SYSTEMATIC REUSE OF SHARED REASONING IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models often repeat the same intermediate reasoning across different queries. Prior work often reuses information between problems that are similar in topic or wording. We instead define relatedness by whether problems require the same intermediate reasoning—even when the queries might not be similar in wording or topic—and reuse that shared reasoning directly. Across four math benchmarks and MuSiQue, we find that shared reasoning structure is much more informative than surface semantic similarity. When two queries share an intermediate reasoning step, having the model solve that step once and reuse its answer for other queries improves accuracy by 7.2 percentage points while reducing total tokens used by 9.4%. We also study whether we can automatically identify the intermediate reasoning that different queries share. Although models can generate and match candidate reasoning steps, the generated steps often change the meaning of the original reasoning, causing reuse to fail even when the match is correct. Our results suggest that efficient multi-query inference should focus on identifying and preserving shared intermediate reasoning, with reliable automatic discovery remaining the key challenge.

1 INTRODUCTION

When we use large language models to answer a collection of queries, some queries may require overlapping intermediate reasoning, such as a shared fact or calculation. Models often recompute that step for each query, wasting inference compute. Prior work has reused information across queries by finding semantically related problems and transferring their responses or reasoning (Zuo et al., 2026). But semantic relatedness does not necessarily mean that two problems share the same reasoning steps.

We distinguish **semantic similarity** (overlapping wording or topic *in the query*) from **structural similarity** (shared intermediate reasoning steps *needed to solve the query*). For example, in MuSiQue—a multi-hop QA benchmark—one question might ask about a historical event and another about a sports team, yet both may first require answering "Pima County contains what city?" Conversely, two algebra problems may look closely related but require entirely different calculation steps. Figure 1 illustrates this general setting of shared reasoning across queries.

This raises our central question: **can we identify the intermediate reasoning that different queries actually share, solve it once, and reuse it instead of repeating the same work?** We study when this kind of structural similarity is useful, whether direct reuse improves accuracy and efficiency, and whether shared reasoning can be identified automatically. As a secondary comparison, we also test whether cross-query information can help decide how compute should be distributed across queries.

Our results show that structural similarity matters much more than semantic similarity. Across four math benchmarks and MuSiQue, queries that share actual reasoning steps show much stronger relationships than queries that merely overlap in wording or topic. More importantly, on Qwen3-8B, reusing a known shared intermediate step improves accuracy by 7.2 percentage points while reducing total tokens used by 9.4%. The main challenge is identifying shared reasoning across queries automatically. On 472 cases where the model-generated and benchmark sub-questions refer to the same intermediate reasoning step, Qwen3-8B answers the benchmark sub-question correctly 83.9% of the time but the model-generated one only 12.5%. This shows that model-generated sub-questions often fail to preserve the meaning of the reasoning steps they represent. Together, these

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.

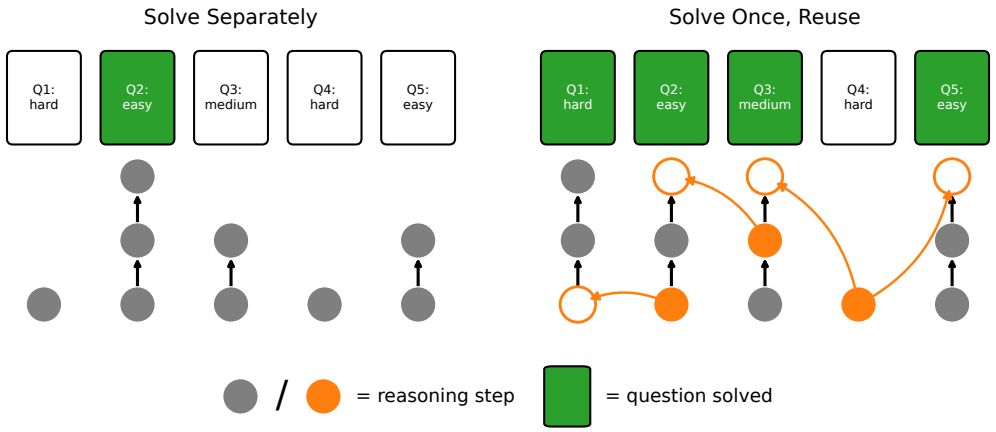


Figure 1: Two queries may look different even though they might require the same intermediate reasoning step. Instead of solving that step separately for each query, we ask whether it can be solved once and reused across queries.

results highlight both the value of reusing shared reasoning and the challenge of identifying those shared reasoning steps reliably.

