and PonderNet learns a halting distribution that balances accuracy and cost [24]. Sequential analysis supplies a statistical foundation for stopping when evidence is sufficient [25]. In a verifier-guided system, that evidence must also be calibrated: confidence should correspond to empirical correctness rather than raw score

magnitude. Prior work on neural calibration [21], supervised probability estimation [22], and Platt-style probability mapping [26] provides the tools used here to convert checkpoint state into a stopping probability.

This study contributes four linked results. First, it constructs a fully measured test-time scaling pipeline in which every GSM8K test problem receives a nested sequence of exact-arithmetic candidates. Second, it trains a candidate verifier on a disjoint development partition and a checkpoint calibrator on a separate policy partition. Third, it compares proposal-score best-of-N, answer voting, verifier reranking, oracle pass@N, and adaptive stopping across seven budgets, with actual candidate counts rather than nominal caps. Fourth, it decomposes the remaining error into generation failure, verifier selection failure, and premature stopping, establishing where further compute produces value and where it does not.

2. Method

2.1 Task Definition and Data Partitioning

For a problem q , the candidate generator produces an ordered sequence $C(q) = (c_1, \dots, c_M)$, where each candidate contains an exact rational answer, an arithmetic operation signature, a source type, and proposal metadata. A selector observes the first n candidates and returns an answer a_n . Fixed-budget evaluation holds n at a checkpoint in $\{1, 2, 4, 8, 16, 32, 64\}$. Adaptive evaluation chooses a stopping checkpoint $N(q)$ from the same set. The optimization target is a pair: exact-match accuracy and the mean number of candidates actually evaluated.

The data consist of 7,473 GSM8K training records and 1,319 test records. Each record contains a natural-language question and an answer rationale whose final label follows the marker "####". Arithmetic spans enclosed by double angle brackets identify intermediate expressions. The training set is shuffled with seed 20240229 and partitioned into 5,979 generator-fit examples, 747 verifier-development examples, and 747 policy-development examples. The test set remains intact. Table 1 reports comparable length, numeric density, and annotated-step distributions across all partitions.

The generator-fit partition supplies retrieval templates only. The verifier-development partition supplies candidate-level correctness labels for fitting the verifier. The policy-development partition is divided internally: the first 300 problems tune answer-level support aggregation, the next 224 fit the checkpoint calibrator, and the final 223 choose the primary stopping threshold. Test rationales are read only after prediction to compute exact match, difficulty groups, and error categories. This separation prevents test answers from influencing candidate order, verifier parameters, or the development-selected stopping rule.

Table 1. GSM8K partitions and observed problem characteristics.

Partition	n	Question words	Numeric mentions	Annotated steps	Median steps
Generator-fit	5979	45.06	4.30	3.17	3

Partition	n	Question words	Numeric mentions	Annotated steps	Median steps
Verifier-development	747	45.73	4.32	3.09	3
Policy-development	747	44.68	4.28	3.27	3
GSM8K test	1319	46.25	4.41	3.25	3

2.2 Trace-Conditioned Candidate Generation

Candidate generation begins with deterministic number parsing. The parser recognizes digits, comma-separated quantities, decimals, percentages, cardinal number words, simple fractions, dozen expressions, and multiplicative words such as "twice" and "triple." Each mention retains its exact Fraction value, text span, semantic kind, and a 76-character local context. Percent mentions retain both the displayed value and its normalized fraction. Ordinal mentions are recorded for template alignment but omitted from generic arithmetic enumeration because they frequently denote sequence position rather than quantity.

The arithmetic annotations in each generator-fit rationale are parsed into expression trees over addition, subtraction, multiplication, division, prior intermediate steps, question-number references, and constants. A valid template ends at the published final answer. At inference, term frequency-inverse document frequency vectors with word unigrams and bigrams retrieve 84 training questions. Each retrieved program is transferred through two deterministic mappings: an order-preserving proportional alignment and a context-greedy alignment based on lexical overlap, number kind, and relative position. Invalid divisions, excessive magnitudes, and denominators above one million are discarded. Retrieved templates remain traceable to their source rationale, following the evidence-grounding principle used in financial RAG systems [29] rather than treating retrieval as an unexamined answer oracle.

Retrieval is supplemented by a generic arithmetic grammar over the first seven non-ordinal number mentions. It emits identity candidates, all ordered two-number operations, cue-ranked three-number binary trees, all-number sums and products, first-minus-rest programs, averages, and unit conversions for weeks, years, minutes, and percentages. Operation cues come directly from the question: words such as "altogether" raise addition weight, "remaining" raises subtraction weight, "each" raises multiplication weight, and "equally" raises division weight. All computations use Python Fraction objects, so arithmetic execution introduces no floating-point error. This separation between finding a transferable program and interpreting it mirrors retrieval-summary integration in code intelligence [30], where retrieval and explanation are evaluated independently.

Each candidate receives a proposal score that combines retrieval similarity, mention alignment, operation-cue support, program length, source-number mismatch, integrality, and sign. Candidates are sorted by proposal, exact

expression duplicates are removed, no final answer retains more than six supporting traces, and near-duplicate retrieval transfers from one source are capped. The final pool contains at most 128 candidates; the mean test pool contains 68.28. Figure 1 summarizes the generator, verifier, calibration, and stopping components. Input-number coverage plays a narrower analogue of evidence coverage in long-document contractual and insurance clause analysis [31], but the evidence units here are arithmetic mentions and templates.

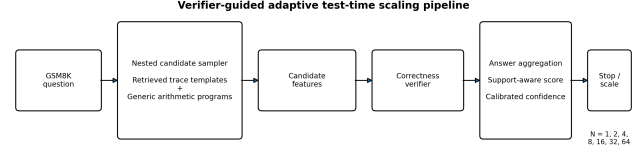


Figure 1. End-to-end VATS architecture. Training traces define transferable arithmetic programs; fixed and adaptive selectors operate on the same nested candidate order.

