STEPROUTER: FROM EFFORT PRIORS TO UTILITY POSTERIORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-Thought (CoT) and related reasoning paradigms rely on *historical tokens*—intermediate conclusions from prior steps. Not all historical tokens are equally useful, and the *lost in the middle* phenomenon makes indiscriminate accumulation harmful. We study how to rank and select historical tokens, organizing them at the granularity of *steps*, which manifest as lemmas in mathematics, TODOs in code, or other domain-specific units. As a first step, we compare two ways to rank candidate steps. The first is an **effort prior**, which favors steps that look expensive to verify. The second is an **ex-post utility** score, which measures how much a step improves the final answer per token, judged by a frontier model. Empirically, effort is nearly uncorrelated with true usefulness, while utility-based ranking substantially outperforms effort and other baselines at selecting high-value steps. Despite the simplicity of these signals, we develop STEPROUTER, a lightweight posterior utility estimator that guides any backbone to rank steps by utility, routing only the top 20% for reuse. The results are surprisingly strong: StepRouter improves *all* tested models on several reasoning benchmarks, yielding +19.6 pp on LiveCodeBench with DeepSeek-R1, +23.2 pp with Qwen3-8B, and pushing GPT-5 to 97.2% on AIME. Fine-tuning Qwen3-8B with only 166 examples yields +12.2 pp on AIME (68.16%→80.36%), demonstrating that models can internalize effective token utilization.

1 INTRODUCTION

LLM reasoning approaches such as chain-of-thought (Wei et al., 2022), process reward models (PRMs) (Zhang et al., 2025; Zheng et al., 2025), and tree-of-thought search (Yao et al., 2023a) rely on *historical tokens*—intermediate conclusions generated during prior reasoning steps. As these sequences grow, the "lost in the middle" phenomenon (Liu et al., 2024) complicates their use. Memory mechanisms (Packer et al., 2023; Xu et al., 2025c; Yao et al., 2023b) address persistent storage, but not all historical tokens are equally useful (Shi et al., 2023; Xia et al., 2025). Indiscriminate accumulation can dilute useful context (Du et al., 2025), motivating explicit *ranking and selection* (Chirkova et al., 2025; Li et al., 2023).

We organize historical tokens at the granularity of *steps*: self-contained intermediate conclusions such as lemmas in mathematics or TODOs in code. We study two signals: an **effort prior**, based on predicted verification cost, and **ex-post utility**, the improvement in judged solution quality per realized verification token. Effort exhibits near-zero rank correlation with raw usefulness ($\rho \approx 0.01$) and negative top-1 marginal gain, whereas learned utility reaches $\rho \approx 0.26$ with better top-k selection (Table 4). This complements evidence that test-time compute must be allocated strategically (Damani et al., 2024; Snell et al., 2024).

We develop STEPROUTER, a lightweight two-layer MLP on frozen bge-m3 embeddings (Chen et al., 2024a), as a *plugin* for ranking historical steps. For each candidate, we compare outputs produced with and without that step, scored by a model judge (Zheng et al., 2023). Outcome-induced preferences train the scorer; at inference time it routes only the top-20% of the candidate pool (Figure 1). Here "posterior" means an *a posteriori* usefulness preference, not a calibrated Bayesian posterior.

One scorer improves the tested DeepSeek-R1, GPT-5, and Qwen3-8B backbones (DeepSeek-AI et al., 2025; Singh et al., 2025; Yang et al., 2025) across AIME, GPQA-Diamond, and LiveCodeBench

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.

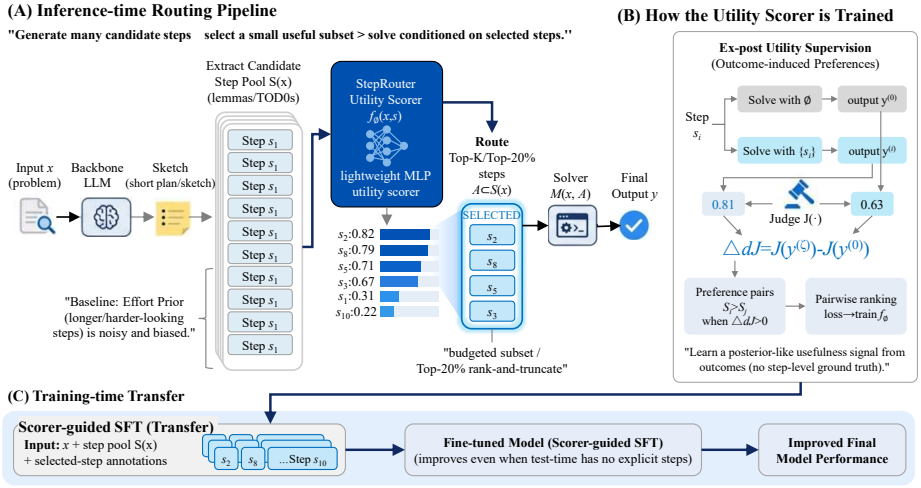

Figure 1: **Overview of StepRouter. (A) Inference:** the backbone sketches a solution and extracts candidate steps $S(x)$; the scorer ranks them, and the solver receives the original problem plus the top-20% subset. **(B) Scorer training:** a judge compares solutions with and without each candidate. Outcome gains, normalized by verification cost, define within-problem preferences for pairwise scorer training. Verification and judging occur offline, not in (A). **(C) Training transfer:** selected-step annotations and selection-and-use explanations provide SFT supervision; the fine-tuned model can then solve without externally supplied steps. The implemented scorer encodes step text only (Section 3.2); diagram counts and scores are schematic.

(Jain et al., 2024; Rein et al., 2023). Gains include 19.61 percentage points (pp) on LiveCodeBench with DeepSeek-R1 and 23.23 pp with Qwen3-8B. Same-pool controls isolate the contribution of selection from simply generating or shortening additional context. Token savings depend on the setting: Qwen3-8B/AIME uses fewer generated tokens, whereas its coding improvement uses more.

The utility signal also transfers to training. With only 166 examples, scorer-guided SFT improves question-only solving and transfers from Qwen3-8B-generated supervision to Qwen3-32B. A matched solution-only control and three-run evaluation further test the contribution of utility-guided supervision beyond ordinary SFT.

Our contributions are (i) characterizing the gap between verification effort and downstream usefulness, (ii) learning a lightweight step router from outcome-induced preferences, and (iii) evaluating utility-guided routing and training-time transfer through same-pool controls, matched SFT, and a step-usage benchmark.