Our contributions:

1. We distinguish semantic similarity from structural similarity, and show that structural similarity better captures useful cross-query relationships.
2. We show that reusing shared intermediate answers improves accuracy and token efficiency, while cross-query compute allocation does not reliably help.
3. We show that model-generated sub-questions can fail to preserve the meaning of the reasoning steps they represent, limiting automatic reuse.
4. We release 16,560 math traces and 6,400 MuSiQue traces with per-token log probabilities and simulated budgets.

2 Related Work

2.1 Cross-query reuse and relatedness

Several methods use information from one query to help answer another. Zuo et al. (2026) retrieve successful responses from queries with similar text and provide them as examples for later queries. Batch-of-Thought (Yang et al., 2026) processes related queries together, while batch prompting (Cheng et al., 2023) places several queries in the same prompt. Semantic caching stores previous answers and reuses them when a new query is sufficiently similar to an earlier one (Schroeder et al., 2026).

Other methods reuse more general information. Dynamic Cheatsheet (Suzgun et al., 2025) builds a growing memory of useful strategies and insights. Thought Propagation (Yu et al., 2024) generates related problems and uses their solutions or strategies to help solve the target problem. Metacognitive Reuse (Didolkar et al., 2025) turns repeated reasoning patterns into reusable behaviors. Sleep-time Compute (Lin et al., 2025) performs useful computation before the final query arrives, while Log-Augmented Generation (Chen et al., 2026) reuses reasoning from earlier tasks.

Logic-Augmented Generation (LAG) (Xiao et al., 2025) is especially close to our setting. It breaks complex queries into smaller sub-questions and stores high-confidence answers in a memory bank so that similar sub-questions can reuse earlier answers. This makes the generated sub-question the unit of reuse. Our work tests a stricter requirement for reuse across different queries: whether the queries truly share the same intermediate reasoning step, and whether the generated sub-question

preserves the meaning of that step. This connects to our distinction between semantic similarity and structural similarity, since queries need not be semantically similar to share reusable reasoning.

2.2 Test-time compute allocation

A separate line of work asks how much inference compute to spend on each query. CLEAR (Wan et al., 2026) estimates how much a query may benefit from additional tokens. Zhai et al. (2026) allocate a fixed total compute budget across queries. TALE (Han et al., 2025) and SelfBudgeter (Li et al., 2025) ask the model to estimate how much reasoning it needs, while Thinkless (Fang et al., 2025) chooses between shorter and longer reasoning modes. Zuo & Zhu (2026) similarly use estimates of problem difficulty to decide where to spend compute.

Other methods begin reasoning first and then decide whether to continue. Certaindex and Dynasor (Fu et al., 2024) use agreement between partial answers as a signal that a query may already be solved. DART (Lee et al., 2026) uses agreement between several short initial answers to decide whether more reasoning is needed. UAB (Nguyen et al., 2026) reallocates compute after an initial round. Related approaches use answer agreement or uncertainty (Fu et al., 2024; Lee et al., 2026), token-level uncertainty (Wang et al., 2025), or learned predictors (Zhang et al., 2025; Vilas et al., 2025; Davidov et al., 2026) to estimate whether more reasoning is useful.

Despite their differences, these methods mainly ask how much compute each query should receive. Plan-and-Budget (Lin et al., 2026) considers several subproblems together, but within one complex query. Our paper instead asks whether some of the reasoning needed for one query has already been performed for another. We therefore study compute allocation as a secondary way of using cross-query information, alongside directly reusing shared reasoning.

2.3 Decomposition and modular reasoning

Another line of work breaks a difficult problem into smaller subproblems. Least-to-Most Prompting (Zhou et al., 2023) first creates simpler subproblems and then solves them in sequence. Decomposed Prompting (Khot et al., 2023) similarly divides a problem into smaller tasks that can be handled by different prompts, models, or tools. These methods use decomposition mainly to make one difficult query easier to solve.

We similarly use decomposition to identify shared reasoning automatically by breaking each query into sub-questions and comparing them across queries. Our setting places an additional requirement on these generated sub-questions; the sub-questions must preserve the meaning of the reasoning steps they represent. If a generated sub-question does not faithfully represent what the original reasoning step asks, it can no longer be safely reused. In this paper, we show that failing to preserve the original reasoning step is a major obstacle to identifying reusable reasoning automatically.