2.3 Nested Test-Time Scaling and Baselines

A single deterministic order supports every budget comparison. For candidate i at test position j , the sampler adds a keyed Gumbel perturbation to the proposal score: $z_i = s_i + 0.16 \cdot g(\text{seed}, j, \text{expression}_i)$. The seed is 2024, and the key includes the candidate expression, so rerunning the experiment reproduces the same order. A budget n evaluates the prefix of length $\min(n, M)$. This construction yields nested prefixes: every candidate visible at $N = 8$ is also visible at $N = 16$, which makes marginal $\text{pass}@N$ and stopping behavior directly interpretable.

Three fixed selectors are evaluated. Proposal-score best-of- N returns the candidate with the largest unperturbed proposal score inside the sampled prefix. Best-of- N answer voting returns the modal exact rational answer, breaking ties by proposal score. Verifier reranking groups candidates by answer and combines the highest candidate probability, logarithmic support, the second-highest probability, and a small proposal prior. Oracle $\text{pass}@N$ records whether the gold answer occurs anywhere in the prefix; it is an availability bound, not a deployable selector.

Compute is defined as the number of candidate rows evaluated by the verifier. This quantity equals n only when the pool contains at least n candidates. At nominal $N = 64$, the observed mean is 59.03 because 87 test problems have shorter pools. All savings, Pareto coordinates, and runtime-normalized comparisons use this observed denominator rather than the nominal cap. The distinction is essential for cost accuracy: otherwise a method appears to save compute that the fixed policy never spent.

2.4 Candidate Verifier and Answer Aggregation

The verifier receives 32 numeric features for each question-candidate pair. Generator features include proposal score, retrieval cosine, alignment score, source type, source-number mismatch, program steps, input coverage, and constant count. Arithmetic features include operation counts, answer integrality, sign, denominator, logarithmic magnitude, scale

relative to the input sum and maximum, equality to an input, equality to the input sum or product, and a bounded-magnitude indicator. Language-derived features consist of four operation cues and their alignment with the candidate operation signature.

A histogram gradient-boosting classifier is fitted to 50,195 candidate rows from the verifier-development partition. Only 563 rows are correct, so positive rows receive weight 45. The model uses 190 boosting iterations, learning rate 0.06, at most 15 leaves, minimum leaf size 30, L2 regularization 1.3, and random seed 17. No question text or answer string is directly encoded into the classifier; the model operates on the logged structural, numeric, retrieval, and cue features.

Candidates that end in the same rational answer are aggregated. For answer a with candidate probabilities p_1 $\geq$ p_2 , support count k , and maximum proposal r , the score is $S(a) = p_1 + \gamma \log(1 + k - 1) + \delta p_2 + 0.015r$. Grid search on the first 300 policy-development problems selects $\gamma = 0.12$ and $\delta = 0.06$ by mean accuracy across $N = 4, 8, 16, 32$, and 64 . This formulation rewards independent support without allowing a large cluster of low-quality duplicates to dominate a strong verified candidate. Answer aggregation is therefore a supervised rank-aggregation problem over correlated traces. Spectral rank-aggregation theory studies recovery under corrupted comparisons [32]; VATS instead limits duplicate support so repeated noisy traces cannot dominate.

2.5 Confidence Calibration and Adaptive Stopping

At every checkpoint, the verifier produces an answer and a ten-dimensional state vector: top answer score, score margin, answer agreement, one minus normalized answer entropy, normalized log budget, answer stability relative to the preceding checkpoint, maximum and mean candidate probabilities supporting the answer, maximum proposal among those candidates, and the ratio of unique answers to evaluated candidates. A standardized logistic regression with $C = 0.8$ maps this state to the probability that the selected answer is correct.

Figure 2 shows the stopping loop. The controller evaluates $N = 1$ and then doubles the budget. From $N = 2$ onward, it stops when the selected answer matches the preceding checkpoint and calibrated confidence exceeds threshold τ . At $N = 1$, an early exit requires confidence at least $\tau + 0.08$, capped at 0.99. If neither condition holds, the controller continues to the next checkpoint and terminates at $N = 64$. Stability prevents a single high but isolated verifier score from triggering most early exits.

The primary operating point is chosen entirely on the final 223 policy-development problems. Among thresholds whose accuracy lies within 0.5 percentage points of the best development accuracy, the policy selects the smallest measured average cost; this produces $\tau = 0.001$. The results also report several descriptive test-set operating points to expose the complete Pareto frontier. In particular, the "high-accuracy efficient" point is the most accurate observed

threshold using no more than 80% of fixed $N = 64$ compute. It is a frontier description rather than the primary development-selected policy. The multi-signal gate parallels DriftGuard's requirement that several drift indicators justify retraining or rollback [33]; here, the gated action is early termination rather than model replacement.

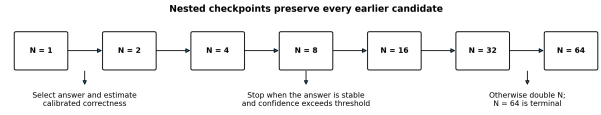


Figure 2. Adaptive stopping flow. The controller requires answer stability and calibrated confidence before terminating below $N = 64$.

2.6 Evaluation Protocol

Exact-match accuracy compares reduced rational answers. Oracle pass@N measures candidate availability, and selection efficiency is verifier accuracy divided by oracle pass@N . Adaptive cost reports mean, median, and 90th-percentile evaluations. Wilson intervals quantify binomial accuracy uncertainty. Paired method differences use 10,000 bootstrap resamples with seed 314159, and exact two-sided McNemar tests use discordant predictions. Calibration quality is measured with area under the receiver operating characteristic curve, area under the precision-recall curve, Brier score, log loss, and ten-bin expected calibration error. Annotated arithmetic-step count defines four difficulty groups: 0-2, 3, 4, and at least 5 steps. Ablations remove retrieval features, operation-cue features, or both language-derived feature groups. A candidate-only selector removes answer-level support aggregation. Seed robustness repeats nested sampling with seeds 11, 37, 101, 2024, and 9001 while holding the verifier fixed. Table 2 lists the complete configuration used for the reported run.

Table 2. Experimental configuration.