2 Problem Formulation

We consider reasoning tasks where a model produces intermediate conclusions before arriving at a final answer. We refer to these conclusions as *historical tokens*. They are abundant and heterogeneous: some capture decisive constraints, while others are redundant, incorrect, or distracting. Our goal is to identify which historical tokens are worth reusing.

We organize them into reusable *steps*, such as lemmas in mathematics or TODO-style subgoals in code. Given an input, a backbone first produces a sketch and extracts a candidate-step pool. A router scores the candidates and supplies a small top-ranked subset to the downstream LLM together with the original problem.

The central problem is to rank steps by *downstream utility*: a step should rank higher if including it tends to improve task success. An **effort prior** instead favors candidates that appear costly to

verify. Our **ex-post utility** signal measures value per verification token: value is an outcome-score difference, and cost comes from an explicit verification run rather than the candidate's text length. This supervision trains a lightweight router that can be reused across backbones and domains.

3 UTILITY-GUIDED STEP ROUTING

3.1 SETUP AND NOTATION

Given an input x, an LLM generates a sketch and extracts a pool $\mathcal{S}(x) = \{s_i\}_{i=1}^{N}$ of reusable lemmas or TODOs. The scorer ranks candidates and forms a routed subset:

$$\mathcal{A}(x) = \mathrm{Top}_{k(x)}\{\widehat{u}_\theta(s) : s \in \mathcal{S}(x)\}, \quad k(x) = \max(1, \lceil 0.2N \rceil), \qquad y = M(x, \mathcal{A}(x)). \tag{1}$$

Top_k returns candidate statements, not their score values; an empty pool supplies no hints. Steps are optional context that the final solver may verify or ignore.

3.2 LEARNING A STEP SCORER FROM WEAK SUPERVISION

Outcome-difference labels. For each candidate s_i, we obtain matched with-step and no-step outputs and compare them with a common judge:

$$y^{(0)} = M(x, \varnothing), \quad y^{(i)} = M(x, \{s_i\}), \quad d_i = J(x, y^{(i)}) - J(x, y^{(0)}). \tag{2}$$

GPT-5 is the default judge; alternative judges are studied in Section 7. Its fixed 0–10 rubric measures solution correctness and reasoning quality. Objective benchmark accuracy is evaluated separately. The efficiency-normalized label is

$$\widehat{u}_i = \frac{d_i}{c_i + \epsilon}, \qquad \epsilon > 0, \tag{3}$$

where c_i is the *realized* generated-token cost of an explicit proof or verification of the candidate. Utilities are normalized to a within-problem 1–10 scale before forming preferences. The ratio defines the training target, rather than only breaking ties between equal gains.

Scorer training data and preferences. We use 647 candidate steps from 60 AIME 2023–2024 problems and 66 LiveCodeBench training problems, all excluded from evaluation. We form within-problem preferences $s_i \succ s_j$ when $\widehat{u}_i > \widehat{u}_j$, and optimize pairwise logistic loss:

$$\mathcal{L} = -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \sigma(\widehat{u}_\theta(s_i) - \widehat{u}_\theta(s_j)). \tag{4}$$

The 80/20 split is grouped by problem ID; training details and validation ranking metrics are in Appendix A.

Scorer architecture. The scorer is a two-layer MLP on *frozen* bge-m3 dense embeddings (Chen et al., 2024a): L2-normalized 1,024-dimensional embeddings feed a 128-unit ReLU hidden layer and a scalar output. It encodes candidate text s only, hence the notation $\widehat{u}_\theta(s)$ rather than an explicitly problem-conditioned $f_\phi(x, s)$. Problem information can appear inside the candidate statement, but the original problem is not separately encoded:

$$\widehat{u}_\theta(s) = \mathrm{MLP}_\theta\left(E(s)/\|E(s)\|_2\right). \tag{5}$$

3.3 INFERENCE-TIME ROUTING WITH THE LEARNED SCORER

At inference time, the same backbone generates the sketch, extracts the pool, and solves with the routed steps. There is no online judge or candidate-by-candidate verification. One trained scorer is reused across the reported domains and backbones. The 20% fraction was selected on AIME 2023–2024 development problems and frozen before evaluation. It constrains candidate count, not token length or total computation.

3.4 Distilling scorer-guided selection into LLMs

A fixed Qwen3-8B generator creates sketches and candidate pools for 166 training problems. The scorer marks the top-20% as `[good]` and the remaining steps as `[bad]`. We LoRA-fine-tune Qwen3-8B and Qwen3-32B to explain the selected steps and solve using them, without a separate RL objective. This distills *selection-and-use* behavior into the backbone. The matched solution-only control uses the same problems, complete solutions, optimization settings, and effective training-token budget, isolating the added utility-guided format.

4 Experimental Setup

Tasks and backbones. We evaluate AIME 2025, GPQA-Diamond (Rein et al., 2023), and LiveCodeBench (Jain et al., 2024), using GPT-5, DeepSeek-R1, and Qwen3-8B. Each backbone generates its own candidate pools; selectors within a backbone comparison share the pool and final-solving protocol. Accuracy uses mathematical answers, fixed answer-choice labels, or execution tests. Qwen3-8B results average per-problem accuracy over 12 completions.

Compared methods. We compare Direct, Effort Prior, S1-like continuation, Self-Consistency, the Qwen2.5-Math PRM, and StepRouter. Candidate selectors use top-20% routing; Direct and continuation are distinct inference strategies rather than subset selectors. The S1-like baseline is local re-prompting, not the trained s1 system (Muennighoff et al., 2025). Additional controls hold the candidate pool fixed and vary selection or compression.

Reporting conventions. Inference cost counts generated output tokens across sketch, extraction, and final solving, including discarded candidates. It excludes input tokens and offline label generation. Tables report each evaluation at its recorded precision. The separate GPT-5 exact-count evaluation uses 30 problems with 12 completions each. Its 360-completion denominator applies only to that evaluation, and its confidence interval resamples problems. For the three-run SFT comparison, uncertainties are sample standard deviations. Accuracy differences are reported in percentage points (pp).

5 Inference-Time Routing

Table 1 compares direct solving, effort routing, self-consistency, PRM selection, S1-like continuation, and StepRouter. On Qwen3-8B, the gains are 6.16 pp on AIME and 23.23 pp on LiveCodeBench. DeepSeek-R1 gains 19.61 pp on LiveCodeBench. These are absolute accuracy differences, not relative percentage improvements.

The GPT-5/AIME exact-count study in Table 3 is a separate evaluation from Table 1, with its own completion counts. Each table retains the measurements from its respective evaluation.