3 Methods

3.1 Experimental setup

Our primary model is Qwen3-8B (Qwen Team, 2025) in thinking mode, with temperature 0.6, top-p 0.95, and top-k 20. All queries use a maximum generation length of $31{,}908$ tokens, chosen from the model's $32{,}768$-token context window after leaving room for the prompt and final answer. Additional model details, including our rejected Qwen2.5-Math setup, are reported in Appendix A.6.

We use four math benchmarks and MuSiQue to study different types of relationships across queries. The math benchmarks are GSM-Symbolic (Mirzadeh et al., 2024), MATH (Hendrycks et al., 2021), AIME (Mathematical Association of America, 2026), and OlympiadBench (He et al., 2024), with 400, 400, 60, and 175 problems respectively, for 1,035 problems total. GSM-Symbolic contains multiple problems generated from the same template, giving us a controlled example of queries that share reasoning structure.

For explicit shared intermediate reasoning, we use MuSiQue (Trivedi et al., 2022). Its multi-hop questions are built from smaller single-hop sub-questions, so different final questions can require the same intermediate answer. We organize the 400 problems into 40 groups of 10 queries, where

each group is treated as one set of queries for our cross-query experiments. Across these groups, there are 543 unique sub-questions, 236 of which are used by more than one query within the same group, meaning that those queries share an intermediate reasoning step.

3.2 Measuring shared reasoning across queries

We first test whether semantic similarity and structural similarity capture the same relationships across queries. On the math benchmarks, we compare several ways of pairing problems. We pair GSM-Symbolic problems generated from the same template, problems with similar text embeddings, MATH problems with the same subject and difficulty level, and random problems. Same-template pairs give us a controlled form of structural similarity because they are generated from the same underlying problem structure, even when their specific values or wording differ. Text-based and subject-based pairs instead capture semantic similarity.

For each type of pair, we ask whether the difficulty of one query helps predict the difficulty of the other. We also test whether knowing the paired query's outcome helps predict the target query's difficulty beyond what the target query alone tells us. We measure this additional predictive value as

$$\Delta R^2 = R^2(\text{target query + paired-query outcome}) - R^2(\text{target query only}). \tag{1}$$

A larger ΔR^2 means that knowing the outcome of the paired query provides more information beyond the target query itself.

MuSiQue gives us a more direct measure because the benchmark tells us which intermediate sub-questions each final question requires. Let $\mathcal{S}_i$ denote the set of intermediate sub-questions required by query i. We define the amount of shared reasoning between queries i and j as

$$S_{ij} = |\mathcal{S}_i \cap \mathcal{S}_j|. \tag{2}$$

We then test whether query outcomes become more closely related as S_{ij} increases.

We also test whether sharing the full solution to one related query is enough to help another. For related math queries, we provide one query's complete solution when solving the other and compare performance with solving the target query alone. This lets us compare reuse of a full reasoning path with reuse of the specific intermediate step that two queries share.

3.3 Reusing shared intermediate reasoning

We next test whether shared intermediate reasoning can be reused directly. For each MuSiQue query, we provide answers to the shared sub-questions it requires before asking the model to solve the final question. We compare benchmark-provided sub-answers, model-generated sub-answers, and controls containing unrelated facts of the same length. The control tests whether any improvement comes from the shared information itself rather than simply from adding more text to the prompt.

For this experiment, we assume we already know which sub-questions are shared across queries. We answer each shared sub-question once with the model, then reuse that answer for every query that requires it.

For a shared sub-question s used by a group of queries G, solving every query independently has token cost

$$C_{\text{indep}}(G) = \sum_{q_i \in G} c(q_i), \tag{3}$$

where $c(q_i)$ is the number of tokens used to solve query i. With reuse, the shared sub-question is answered once, giving

$$C_{\text{reuse}}(G) = c(s) + \sum_{q_i \in G} c(q_i \mid s), \tag{4}$$

where $c(s)$ is the one-time cost of answering the shared sub-question and $c(q_i \mid s)$ is the cost of solving query i with that answer provided. We compare these total costs together with final accuracy.

Because shared intermediate facts can sometimes contain the final answer itself, we remove any such fact before testing reuse. After this filtering, 375 of the 400 MuSiQue problems have at least one usable shared fact. The headline reuse results are computed over all 400 problems; the remaining 25 receive no injected shared fact and therefore contribute no reuse effect. Analyses broken down by the number of shared facts use the 375 problems with at least one usable fact (Appendix A.1).