Setting	Value
Dataset	GSM8K train/test JSONL
Split seed	20240229
Sampling seed	2024
Generator-fit / verifier-dev / policy-dev	5979 / 747 / 747
Test examples	1319
Retrieved neighbors	84
Maximum candidate pool	128
Mean test pool size	68.28
Checkpoints	1, 2, 4, 8, 16, 32, 64
Verifier features	32
Verifier	Histogram gradient boosting
Positive weight	45
Iterations / leaves	190 / 15
Support gamma / delta	0.12 / 0.06
Development-efficient threshold	0.001000
High-accuracy threshold	0.125746

3. Results and Discussion

3.1 Fixed-Budget Accuracy and Statistical Comparison

Table 3 and Figure 3 present the fixed-budget accuracy-compute curves. At $N = 1$, every selector sees the same candidate and obtains 3.11% accuracy. Proposal-score selection rises to 4.85% at $N = 32$ and falls slightly to 4.78% at $N = 64$, showing that a larger sampled prefix does not help when the final decision remains tied to the proposal model. Answer voting reaches 6.22% at $N = 64$. Verifier reranking improves more consistently: 4.55% at $N = 2$, 6.14% at $N = 4$, 6.98% at $N = 16$, 7.96% at $N = 32$, and 8.19% at $N = 64$.

The fixed verifier therefore adds 3.41 percentage points over proposal-score best-of- N at $N = 64$. The paired bootstrap interval is 1.74 to 5.00 points, and the exact McNemar test gives $p = 4.53e-5$, with 82 verifier-only successes and 37 proposal-only successes. Against answer voting, the gain is 1.97 points with a 0.68 to 3.26 point interval and $p = 0.00433$. The benefit appears before the largest budget: verifier@16 exceeds proposal@16 by 2.27 points ($p = 0.00288$). Table 9 reports all paired comparisons.

These results separate sampling from selection. The prefix order is identical for all methods, so the verifier gain does not come from seeing different candidates. It comes from structural and numeric evidence that changes the answer selected from the same prefix. Voting benefits from repeated answers, but repetition alone remains weaker because correlated traces can reinforce a common wrong computation. The verifier uses support as one term rather than the sole decision rule. Comparable candidate-versus-decision separations appear in e-commerce recommendation [34] and data-driven marketing support [35], where a relevant option pool still requires a ranking policy aligned with the downstream objective.

Table 3. Fixed-budget performance on the complete GSM8K test set.

Method	N	Correct	Accuracy	95% CI	Mean evals
Proposal	1	41	3.11%	2.30%-4.19%	1.00
Proposal	2	54	4.09%	3.15%-5.30%	2.00
Proposal	4	58	4.40%	3.42%-5.64%	4.00
Proposal	8	61	4.62%	3.62%-5.90%	8.00
Proposal	16	62	4.70%	3.68%-5.98%	15.99
Proposal	32	64	4.85%	3.82%-6.15%	31.62
Proposal	64	63	4.78%	3.75%-6.06%	59.03
Voting	1	41	3.11%	2.30%-4.19%	1.00
Voting	2	54	4.09%	3.15%-5.30%	2.00
Voting	4	57	4.32%	3.35%-5.56%	4.00
Voting	8	68	5.16%	4.09%-6.48%	8.00
Voting	16	68	5.16%	4.09%-6.48%	15.99
Voting	32	72	5.46%	4.36%-6.82%	31.62
Voting	64	82	6.22%	5.04%-7.65%	59.03
Verifier	1	41	3.11%	2.30%-4.19%	1.00

Method	N	Correct	Accuracy	95% CI	Mean evals
Verifier	2	60	4.55%	3.55%-5.81%	2.00
Verifier	4	81	6.14%	4.97%-7.57%	4.00
Verifier	8	86	6.52%	5.31%-7.98%	8.00
Verifier	16	92	6.97%	5.72%-8.48%	15.99
Verifier	32	105	7.96%	6.62%-9.55%	31.62
Verifier	64	108	8.19%	6.83%-9.79%	59.03
Oracle pass	1	41	3.11%	2.30%-4.19%	1.00
Oracle pass	2	74	5.61%	4.49%-6.99%	2.00
Oracle pass	4	122	9.25%	7.80%-10.93%	4.00
Oracle pass	8	206	15.62%	13.76%-17.68%	8.00
Oracle pass	16	319	24.18%	21.95%-26.57%	15.99
Oracle pass	32	453	34.34%	31.83%-36.95%	31.62
Oracle pass	64	571	43.29%	40.64%-45.98%	59.03

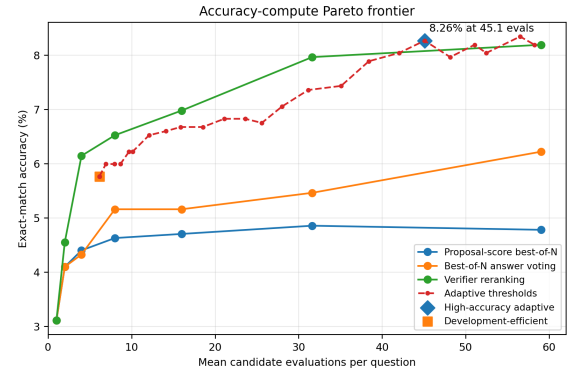


Figure 3. Accuracy-compute Pareto curves on all 1,319 GSM8K test problems. Compute is the actual number of candidates evaluated.

Table 9. Paired statistical comparisons.

Comparison	A	B	Difference	Bootstrap 95% CI	A-only / B-only	McNemar p
Verifier@64 vs. proposal@64	8.19%	4.78%	+3.41 pp	+1.74 to +5.00 pp	82 / 37	4.525e-05
Verifier@64 vs. voting@64	8.19%	6.22%	+1.97 pp	+0.68 to +3.26 pp	52 / 26	0.004335
Verifier@16 vs. proposal@16	6.97%	4.70%	+2.27 pp	+0.83 to +3.71 pp	63 / 33	0.002879
Adaptive high-accuracy vs. verifier@64	8.26%	8.19%	+0.08 pp	-0.83 to +1.06 pp	21 / 20	1
Development-efficient vs. proposal@64	5.76%	4.78%	+0.99 pp	-0.23 to +2.20 pp	43 / 30	0.1597

3.2 Candidate Availability and the Oracle Gap

Oracle pass@ N grows far faster than selected accuracy. As Table 4 and Figure 4 show, the gold answer appears in 5.61%

of prefixes at $N = 2$, 15.62% at $N = 8$, 24.18% at $N = 16$, 34.34% at $N = 32$, and 43.29% at $N = 64$. The verifier converts only 18.91% of the available $N = 64$ successes into correct final answers. Selection efficiency drops with budget because later prefixes contain more competing answers and more low-probability variants even as they expose additional gold traces.