Accuracy and generation cost must be read together. Qwen3-8B/AIME improves while using 9.53% fewer generated tokens (18,532 versus 20,484). On LiveCodeBench, the same backbone improves more substantially, but generates 27.85% more tokens (8,356 versus 6,536). Thus routing is not universally token-saving. Its benefit is to allocate reusable context effectively; whether that also shortens generation depends on the task and backbone.

5.1 What does learned selection add?

Table 2 holds the candidate pool and solver fixed. Random, positional, effort-based, and learned selectors retain the same number of candidates. Fixed selection takes candidates in sketch order; effort selection prefers larger predicted verification-token requirements. StepRouter exceeds random selection by 5.28 pp on AIME and 18.92 pp on LiveCodeBench. These differences estimate the added benefit of learned ranking *within the same pipeline*, rather than attributing the entire improvement over Direct to the ranker.

Direct and all-step routing provide complementary controls rather than equal-budget comparisons. Supplying every candidate reduces accuracy below Direct on both tasks; merely adding a sketch-derived pool is insufficient. Compressing all candidates to the comparison's context budget improves

Table 1: **Inference-time routing results (Top-20%).** Selectors rank extracted candidates and route the top 20% before solving. **Acc** is accuracy in percent; **Tok** is the mean number of generated tokens (reasoning + output), rounded to integers, across sketching, candidate extraction, and final solving. Self-consistency includes the cost of all sampled solutions.

Backbone	Method (Top-20% routing)	AIME (2025)		GPQA-Diamond		LiveCodeBench	
		Acc ↑	Tok ↓	Acc ↑	Tok ↓	Acc ↑	Tok ↓
DeepSeek-R1-0120	Direct (no routing)	70.42	21392	68.52	14835	70.00	17642
	Effort Prior (Sketch→Steps→Final)	75.51	23686	75.12	18260	81.06	20491
	Self-Consistency	71.39	57104	71.57	41451	–	–
	PRM (Qwen2.5-math-PRM)	76.47	22582	–	–	–	–
	S1-like	58.21	25814	61.46	16452	65.39	20216
	STEPROUTER (Utility Scorer)	**80.04**	**18582**	**77.11**	**13942**	**89.61**	**14295**
GPT-5	Direct (no routing)	92.63	8449	82.02	9455	82.10	15479
	Effort Prior (Sketch→Steps→Final)	96.26	12932	88.52	14621	89.70	18363
	PRM (Qwen2.5-math-PRM)	95.42	11462	–	–	–	–
	S1-like	–	–	80.52	10471	81.53	16365
	Self-Consistency(3-Samples)	94.52	23691	85.12	29450	–	–
	STEPROUTER (Utility Scorer)	**97.19**	**9026**	**89.24**	**8326**	**92.61**	**14954**
Qwen3-8B	Direct (no routing)	68.16	20484	62.01	8977	57.56	6,536
	Effort Prior (Sketch→Steps→Final)	70.25	21306	65.14	9524	70.46	13786
	Self-Consistency(3-Samples)	70.82	61326	64.92	27932	–	–
	PRM (Qwen2.5-math-PRM)	71.36	20892	–	–	–	–
	S1-like	50.51	22435	–	–	–	–
	STEPROUTER (Utility Scorer)	**74.32**	**18532**	**67.62**	**7636**	**80.79**	**8356**

Table 2: **Same-pool routing controls.** Qwen3-8B accuracy in percent. The top block matches candidate count, not necessarily token count. Compression instead matches the comparison's context-token budget. The post-hoc oracle ranks individual candidates using evaluation-time outcomes; it is neither deployable nor an optimal joint subset selector.

Method	Retained context	AIME	LiveCodeBench
Random	20% of candidates	69.04	61.87
Fixed order	20% of candidates	69.5	62.4
Effort	20% of candidates	70.25	70.46
STEPROUTER	20% of candidates	**74.32**	**80.79**
Direct	No routed candidates	68.16	57.56
All-step	All candidates	65.29	54.31
Compressed all-step	Token-matched compressed context	70.1	65.3
Post-hoc oracle	20%, ranked by measured gain	77.90	85.40

over uncompressed all-step routing, but remains below learned selection. This result distinguishes the learned selector from the tested shortening strategy, without claiming that every possible compressor has been ruled out. The post-hoc oracle ranks candidates by their individual empirical gain and exposes remaining ranking headroom, not optimal joint selection.

A near-ceiling backbone. The GPT-5/AIME comparison in Table 3 uses one common 360-completion protocol. STEPROUTER obtains 350 correct completions versus 339 for Random. The difference is $100(350 - 339)/360 = 3.06$ pp after rounding, with the reported problem-cluster bootstrap 95% interval $[+0.7, +5.4]$ pp. This interval concerns STEPROUTER minus Random, not STEPROUTER minus Direct, and is not evidence of significance for every result in Table 1.

5.2 WHY UTILITY RANKING WORKS

Training uses the cost-normalized label in Equation 3, but the diagnostic in Table 4 evaluates correlation with the *unnormalized* matched outcome difference d_i. Effort has near-zero Spearman and Kendall correlation with raw gain, whereas the learned scorer is positively correlated. The Top-k

Table 3: **GPT-5 exact-count evaluation.** Each method is evaluated on 30 AIME problems with 12 completions per problem. Percentages and differences are calculated from counts before rounding; repeated completions are not independent problems.

GPT-5/AIME method	Correct / 360	Accuracy (%)
Direct	334/360	92.78
Random, 20%	339/360	94.17
Effort, 20%	346/360	96.11
STEPROUTER, 20%	350/360	**97.22**

Table 4: **Ranking quality on held-out candidate pools.** Spearman's ρ and Kendall's τ compare each signal with raw outcome gain. Top-k reports the mean individual gain of selected candidates, not their joint effect. Effort has near-zero correlation and negative Top-1 gain; the utility scorer improves all five metrics.

Signal	$\rho \uparrow$	$\tau \uparrow$	Top-1 $\uparrow$	Top-3 $\uparrow$	Top-5 $\uparrow$
Effort Prior	0.0076	0.0019	-0.1187	0.1375	0.0463
Utility Scorer	**0.2608**	**0.2019**	**0.9812**	**0.6000**	**0.3700**

diagnostic averages the individual raw gains of the top-ranked candidates; it does not evaluate their joint gain. These score units are neither accuracy percentages nor pp.

5.3 SENSITIVITY TO THE ROUTING FRACTION

Figure 2 shows a non-monotonic relationship between the routing fraction and downstream accuracy. The 20% fraction was fixed on development problems; the displayed evaluation sweep is a sensitivity analysis, not test-set tuning. Retaining 10% minimizes generation among these settings, whereas 20% gives the highest accuracy. Moving to 50% increases generation while reducing accuracy. Thus retaining more candidates does not necessarily yield more useful context.

Scorer architecture. The comparison of linear, four-layer MLP, and two-layer MLP heads is shown in Figure 3. The two-layer head gives the strongest reported ranking results on both Lemma and TODO evaluations.

6 TRAINING-TIME TRANSFER

We fine-tune Qwen3-8B and Qwen3-32B on 166 instances generated by Qwen3-8B, and evaluate with and without external lemmas. The matched solution-only comparison uses the same 166 problems and complete reference solutions, LoRA configuration, optimization steps, and effective training-token budget. The difference is the utility-guided selection-and-use supervision described in Section 3.4. Evaluation is question-only: no external lemma list is supplied.

Table 5: **Matched solution-only SFT control.** Qwen3-8B accuracy in percent. AIME uncertainty is the sample standard deviation of three runs. MATH-500 is a separate transfer evaluation, with no reported run SD; the base model scores 97.4% on this benchmark.

Training	AIME run 1	Run 2	Run 3	AIME mean $\pm$ SD	MATH-500
Solution-only	67.9	68.6	68.8	68.4$\pm$0.5	97.1
Utility-SFT	79.2	80.1	80.4	**79.9$\pm$0.6**	**99.2**

Utility-SFT improves on matched solution-only SFT by approximately 11.5 pp (Table 5). On MATH-500, drawn from the MATH evaluation benchmark (Hendrycks et al., 2021; Lightman et al., 2023), which is not used in scorer or SFT data construction, the base, solution-only, and Utility-SFT accuracies are 97.4%, 97.1%, and 99.2%. Thus the improvement is not confined to the AIME evaluation.

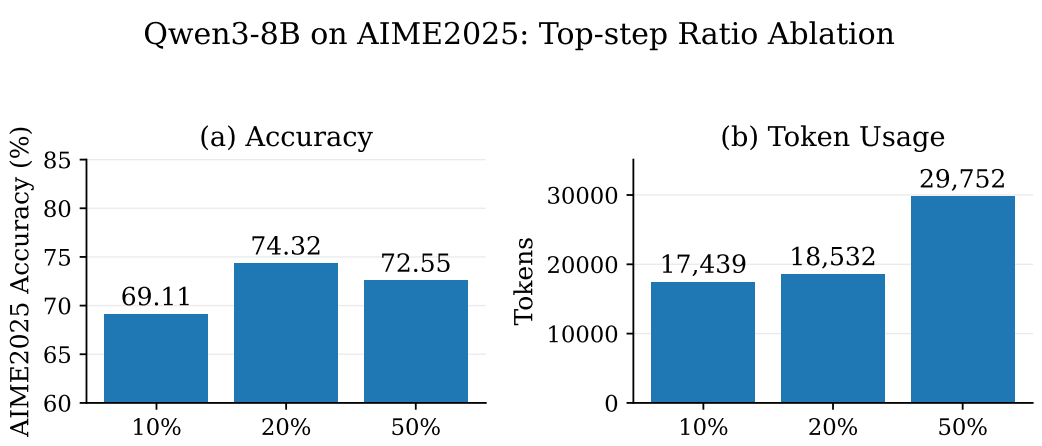

Figure 2: **Effect of the routing fraction on Qwen3-8B/AIME.** Both panels vary the fraction of candidate steps retained. **(a)** Accuracy peaks at 20%. **(b)** Mean generated tokens across sketching, extraction, and solving are lowest at 10%; retaining 50% costs more without improving accuracy. The 20% setting was fixed on development data. This sensitivity sweep omits the main pipeline's at-least-one-candidate safeguard.

Table 6: **Training-time transfer under varying lemma availability.** Direct Solve and Without Lemma use question-only evaluation. With Lemma supplies Qwen3-8B-generated lemmas without scorer selection; Lemma Usage evaluates lemma-conditioned solving on the step-usage benchmark. **Tok** counts reasoning and final-output tokens on AIME, and prompt plus generation tokens on the lemma benchmark. Appendix C illustrates the training rationale.

Backbone	Training	AIME				Our Lemma Benchmark			
		Direct Solve		With Lemma		Without Lemma		Lemma Usage	
		Acc ↑	Tok ↓	Acc ↑	Tok ↓	Acc ↑	Tok ↓	Acc ↑	Tok ↓
Qwen3-8B	None	68.16	18484.1	62.93	12226.6	79.07	2808.3	80.46	1983.8
	Scorer-guided SFT (166 ex.)	**80.36**	**14529.4**	**68.59**	**11033.6**	**80.23**	**2804.7**	**84.63**	**901.4**
Qwen3-32B	None	73.63	17252.0	63.60	10934.8	79.15	2651.4	78.15	1683.8
	Scorer-guided SFT (166 ex.)	**78.72**	**16055.0**	**70.26**	**10859.6**	**81.02**	**1752.8**	**82.00**	**1027.0**

Increasing Utility-SFT data from 42 to 83 to 166 examples raises AIME accuracy from 72.6% to 76.8% to 79.9±0.6%. At 332 and 664 examples, the reported results are $80.3 \pm 0.7\%$ and $79.6 \pm 0.9\%$, respectively. The curve plateaus with a small observed decline; it does not show monotonic improvement with more data or, by itself, establish overfitting.

Model scale and external hints. Table 6 reports a separate single-run transfer comparison for Qwen3-8B and Qwen3-32B. Utility-SFT improves question-only accuracy at both scales. Supplying unfiltered Qwen3-8B-generated lemmas, however, lowers accuracy relative to question-only evaluation for both base and fine-tuned models. Thus learning from utility annotations does not make arbitrary external context beneficial. This condition supplies hints without inference-time scorer filtering and must not be conflated with the routing pipeline in Table 1.

These single-run transfer results are separate from the matched three-run control. The lemma-benchmark columns use prompt-plus-generation tokens; details and the worked case study appear in Appendices B.3 and C.

7 Robustness, Generalization, and Cost Analysis

Supervision source. Changing the Qwen3-8B label judge from GPT-5 to DeepSeek-R1 or Qwen3-32B gives AIME accuracies of 74.32%, 73.66%, and 72.95% (Table 7a), a spread of 1.37 pp. All

Table 7: **Robustness, generalization, and cost.** Qwen3-8B evaluations of (a) the label judge, (b) the scorer-training data fraction, (c) zero-shot transfer, and (d) end-to-end latency. Pair-Acc is pairwise ranking accuracy; task accuracies are percentages.

(a) Judge ablation: at most 1.37 pp gap.