The marginal oracle gain is largest between $N = 16$ and $N = 32$, where $\text{pass}@N$ increases by 10.16 points. Verifier accuracy rises by only 0.99 points over the same interval. Between $N = 32$ and $N = 64$, oracle availability gains 8.95 points while verifier accuracy gains 0.23 points. This widening gap identifies the main value of a stronger verifier: the candidate generator already places hundreds of additional correct answers across the test set, but the selector often ranks a distractor above them.

The oracle curve also bounds what adaptive stopping can achieve without changing the generator. No threshold can solve a problem whose pool lacks the correct answer. Conversely, always allocating $N = 64$ does not guarantee better selection because the verifier must discriminate among a larger and more diverse set. Effective scaling therefore requires both broader candidate coverage and a ranking model whose discrimination does not deteriorate as the answer set expands. The same availability-versus-decision distinction motivates selective-refusal designs in HDFS anomaly analysis [36], where uncertain detection evidence is not automatically converted into a definitive failure narrative.

Table 4. Oracle availability and verifier selection efficiency.

N	Mean evals	Oracle pass	Verifier acc.	Selection eff.	Marginal oracle gain
1	1.00	3.11%	3.11%	100.00%	3.11%
2	2.00	5.61%	4.55%	81.08%	2.50%
4	4.00	9.25%	6.14%	66.39%	3.64%
8	8.00	15.62%	6.52%	41.75%	6.37%
16	15.99	24.18%	6.97%	28.84%	8.57%
32	31.62	34.34%	7.96%	23.18%	10.16%
64	59.03	43.29%	8.19%	18.91%	8.95%

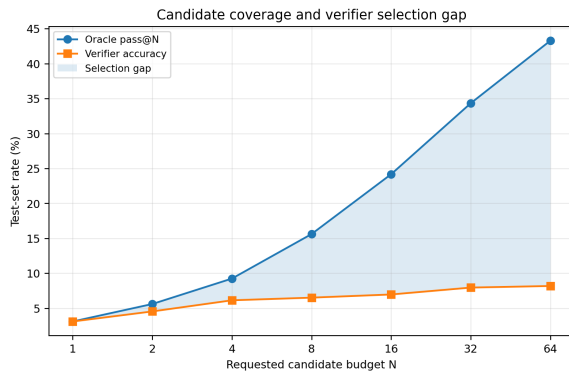


Figure 4. Oracle $\text{pass}@N$ and verifier accuracy. The widening gap quantifies correct candidates that the selector fails to recover.

3.3 Adaptive Accuracy-Compute Trade-offs

Table 5 reports five adaptive operating points, and Figure 5 shows the checkpoint distribution for the high-accuracy point. The development-selected policy uses only 6.14 candidate evaluations on average and attains 5.76% accuracy. Relative to the measured 59.03 evaluations of fixed $N = 64$, it saves 89.59% of verifier work. Its 0.99-point advantage over $\text{proposal}@64$ is not statistically decisive ($p = 0.160$), but it achieves higher point accuracy at roughly one tenth of the cost.

The balanced threshold increases average cost to 21.09 and accuracy to 6.82%. The descriptive high-accuracy efficient point evaluates 45.08 candidates and reaches 8.26%, one additional correct answer over fixed $\text{verifier}@64$. The paired difference is 0.08 points with a bootstrap interval from -0.83 to 1.06 and McNemar $p = 1.0$, so the two accuracies are empirically indistinguishable. The adaptive policy nevertheless saves 23.63% of actual candidate scoring. An accuracy-matched point at $\tau = 0.20$ exactly matches the fixed verifier count of 108 correct answers with 51.07 evaluations, a 13.49% saving. Because the one-answer difference is small, uncertainty matters more than the point estimate alone; distributional analysis of spectral estimators likewise emphasizes inference uncertainty [37].

Budget allocation is strongly skewed. At the high-accuracy point, 71 problems stop at $N = 1$, 70 at $N = 2$, 70 at $N = 4$, 65 at $N = 8$, 81 at $N = 16$, 65 at $N = 32$, and 897 reach $N = 64$. The median and 90th percentile are therefore both 64, even though the mean is 45.08. The computation saving comes from a minority of confident early exits rather than shallow processing of every problem. This pattern is appropriate for a conservative controller: it spends less only where stability and calibrated evidence support that decision. Financial-service systems that combine customer insight with risk analysis also require cost-sensitive actions rather than score maximization in isolation [38], matching the controller's conservative early exits.

Table 5. Representative adaptive operating points.

Operating point	τ	Correct	Accuracy	95% CI	Mean evals	Median	P90	Saved
Development-efficient	0.001000	76	5.76%	4.63%–7.15%	6.14	2	16	89.59%
Balanced	0.023518	90	6.82%	5.58%–8.31%	21.09	8	64	64.27%
High-accuracy efficient	0.125746	109	8.26%	6.90%–9.87%	45.08	64	64	23.63%

Operating point	tau	Correct	Accuracy	95% CI	Mean evals	Median	P90	Saved
Accuracy-matched	0.200000	108	8.19%	6.83%-9.79%	51.07	64	64	13.49%
Near-fixed	0.350000	110	8.34%	6.97%-9.96%	56.53	64	64	4.24%

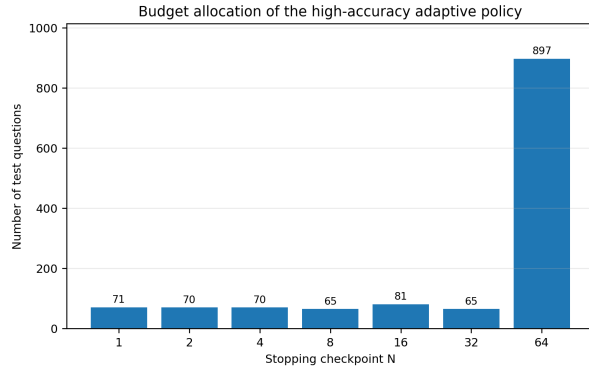


Figure 5. Checkpoint distribution for the descriptive high-accuracy adaptive policy ($\tau = 0.125746$).