Judge	Pair-Acc	AIME
GPT-5 *(default)*	**0.7735**	**74.32**
DeepSeek-R1-0126	0.7632	73.66
Qwen3-32B *(OSS)*	0.7458	72.95

(b) Data scaling: 50% exceeds Effort.

Data fraction	Pair-Acc	AIME
25 %	0.6431	69.82
50 %	0.6982	71.32
100 %	**0.7735**	**74.32**

(c) Zero-shot MMLU-PRO (excluded from training).

Setting (MMLU-PRO)	Acc
Direct	72.10
Pipeline w/o utility filter	69.34
Pipeline + StepRouter	**77.25** (+5.15)

(d) AIME latency on a single A100.

Method	Time	Rel.
Direct	56.8 s	1.00×
StepRouter pipeline	**53.6 s**	**0.94**×

downstream endpoints use objective scoring. In the GPT-5 objective-label comparison, binary correctness-difference labels yield 94.44%, versus 97.22% with graded judge differences. The latter retain distinctions between partially useful steps that binary outcomes can collapse.

Data efficiency and generalization. At 25%, 50%, and 100% of the scorer data, AIME accuracy is 69.82%, 71.32%, and 74.32%. The 50% setting already exceeds Effort's 70.25%; the 25% setting exceeds Direct but not Effort. Zero-shot MMLU-PRO (Wang et al., 2024), absent from training, improves from 72.10% to 77.25%, while the unfiltered pipeline achieves 69.34% (Table 7c).

Alternative compute allocation. On Qwen3-8B/AIME, 3-, 5-, and 10-sample self-consistency reaches 70.82%, 72.1%, and 73.1%, using 61,326, 101,970, and 203,220 generated tokens. StepRouter reaches 74.32% with 18,532 tokens: the tested 10-sample configuration costs 10.97 times as much. The same-fraction PRM selector reaches 71.36%, compared with StepRouter's 74.32%.

End-to-end cost. On the reported single-A100 setup, latency is 53.6 seconds versus Direct's 56.8 seconds, comprising 20.2 seconds of upstream work and 33.4 seconds of final solving. The separate stage-instrumented token total is $1,998 + 3,304 + 14,727 = 20,029$ (Table 11), not a decomposition of the main run's 18,532 tokens. Discarded candidates are included in generation cost. Scoring emits no language-model tokens but still incurs compute; offline labeling and input-token charges are outside this metric.

8 Related Work

Reasoning with intermediate steps and CoT pruning. Chain-of-thought, self-consistency, and structured search improve reasoning through intermediate text (Shum et al., 2023; Wang et al., 2023; Wei et al., 2022; Yao et al., 2023a;b). A parallel line prunes or compresses traces using length, perceived difficulty, or confidence (Cui et al., 2025; Hsieh et al., 2023; Huang et al., 2024; Ning et al., 2023; Wang et al., 2025a; Xia et al., 2025; Xu et al., 2025b). Our focus is the usefulness of independently reusable steps: the effort prior is nearly uncorrelated with raw outcome gain, while learned utility improves budgeted selection.

Verification, process supervision, and reward modelling. Verifiers and reward models score solutions or trajectories (Cobbe et al., 2021; Uesato et al., 2022; Zhang et al., 2024), while process supervision provides intermediate correctness feedback (Lightman et al., 2023; Zhang et al., 2025). StepRouter differs in granularity and use: it estimates a reusable step's incremental utility for *pre-solve routing*, rather than treating correctness and usefulness as interchangeable. The PRM comparison tests these selection signals within our pipeline.

Table 8: **Positioning of StepRouter.** "Needs Step GT" refers to dense step-level annotations, beyond weak preferences derived from LLM-judged outcomes.

Method Family	Unit	Needs Step GT	Explicit Cost	Budgeted Subset	Where Used
Effort/length heuristics	step	No	Proxy	Yes	Inference
CoT pruning/compression	trace	No	Proxy	Indirect	Inference
Verifier / PRM selection	sol/trace	No	Optional	Indirect	Inference / Search
Process supervision	state/step	Often Yes	Optional	Indirect	Search / RL
Example-level data filtering	example	No	No	No	Training
StepRouter (ours)	step	No (weak pairs)	Proxy / Optional	**Yes (Top-20%)**	Inference & Training

Weak supervision and fine-grained preferences. Teacher–student distillation, self-training, and preference optimization often supervise complete outputs (Li et al., 2025; Yu et al., 2025). Finer-grained objectives address intermediate decisions (Wang et al., 2025b; Xu et al., 2025a). StepRouter obtains weak step-level preferences from judged with-step/no-step outcomes, then reuses them for both routing and data-efficient selection-and-use SFT.

Routing. Routing commonly dispatches inputs to experts or models (Chen et al., 2024b; Fedus et al., 2022; Shazeer et al., 2017). Our use is orthogonal: we route intermediate steps into a fixed backbone under a candidate-count budget.

9 Conclusion

Reasoning systems increasingly rely on intermediate steps, yet deciding which ones to reuse remains unreliable. We show that effort priors are poor proxies for usefulness and introduce StepRouter, a lightweight scorer trained from outcome-induced preferences. It routes the top-20% of a candidate pool and improves the tested backbones across mathematics, science, and code. Same-pool controls distinguish learned ranking from unfiltered context reuse and the tested compression strategy. The same utility signal transfers through SFT on 166 examples; matched solution-only training and three independent runs strengthen the evidence for its added value.

Limitations. The scorer ranks individual candidates and does not model their interactions. It can route only what the sketcher proposes: on AIME 2025 I-13, the tested backbones omitted a critical combinatorial lemma, so ranking could not supply the missing insight. All-step routing reaches only 65.29% on AIME (Table 2), underscoring that more context alone is insufficient. On Qwen3-8B, the MMLU-PRO gain of 5.15 pp is smaller than the AIME and LiveCodeBench gains of 6.16 and 23.23 pp. Costs remain setting-dependent, and judge-based supervision, step-only encoding, and small test sets bound the conclusions.

AI Use Statement

Generative AI produced sketches, steps, verifications, judgments, and SFT examples (Sections 3 and D). AI assisted writing, result interpretation, consistency and citation checks, and figure-label corrections. The authors are responsible for all final content.

Ethics Statement. Generated hints and judgments can be incorrect. API use and artifact redistribution must respect provider terms and licenses. Token savings do not measure environmental impact.

Reproducibility Statement. Method, metrics, and protocols appear in Sections 3–7; configurations, additional results, and prompts appear in Appendices A, B, and D, respectively.

REFERENCES

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024a. URL https://arxiv.org/abs/2402.03216.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024b. ISSN 2835-8856. URL https://openreview.net/forum?id=cSimKw5p6R. Featured Certification.