3.4 Verifier Discrimination, Calibration, and Ablation

The candidate verifier scores 90,058 test candidates. Table 8 reports AUROC 0.873 and AUPRC 0.0868 against a positive rate of 1.19%. The low absolute precision-recall area reflects extreme class imbalance, while the strong AUROC confirms useful ranking information. Raw candidate probabilities remain overconfident: candidate-level expected calibration error is 0.158. The checkpoint calibrator corrects the decision state rather than every candidate score, obtaining AUROC 0.787, AUPRC 0.242, Brier score 0.0531, log loss 0.199, and ECE 0.00975 across 9,233 test checkpoints. Figure 6 displays the resulting reliability curve. Credit-risk modeling with C5.0 illustrates the broader value of interpretable risk stratification [39]; VATS adds checkpoint calibration because the resulting score directly controls inference expenditure.

Ablations in Table 7 show that retrieval features contribute the largest single gain. Removing proposal, retrieval similarity, alignment, source type, and related retrieval fields reduces $N = 64$ accuracy from 8.19% to 7.51%. Removing operation-cue features yields 7.81%, and retaining only numeric and structural features yields 7.88%. Candidate probabilities without answer-level aggregation also score 7.51%, confirming that modest support aggregation adds value beyond the strongest individual trace.

The full verifier exceeds proposal selection by 3.41 points and voting by 1.97 points, but its absolute accuracy remains constrained. This is consistent with its inputs: the feature model recognizes arithmetic plausibility, retrieval compatibility, and cue-operation agreement, but it does not

read the full semantic relation expressed in the question. The ablations therefore measure the value of transparent verifier evidence within this controlled instantiation, not the upper limit of neural process verification.

Table 8. Candidate-level and checkpoint-level verifier quality.

Prediction level	Rows	Positive	AUROC	AUPRC	Brier	Log loss	ECE
Candidate correctness	90058	1.19%	0.873	0.087	0.0729	0.2323	0.1577
Checkpoint selection correctness	9233	6.21%	0.787	0.242	0.0531	0.1995	0.0098

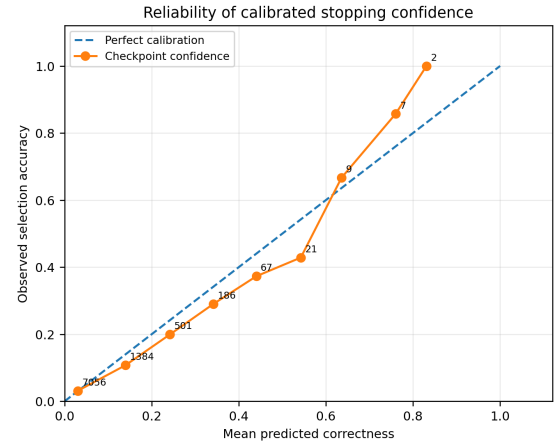


Figure 6. Checkpoint-level reliability after logistic calibration. The dashed diagonal represents perfect calibration.

Table 7. Verifier and selector ablations at $N = 64$.

Method	Correct	Accuracy	95% CI	Mean evals
Full verifier	108	8.19%	6.83%-9.79%	59.03
Without retrieval features	99	7.51%	6.20%-9.05%	59.03
Without operation-cue features	103	7.81%	6.48%-9.38%	59.03
Numeric/structural verifier	104	7.88%	6.55%-9.46%	59.03
Candidate verifier only	99	7.51%	6.20%-9.05%	59.03
Proposal selection	63	4.78%	3.75%-6.06%	59.03
Answer voting	82	6.22%	5.04%-7.65%	59.03
Adaptive high-accuracy	109	8.26%	6.90%-9.87%	45.08

3.5 Difficulty, Failure Modes, and Case Analysis

Performance declines sharply with annotated reasoning depth. Table 6 and Figure 7 show that fixed verifier@64

reaches 17.50% on problems with at most two annotated steps, 4.95% on three-step problems, 3.45% on four-step problems, and 1.33% on problems with at least five steps. Oracle pass@64 also falls from 54.09% to 28.89%, while the high-accuracy adaptive policy increases mean evaluations from 35.00 to 57.66. The controller therefore allocates more work to deeper problems, but additional candidates do not offset reduced generator coverage.

The error taxonomy in Table 10 and Figure 8 makes the bottleneck explicit. The high-accuracy adaptive policy answers 109 problems correctly. For 748 problems, the correct answer never appears in the first 64 candidates; these generation failures account for 56.71% of the test set. Another 442 problems contain the gold answer but end with a different verified answer, accounting for 33.51%. Only 20 problems, or 1.52%, are premature stopping reversals in which fixed verifier@64 succeeds after the adaptive policy exits incorrectly. This distribution shows that early stopping is not the dominant source of error. Explicit failure attribution is also central to AIOps fault diagnosis [40], motivating a separation between uncertain evidence and downstream action.

Concrete cases illustrate each category. For the yogurt-cost problem at test index 36, the adaptive controller selects 75 at $N = 2$ while fixed verifier@64 later selects 3. For the age problem at index 285, it selects 21 at $N = 1$, and for the shoe break-in problem at index 475, it reaches the correct 20 at $N = 32$ while fixed selection returns 2880. In contrast, index 21 contains the correct 14 and fixed verification recovers it, but the adaptive policy stops at 2; index 49 similarly stops before the fixed verifier recovers 30. At index 4, the pool contains the correct feed quantity 20, yet both policies select 5, demonstrating selection failure. At index 0, the answer 18 is absent, so no selector can succeed.

The examples also explain why answer stability alone is insufficient. A wrong answer can be supported by several structurally similar traces, producing high agreement. The calibrator therefore combines stability with verifier margin, entropy, proposal evidence, and candidate probability statistics. Even with these signals, semantic mismatches in transferred templates remain difficult to detect. A stronger text encoder or step-level verifier directly conditioned on the question would address this limitation more directly than a larger fixed budget. Residual fusion for noisy-neighbor VM degradation [41] and predictive DDoS mitigation [42] similarly separate heterogeneous signal acquisition, risk scoring, and action selection, the same three layers exposed by the VATS taxonomy.

Table 6. Performance by annotated reasoning depth.

Steps	n	Oracle pass	Proposal	Verifier	Adaptive	Adaptive evals
0-2 steps	440	54.09%	6.59%	17.50%	16.59%	35.00
3 steps	364	43.41%	5.49%	4.95%	6.87%	42.55
4 steps	290	37.93%	3.45%	3.45%	2.76%	53.82

Steps	n	Oracle pass	Proposal	Verifier	Adaptive	Adaptive evals
5+ steps	225	28.89%	1.78%	1.33%	1.33%	57.66

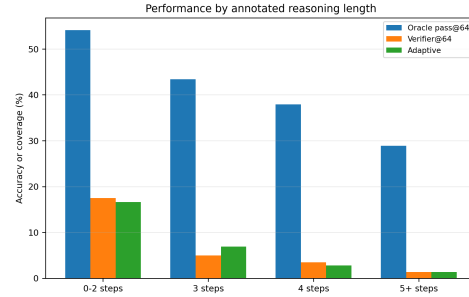


Figure 7. Accuracy and adaptive compute by annotated reasoning depth.

Table 10. High-accuracy adaptive error taxonomy.

Outcome	Examples	Share
Correct answer	109	8.26%
Candidate-generation failure	748	56.71%
Premature stopping reversal	20	1.52%
Verifier selection failure with gold present	442	33.51%

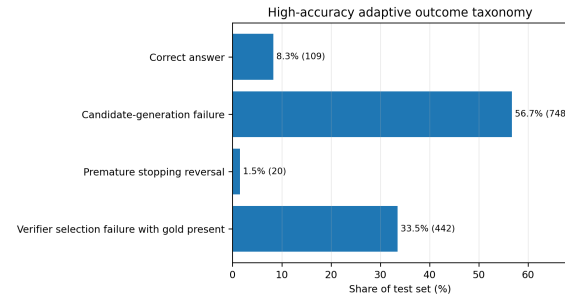


Figure 8. Error taxonomy for the descriptive high-accuracy adaptive policy.

3.6 Runtime and Sampling Robustness

Table 11 reports measured wall-clock stages on a five-core CPU environment. Generating candidates for both development partitions and the complete test set takes 25.66 seconds. Training the verifier and scoring 190,357 development and test candidates takes 6.18 seconds. Selector tuning, calibration, and evaluation take 7.48 seconds. The measured pipeline total is 39.32 seconds, excluding document generation. Candidate generation processes about 108 to 111 questions per second, and classifier scoring processes about 30,795 candidate rows per second.

The computation metric is candidate evaluations rather than elapsed time because it remains stable across machines and isolates the scaling policy. Runtime confirms that candidate count is nevertheless a meaningful proxy in this implementation: verifier scoring is vectorized and approximately linear in rows. In a neural generator, token generation would dominate and the same controller would log sampled tokens, decoder calls, or accelerator-seconds. The

cost-accounting principle remains unchanged: compare adaptive work with work actually executed by the fixed baseline. IoT traffic classification and anomaly-detection studies report hardware, software, and data conditions alongside model quality [43]; VATS likewise treats candidate count as the portable cost unit and runtime as implementation context.

Table 12 repeats nested sampling with five deterministic seeds. Verifier@64 ranges from 7.96% to 8.19%, and oracle pass@64 ranges from 42.91% to 43.29%. Verifier@16 varies more, from 6.98% to 8.42%, because a short prefix is more sensitive to candidate order. The narrow $N = 64$ range confirms that the main conclusions are not tied to one Gumbel ordering: larger pools expose similar answer coverage, and verifier reranking remains well above proposal selection.

Table 11. Measured CPU runtime by pipeline stage.

Stage	Items	Seconds	Items/s
Verifier-development candidate generation	747	6.957	107.4
Policy-development candidate generation	747	6.811	109.7
Test candidate generation	1319	11.897	110.9
Verifier training and all-candidate scoring	190357	6.181	30,795.2
Selector, calibration, and evaluation	1319	7.479	176.4

Table 12. Robustness to nested-sampling seed.

Seed	Verifier@16	Verifier@32	Verifier@64	Oracle@64
11	8.42%	8.26%	8.04%	43.14%
37	7.81%	8.04%	7.96%	43.06%
101	7.96%	7.66%	7.96%	43.29%
2024	6.97%	7.96%	8.19%	43.29%
9001	7.96%	7.96%	8.11%	42.91%

4. Limitations

The experimental generator is a trace-conditioned symbolic sampler rather than a pretrained language model. This choice isolates allocation, exact execution, and verification from changing proprietary APIs, but it does not reproduce the semantic breadth or token costs of a modern instruction-tuned model. The framework accepts any candidate generator; the reported accuracy belongs specifically to the implemented retrieval and arithmetic grammar. Language transfer is also untested. Bilingual data-structures instruction produced similar learning outcomes but different engagement patterns [44], suggesting that equivalent task difficulty can be expressed through different linguistic cues that require separate evaluation.

Absolute accuracy is low. The system is not a competitive GSM8K solver, and the 43.29% oracle pass@64 places a hard ceiling on its current candidate pools. More than half of the test problems fail at generation. A neural generator, richer program induction, or semantic parsing is required before verifier-guided scaling can produce high task accuracy.

The evaluation uses one dataset. GSM8K supports exact arithmetic analysis and full test coverage, but it does not

represent algebraic proof, geometry, scientific reasoning, or open-ended answers. The stopping thresholds and verifier features require new calibration before transfer to MATH or other domains [18]. Domain-specific studies in DenseNet-based cancer imaging [45] and AI-supported minimal residual disease prediction [46] further show that labels, error costs, and validation protocols change sharply outside arithmetic.