Nadezhda Chirkova, Thibault Formal, Vassilina Nikoulina, and Stéphane Clinchant. Provence: efficient and robust context pruning for retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=TDy5Ih78b4.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Changyi Cui, Yiqi Zhao, Zheng Wang, Xijia Tao, Le Ye, Yue Zhang, et al. Stepwise perplexity-guided refinement and information transfer for efficient reasoning, 2025. Accepted to Findings of ACL 2025.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. *arXiv preprint arXiv:2410.04707*, 2024. URL https://arxiv.org/abs/2410.04707.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, et al. Context length alone hurts LLM performance despite perfect retrieval. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23281–23298, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1264. URL https://aclanthology.org/2025.findings-emnlp.1264/.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, et al. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021. URL https://arxiv.org/abs/2103.03874.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, et al. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, Fan Yang, and Mao Yang. Fewer is more: Boosting math reasoning with reinforced context pruning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13674–13695, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.758. URL https://aclanthology.org/2024.emnlp-main.758/.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, et al. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, et al. NuminaMath. Hugging Face dataset repository, 2024. URL https://huggingface.co/datasets/AI-MO/NuminaMath-CoT.

Yang Li, Youssef Emad, Karthik Padthe, Jack Lanchantin, Weizhe Yuan, Thao Nguyen, et al. NaturalThoughts: Selecting and distilling reasoning traces for general reasoning tasks, 2025. URL https://arxiv.org/abs/2507.01921.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, et al. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. URL https://arxiv.org/abs/2305.20050. MATH-500 split released at https://github.com/openai/prm800k/tree/main/prm800k/math_splits.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, et al. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, et al. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025. URL https://arxiv.org/abs/2501.19393.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-Thought: Prompting LLMs for efficient parallel generation. *arXiv preprint arXiv:2307.15337*, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, et al. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, et al. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023. URL https://arxiv.org/abs/2311.12022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, and Geoffrey Hinton. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, et al. Large language models can be easily distracted by irrelevant context. In *ICML*, pp. 31210–31227, 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Kashun Shum, Shizhe Diao, and Tong Zhang. Automatic prompt augmentation and selection with chain-of-thought from labeled data. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 12113–12139, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.811. URL https://aclanthology.org/2023.findings-emnlp.811/.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, et al. OpenAI GPT-5 system card. *arXiv preprint arXiv:2601.03267*, 2025. URL https://arxiv.org/abs/2601.03267v1.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL https://arxiv.org/abs/2408.03314.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, et al. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7459–7482, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.394. URL https://aclanthology.org/2025.findings-emnlp.394/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, et al. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Yeyuan Wang, Dehong Gao, Rujiao Long, Lei Yi, Linbo Jin, Libin Yang, et al. ASPO: Adaptive sentence-level preference optimization for fine-grained multimodal reasoning, 2025b. URL https://arxiv.org/abs/2505.19100.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024. URL https://arxiv.org/abs/2406.01574.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs. *arXiv preprint arXiv:2502.12067*, 2025. URL https://arxiv.org/abs/2502.12067.

Huimin Xu, Xin Mao, Feng-Lin Li, Xiaobao Wu, Wang Chen, Wei Zhang, et al. Full-Step-DPO: Self-supervised preference optimization with step-wise rewards for mathematical reasoning, 2025a. URL https://arxiv.org/abs/2502.14356.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025b. URL https://arxiv.org/abs/2502.18600.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents, 2025c. URL https://arxiv.org/abs/2502.12110.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, et al. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. arXiv:2210.03629.

Bin Yu, Hang Yuan, Haotian Li, Xueyin Xu, Yuliang Wei, Bailing Wang, et al. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models, 2025. URL https://arxiv.org/abs/2505.03469.

Hanning Zhang, Pengcheng Wang, Shizhe Diao, Yong Lin, Rui Pan, Hanze Dong, et al. Entropy-regularized process reward model. *arXiv preprint arXiv:2412.11006*, 2024. URL https://arxiv.org/abs/2412.11006. Submitted Dec 2024; revised Sep 2025.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, et al. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025. URL https://arxiv.org/abs/2501.07301.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, et al. ProcessBench: Identifying process errors in mathematical reasoning, 2025. URL https://arxiv.org/abs/2412.06559.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

A Implementation and Evaluation Details

A.1 Scorer configuration

Table 9 gives the ranking-head configuration. The count of 647 refers to *candidate steps*, not preference pairs or distinct problems. The development corpus contains 126 problems before the problem-group split. The 166-example Utility-SFT corpus is a separate training resource; its example count must not be substituted for the scorer-development count.

Table 9: **Scorer configuration.** Architecture and training settings.

Component	Setting
Embedding backbone	**BAAI/bge-m3** dense embedding (d=1024), L2-normalized
Scorer $s(\ell)$	2-layer MLP: Linear(1024→128)+ReLU+Linear(128→1)
Objective (ranking)	$-\log \sigma(s(\ell^+) - s(\ell^-))$
Supervision	outcome-induced pairs (oracle-vs-random / top-vs-bottom)
Optimizer	Adam, lr $= 10^{-3}$, weight decay $= 10^{-3}$
Schedule	10 epochs, CPU, full-batch (1 update / epoch)
Split	80/20 by `problem_id` (seed=42) to avoid leakage
Implementation	embeddings normalized again in forward pass; on-disk embedding cache

Table 10: **Scorer validation.** Pairwise accuracy and recall on held-out problems; these are ranking metrics, not task accuracies.

Metric	Value
Pairwise Accuracy	0.7735
Recall@1	0.5556
Recall@3	0.6667
Recall@5	1.0000

The scorer's validation pairwise accuracy is 0.7735. The associated ranking diagnostic compares the highest empirical-utility candidate against alternatives within its problem. Recall@1, Recall@3, and Recall@5 are 0.5556, 0.6667, and 1.0000, respectively. These are validation ranking metrics, not downstream task accuracies or confidence bounds. The grouped split avoids placing candidate statements from one problem into both scorer training and validation.

A.2 Fine-tuning configuration

The reported SFT configuration uses LoRA rank 8, all target modules, learning rate 5×10^{-5}, five epochs, a cosine learning-rate schedule, and warmup ratio 0.1. The per-device batch size is 1, with gradient accumulation of 4, a 4,096-token training cutoff, and bf16 computation. The Qwen3-32B configuration uses DeepSpeed ZeRO-3. The solution-only control keeps the training hyperparameters and problem/reference-solution set matched. It omits the utility-selection annotations and explanations.