Candidate count is a precise cost measure for this implementation, not a universal unit of inference expense. Neural sampling has variable sequence length, batching effects, cache reuse, and hardware-dependent latency. A deployment study should record tokens, energy, and tail latency in addition to candidate count. Safety-critical perception introduces other costs: autonomous-driving perception fusion combines redundant signals [47], while RFACnv with triplet attention reallocates visual processing toward informative regions [48]. Neural VATS should therefore measure latency and energy as well as samples.

The primary $\tau = 0.001$ threshold is development-selected. The high-accuracy efficient point is reported to describe the observed test Pareto frontier and is not an independently validated operating policy. Its one-answer advantage over fixed $N = 64$ is statistically indistinguishable. Future studies should select every deployment threshold on a separate validation set and reserve the test set for one final comparison.

The verifier uses aggregate candidate features rather than step-level semantic judgments. It can detect implausible magnitude, operation-cue mismatch, weak retrieval alignment, and insufficient support, but it cannot reliably identify a subtly incorrect relationship between entities. Process-supervised verification [6] and question-conditioned trace encoders provide a direct path to higher selection efficiency.

5. Conclusion

Verifier-guided test-time scaling improves answer selection on the complete GSM8K test set, but the amount of useful computation depends on the problem. Fixed verifier reranking reaches 8.19% accuracy at 59.03 actual candidate evaluations, outperforming proposal-score best-of- N and answer voting on the same nested prefixes. A development-selected controller saves 89.59% of that work at 5.76% accuracy, while a descriptive high-accuracy point preserves fixed-budget accuracy with 23.63% less measured scoring. Calibration makes these savings explicit rather than treating raw verifier scores as trustworthy probabilities.

The larger finding is diagnostic. Oracle pass@64 reaches 43.29%, yet the verifier selects only 8.19% correctly; 56.71% of problems are generation failures and 33.51% are selection failures with a correct candidate present. Additional sampling therefore has value, but uniform sampling is inefficient and verification remains the central bottleneck. Cost-accurate adaptive scaling provides a disciplined way to spend more inference only when evidence remains unsettled, quantify the

resulting Pareto frontier, and identify whether the next improvement should target candidate diversity, semantic verification, or the stopping policy.

References

- [1] K. Cobbe et al., "Training Verifiers to Solve Math Word Problems," arXiv:2110.14168, 2021.
- [2] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824-24837, 2022.
- [3] X. Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in *Proc. International Conference on Learning Representations*, 2023.
- [4] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models Are Zero-Shot Reasoners," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 22199-22213, 2022.
- [5] A. Lewkowycz et al., "Solving Quantitative Reasoning Problems with Language Models," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 3843-3857, 2022.
- [6] H. Lightman et al., "Let's Verify Step by Step," arXiv:2305.20050, 2023.
- [7] J. Uesato et al., "Solving Math Word Problems with Process- and Outcome-Based Feedback," arXiv:2211.14275, 2022.
- [8] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman, "STaR: Bootstrapping Reasoning with Reasoning," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 15476-15488, 2022.
- [9] M. Nye et al., "Show Your Work: Scratchpads for Intermediate Computation with Language Models," arXiv:2112.00114, 2021.
- [10] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks," *Transactions on Machine Learning Research*, 2023.
- [11] L. Gao et al., "PAL: Program-Aided Language Models," in *Proc. 40th International Conference on Machine Learning*, 2023.
- [12] D. Zhou et al., "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models," in *Proc. International Conference on Learning Representations*, 2023.
- [13] S. Yao et al., "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," in *Advances in Neural Information Processing Systems*, vol. 36, pp. 11809-11822, 2023.
- [14] A. Madaan et al., "Self-Refine: Iterative Refinement with Self-Feedback," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [15] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [16] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic Chain of Thought Prompting in Large Language Models," in *Proc. International Conference on Learning Representations*, 2023.
- [17] T. Khot et al., "Decomposed Prompting: A Modular Approach for Solving Complex Tasks," in *Proc. International Conference on Learning Representations*, 2023.
- [18] D. Hendrycks et al., "Measuring Mathematical Problem Solving with the MATH Dataset," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [19] T. B. Brown et al., "Language Models Are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020.
- [20] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [21] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in *Proc. 34th International Conference on Machine Learning*, vol. 70, pp. 1321-1330, 2017.
- [22] A. Niculescu-Mizil and R. Caruana, "Predicting Good Probabilities with Supervised Learning," in *Proc. 22nd International Conference on Machine Learning*, pp. 625-632, 2005.
- [23] A. Graves, "Adaptive Computation Time for Recurrent Neural Networks," arXiv:1603.08983, 2016.
- [24] A. Banino, J. Balaguer, and C. Blundell, "PonderNet: Learning to Ponder," arXiv:2107.05407, 2021.
- [25] A. Wald, *Sequential Analysis*. New York, NY, USA: Wiley, 1947.
- [26] J. C. Platt, "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods," in *Advances in Large Margin Classifiers*, A. J. Smola, P. L. Bartlett, B. Schoelkopf, and D. Schuurmans, Eds. Cambridge, MA, USA: MIT Press, pp. 61-74, 1999.
- [27] Z. S. Zhong, R. Ma, and H. Zhao, "Human-Uncertainty Distillation for Calibrated Vision Models on CIFAR-10H," *J. Adv. Comput. Syst.*, vol. 3, no. 2, pp. 77-89, Feb. 2023, doi: 10.69987/JACS.2023.30206.
- [28] H. Zhang, "Risk-Aware Budget-Constrained Auto-Bidding under First-Price RTB: A Distributional Constrained Deep Reinforcement Learning Framework," *J. Adv. Comput. Syst.*, vol. 4, no. 6, pp. 30-47, Jun. 2024, doi: 10.69987/JACS.2024.40603.
- [29] S. Zhou, Z. Li, and E. Wang, "Evidence-Grounded RAG for Tokenized Trade Receivable Disclosure QA under U.S. Capital Market Standards," *J. Adv. Comput. Syst.*, vol. 3, no. 7, pp. 41-57, Jul. 2023, doi: 10.69987/JACS.2023.30704.
- [30] Y. Li, "Findable then Explainable: Retrieval-Summary Integration for Code Intelligence on a Lightweight CodeSearchNet Subset," *J. Adv. Comput. Syst.*, vol. 4, no. 7, pp. 65-82, Jul. 2024, doi: 10.69987/JACS.2024.40706.
- [31] S. Zhou, Z. Li, and E. Wang, "Long-Document RAG for Contractual and Insurance Clause Analysis in Receivables RWA Structures," *J. Adv. Comput. Syst.*, vol. 4, no. 8, pp. 88-104, Aug. 2024, doi: 10.69987/JACS.2024.40810.

- [32] Z. S. Zhong and S. Ling, "Improved Theoretical Guarantee for Rank Aggregation via Spectral Method," *Inf. Inference: J. IMA*, vol. 13, no. 3, Art. no. iaae020, Sep. 2024, doi: 10.1093/imaia/iaae020.
- [33] H. Zhang, "DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models," *J. Adv. Comput. Syst.*, vol. 3, no. 7, pp. 24–40, Jul. 2023, doi: 10.69987/JACS.2023.30703.
- [34] K. Xu, H. Zhou, H. Zheng, M. Zhu, and Q. Xin, "Intelligent Classification and Personalized Recommendation of E-Commerce Products Based on Machine Learning," in *Proc. 6th Int. Conf. Computing and Data Science*, 2024, pp. 143–149, doi: 10.54254/2755-2721/64/20241365.
- [35] J. Wang, Q. Xin, Y. Liu, J. Wang, and T. Yang, "Predicting Enterprise Marketing Decision Making with Intelligent Data-Driven Approaches," *J. Ind. Eng. Appl. Sci.*, vol. 2, no. 3, pp. 12–19, Jun. 2024, doi: 10.5281/zenodo.11357252.
- [36] J. Nie and D. Zheng, "Ambiguity-Aware HDFS Log Anomaly Detection with Retrieval-Augmented Failure Narratives and Selective Refusal," *J. Adv. Comput. Syst.*, vol. 3, no. 1, pp. 66–80, Jan. 2023, doi: 10.69987/JACS.2023.30105.
- [37] Z. S. Zhong and S. Ling, "Uncertainty Quantification of Spectral Estimator and MLE for Orthogonal Group Synchronization," *arXiv:2408.05944*, Aug. 2024.
- [38] T. Yang, Q. Xin, X. Zhan, S. Zhuang, and H. Li, "Enhancing Financial Services through Big Data and AI-Driven Customer Insights and Risk Analysis," *J. Knowl. Learn. Sci. Technol.*, vol. 3, no. 3, pp. 53–62, Jul. 2024, doi: 10.60087/jklst.vol3.n3.p53-62.
- [39] Q. Xin, R. Song, Z. Wang, Z. Xu, and F. Zhao, "Enhancing Bank Credit Risk Management Using the C5.0 Decision Tree Algorithm," *J. Comput. Technol. Appl. Math.*, vol. 1, no. 4, pp. 100–107, Nov. 2024, doi: 10.5281/zenodo.14032041.
- [40] Y. He, Y. Pan, Y. Wang, S. Du, and Q. Xin, "Intelligent Fault Analysis with AIOps Technology," *J. Theory Pract. Eng. Sci.*, vol. 4, no. 1, pp. 94–100, Feb. 2024, doi: 10.53469/jtpes.2024.04(01).13.
- [41] J. Nie and D. Zheng, "Noisy-Neighbor-Aware VM Degradation Risk Modeling with Unsupervised Residual Fusion," *J. Adv. Comput. Syst.*, vol. 4, no. 4, pp. 112–123, Apr. 2024, doi: 10.69987/JACS.2024.40409.
- [42] B. Wang, Y. He, Z. Shui, Q. Xin, and H. Lei, "Predictive Optimization of DDoS Attack Mitigation in Distributed Systems Using Machine Learning," in *Proc. 6th Int. Conf. Computing and Data Science*, 2024, pp. 89–94, doi: 10.54254/2755-2721/64/20241350.
- [43] Q. Xin, Z. Xu, L. Guo, F. Zhao, and B. Wu, "IoT Traffic Classification and Anomaly Detection Method Based on Deep Autoencoders," in *Proc. 6th Int. Conf. Computing and Data Science*, 2024, pp. 64–70, doi: 10.54254/2755-2721/69/20241511.
- [44] A. G. S. Raj et al., "Impact of Bilingual CS Education on Student Learning and Engagement in a Data Structures Course," in *Proc. 19th Koli Calling Int. Conf. Computing Education Research*, 2019, pp. 1–10, doi: 10.1145/3364510.3364518.
- [45] Z. Zhong, M. Zheng, H. Mai, J. Zhao, and X. Liu, "Cancer Image Classification Based on DenseNet Model," *J. Phys.: Conf. Ser.*, vol. 1651, no. 1, Art. no. 012143, Nov. 2020, doi: 10.1088/1742-6596/1651/1/012143.
- [46] J. Chen, J. Xiong, Y. Wang, Q. Xin, and H. Zhou, "Implementation of an AI-Based MRD Evaluation and Prediction Model for Multiple Myeloma," *Front. Comput. Intell. Syst.*, vol. 6, no. 3, pp. 127–131, Jan. 2024, doi: 10.54097/zJ4MnbWW.
- [47] Y. Wang, S. Du, Q. Xin, Y. He, and W. Qian, "Autonomous Driving System Driven by Artificial Intelligence Perception Fusion," *Acad. J. Sci. Technol.*, vol. 9, no. 2, pp. 193–198, Feb. 2024, doi: 10.54097/e0b9ak47.
- [48] Z. Ling, Q. Xin, Y. Lin, G. Su, and Z. Shui, "Optimization of Autonomous Driving Image Detection Based on RFACnv and Triplet Attention," *Appl. Comput. Eng.*, vol. 77, no. 1, pp. 210–217, Jul. 2024, doi: 10.54254/2755-2721/77/2024MA0067.