The reported Qwen evaluation recipe uses temperature 0.7, top-p 0.95, top-k 20, and a final-generation limit of 36,666 tokens. These are the open-weight-model settings; they are not asserted to be identical configurable parameters for GPT-5. The main inference recipe specifies sketch and extraction limits of 4,096 tokens each. Candidate extraction is instructed to retain intermediate information rather than the final answer.

A.3 Evaluation quantities

For n evaluated problems and m_i completions on problem i, let $r_{ij} \in \{0, 1\}$ indicate objective correctness. The macro-averaged per-problem accuracy is

$$\text{Acc} = \frac{100}{n} \sum_{i=1}^{n} \frac{1}{m_i} \sum_{j=1}^{m_i} r_{ij}. \tag{6}$$

The exact-count GPT-5 experiment has $n = 30$ and $m_i = 12$, so this becomes $100k/360$. The main evaluation and the newer exact-count experiment are kept in different tables rather than reinterpreting older percentages as counts from the newer run. A percentage-point difference is $a - b$ for accuracies expressed as percentages; a relative percentage change is $100(a - b)/b$. We use "pp" only for the former.

For the generation-cost metric, let $t_{i,\text{sketch}}$, $t_{i,\text{pool}}$, and $t_{i,\text{solve}}$ denote output tokens. The reported mean is

$$T_{\text{out}} = \frac{1}{n} \sum_{i=1}^{n} \left(t_{i,\text{sketch}} + t_{i,\text{pool}} + t_{i,\text{solve}} \right). \tag{7}$$

Direct solving has no sketch or pool-generation term. The metric includes generated candidate text even when it is not routed, but excludes input tokens, offline candidate verification and judging, and scorer-training cost. Scoring contributes zero *generated* tokens, not zero compute. A self-consistency baseline is charged for the multiple sampled solutions. Original tables retain their reported token precision; prose rounds mean token counts to whole tokens.

B Additional Experimental Results

B.1 Scorer architecture

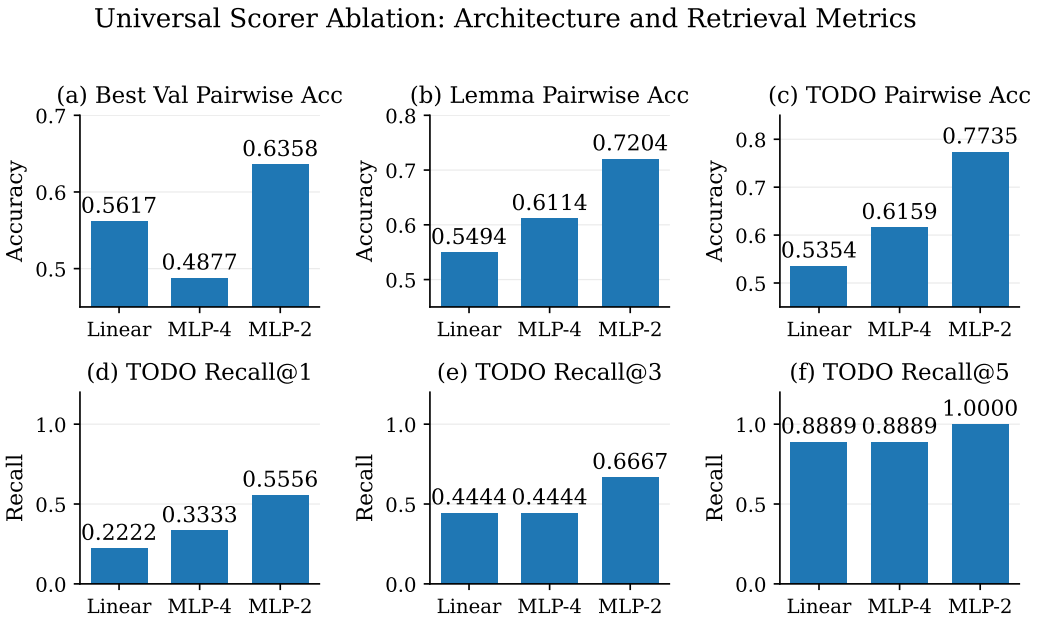

Figure 3: **Scorer architecture ablation for step ranking.** Each panel compares a linear head, a four-layer MLP (MLP-4), and a two-layer MLP (MLP-2). **(a)** Best validation pairwise accuracy; **(b–c)** pairwise accuracy on Lemma and TODO candidates; **(d–f)** TODO Recall@1, @3, and @5. Pairwise accuracy measures agreement with labeled preferences; Recall@k measures retrieval of the highest-utility candidate among the top k ranked steps. Higher is better in every panel. MLP-2 performs best across these ranking metrics, which are distinct from downstream task accuracy.

The two-layer head has the highest reported value in each panel. The best-validation metric in panel (a) is not interchangeable with the TODO metric in panel (c) or the oracle-ranking summary in Table 10. The comparison supports the tested lightweight architecture, but does not establish a universal depth optimum.

B.2 Stage-instrumented profiling

Table 11: **Stage-instrumented generation cost.** Mean generated tokens per problem for Qwen3-8B.

Stage	AIME	GPQA
Sketch	1998	2711
Lemma/Todo	3304	403
Final Output	14,727	4661
Total	20029	7775

The stage-instrumented profiling includes both AIME and GPQA. Its sums are $1,998 + 3,304 + 14,727 = 20,029$ and $2,711 + 403 + 4,661 = 7,775$, respectively. Neither total replaces the corresponding main-inference measurement in Table 1. Scoring adds no generated language-model tokens but still incurs embedding and ranking computation. The associated latency is specific to the reported single-A100 setup; it does not include amortized offline labeling cost.

B.3 Lemma-usage benchmark

We sample 200 Numina-Math (Li et al., 2024) problems, generate Qwen3-8B lemma pools, and remove cases with obvious answer leakage in the supplied lemmas, leaving 174 problems. This filtering targets an explicit prompt-leakage failure mode; it does not rule out pretraining exposure. The benchmark tests using a supplied pool, rather than learning an inference-time selection policy.

In Table 12, the Qwen results correspond to the fine-tuned, lemma-conditioned values in Table 6, not that table's base-model rows. For Qwen3-8B, both tables report 901.4 tokens. Here tokens include both prompt and generation; these costs are therefore not directly comparable to Table 1's generated-output-only totals.

C Qualitative Illustration of Utility-SFT

Let $d = n + 2$. Then $n + 3 = d + 1$ and $n^2 + 9 = d^2 - 4d + 13$. Consequently d divides $3(n + 3)(n^2 + 9)$ precisely when d divides 39. Since n is positive, the admissible divisors are $d = 3, 13, 39$, giving $n = 1, 11, 37$. This calculation checks the example's arithmetic; the matched study in Table 5, rather than the schematic, supplies the empirical comparison with solution-only training.

Table 12: **Lemma-usage benchmark.** Accuracy and mean total tokens (prompt + generation) on 174 filtered NUMINA-MATH problems. Qwen rows use the fine-tuned models.

Model	Accuracy (%)	Avg. Tokens
GPT-5-high	81.29	2171
DeepSeek-R1-0123	80.46	1445
Qwen3-8B	84.63	901.4
Qwen3-32B	82.00	1027

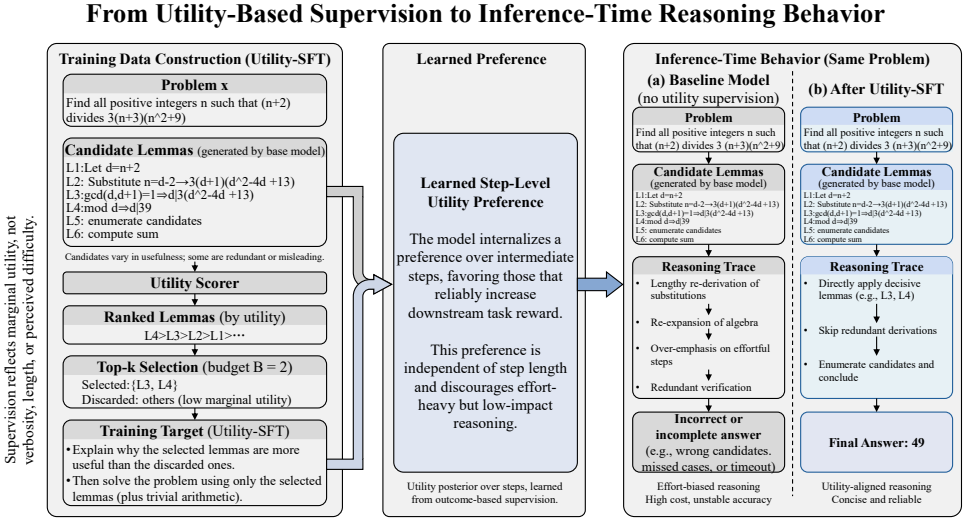

Figure 4: **From utility-based supervision to reasoning behavior. Left:** the scorer ranks candidate lemmas and selects L3 and L4 under an illustrative two-step budget; the SFT target explains the selection and solves the problem. **Center:** the proposed training rationale is to favor useful intermediate constraints rather than lengthy derivations. **Right:** the schematic contrasts redundant reasoning with direct reuse of the divisibility constraint $d \mid 39$. The valid positive integers are $n = 1, 11, 37$, with sum 49. These are illustrative reasoning paths, not measured model traces or evidence of an internal "utility preference"; Table 5 provides the matched empirical comparison.

D PROMPT ROLES AND CONSTRAINTS

The following specifications distinguish the deployment pipeline from the training-label construction pipeline. The extraction prompt below is an abbreviated rendering of the main inference prompt; its long numerical effort-scale examples are omitted for readability. The one-step verification and judging prompts are used offline. In particular, a requirement to use a candidate during label collection is not a claim that every routed candidate is guaranteed correct at deployment.

D.1 SKETCH AND CANDIDATE GENERATION

The main inference sketch prompt requests one concise paragraph of high-density, executable reasoning. The main extraction prompt is:

```
You are a domain-agnostic reasoning cost auditor.
Given a solution draft or scratch reasoning, extract reusable
key intermediate conclusions (lemmas) and estimate the additional
reasoning tokens required to expand each lemma into verifiable
reasoning, a proof, or an implementation.

effort_tokens is the estimated incremental reasoning needed to
expand the lemma. It is NOT the lemma text length or draft length.

- Extract reusable intermediate conclusions.
- Avoid trivial transformations and obvious paraphrases.
- Do NOT reveal the final answer, option, or code output.
- If no reusable conclusions exist, output [].
- Keep at most 16 items, merging redundant statements.
- Output only JSON:
  [{"lemma": string, "effort_tokens": integer}, ...]

Problem: {question}
Solution draft: {solution}
```

The effort baseline ranks the resulting estimated effort field. The learned selector instead ranks the same candidate statements using Equation 5; it does not use the estimated effort field as its inference score.

D.2 FINAL SOLVING WITH ROUTED CONTEXT

For the main AIME inference pipeline:

```
You will complete the final reasoning and produce the final answer
based on candidate lemmas extracted in the previous stage.
AIME-specific instruction: The final answer must be a single
integer from 000 to 999 (exactly three digits, including leading
zeros if needed). Do not output anything else.

Problem statement: {question}
Candidate lemmas (untrusted; for reference only): {lemma_block}
```

The coding prompt requests a standalone Python solution with correct input/output handling and instructs the solver to assess whether the provided TODO is relevant and logically sound before incorporating it. Benchmark correctness is determined by execution tests rather than by the utility judge.

D.3 VERIFICATION AND LABEL JUDGING

Verification measures the generated tokens needed to establish one candidate, without solving the whole problem:

```
You are given a math/programming problem and a lemma/TODO.
PROVE or VERIFY this lemma/TODO ONLY.

1. Do NOT solve the complete problem or provide the final answer.
2. Focus strictly on this lemma/TODO.
3. Be concise but rigorous.
4. For an algorithmic step, provide a minimal implementation
   sketch and/or a short correctness argument.

Problem: {problem}
Lemma/TODO: {lemma}
```

The label judge receives the original problem and a generated solution. It returns JSON with `score`, `confidence`, `error_type`, and a short `reason`. Only the scalar solution score enters Equation 2. Its rubric assigns 0 to empty or garbled output, intermediate scores to partial progress or incomplete reasoning, and the highest scores to correct, well-justified solutions. It also rewards clarity and efficiency; consequently the label is a judge-defined quality signal, not a pure correctness indicator. Benchmark evaluation remains objective exact-match, answer-choice, or execution scoring.

D.4 UTILITY-SFT FORMAT

```
You are a rigorous mathematical problem-solving assistant.

Problem: {problem}
Available Lemmas: {lemmas}

1. Use only lemmas marked (good) to solve the problem.
2. Explain why these lemmas are marked (good), for example,
   why they are directly relevant to the problem.
3. Solve the problem using the (good) lemmas.
4. End with: Final Answer: <answer>
```

Question-only evaluation removes the external lemma list and its annotations. Thus the no-hint evaluation is not scored on reproducing the training annotations; it is scored on the final answer.