3.4 Identifying shared reasoning automatically

The previous experiment assumes that we already know which intermediate reasoning steps are shared. We next ask Qwen3-8B to break each query into sub-questions, embed them using MiniLM-L6-v2, and compare them across queries within the same 10-query MuSiQue group. Two generated sub-questions are treated as a possible match when their cosine similarity is at least τ, and connected matches spanning at least two queries are grouped together. We test cosine-similarity thresholds from $\tau = 0.50$ to $\tau = 0.95$; Appendix A.4.1 gives the full matching and evaluation procedure.

Because MuSiQue provides benchmark sub-questions, we evaluate whether these automatically identified matches correspond to the same intermediate reasoning step by measuring precision and recall against the benchmark sub-question identities. We separately test whether generated sub-questions preserve what the original reasoning step is asking: in 472 cases where the generated and benchmark sub-questions correspond to the same step, we ask Qwen3-8B to answer both versions.

3.5 Cross-query compute allocation

As a secondary comparison to reusing shared intermediate answers, we test whether information across queries can help decide how much compute each query should receive. Suppose query i receives a budget of k_i tokens and the total available budget across all N queries is B. Let $a_i(k_i)$ denote the accuracy of query i when given k_i tokens. The goal is to choose the budgets to maximize average accuracy,

$$\max_{k_1,\ldots,k_N} \frac{1}{N} \sum_{i=1}^{N} a_i(k_i) \qquad \text{subject to} \qquad \sum_{i=1}^{N} k_i \leq B. \tag{5}$$

Trace corpus and budget simulation. To compare different allocation methods fairly, we generate a fixed set of reasoning traces and reuse the same traces across all methods. For each query, we sample 16 traces at the largest token budget, giving 16,560 math traces and 6,400 MuSiQue traces. To simulate a budget of k tokens, we keep only the first k tokens of a stored trace. If the model has not finished reasoning, we ask it to produce a final answer using at most 64 additional tokens. We evaluate $k \in \{128, 256, 512, 1024, 2048, 4096, 8192, 16384, 31908\}$. We validate both the forced-answer procedure and our use of truncated traces in Appendix A.1.

Allocation methods. On the math benchmarks, we compare twelve methods (Figure 3), including uniform allocation, an oracle that knows how much each problem benefits from additional tokens, CLEAR-style methods (Wan et al., 2026) based on predicted difficulty, Dynasor- and UAB-style methods (Fu et al., 2024; Nguyen et al., 2026) based on early answer agreement, and methods that use information from a structurally related query. On MuSiQue, we evaluate nine similar methods that use information from other queries in the same 10-query group. We compare them with a baseline that decides each query's budget independently. This tests whether information from other queries improves compute allocation beyond treating each query separately.

3.6 Evaluation protocol

Problem-disjoint evaluation. For experiments where we train a separate model to predict query difficulty or reasoning progress, we split the data so that all reasoning traces from the same problem stay in the same cross-validation split. Otherwise, different attempts at the same problem could appear in both the training and evaluation sets, allowing the model to recognize which problems are

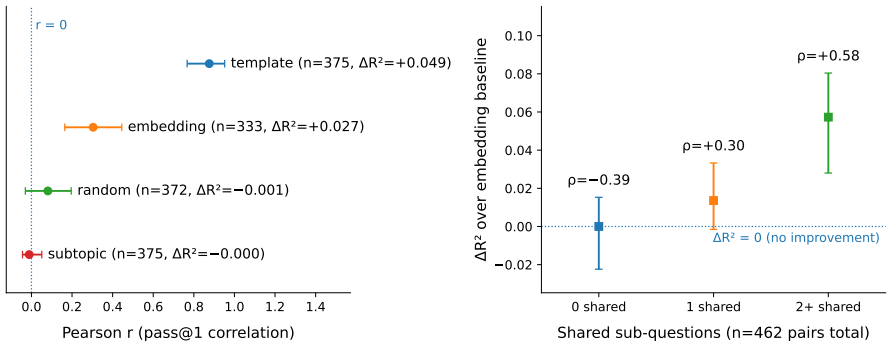


Figure 2: Shared reasoning is more informative than semantic similarity. (left) Outcomes are strongly correlated for problems generated from the same template, but much less so for semantically similar or random pairs. (right) Predictive value increases as two MuSiQue queries share more intermediate sub-questions.

usually easy rather than learn whether a particular reasoning run is going well. For MuSiQue, all queries from the same 10-query group stay in the same split. When we use information from other queries, we exclude the target query itself so that any improvement comes from the other queries rather than from information about the target being reused in its own prediction.

Compute accounting. Any tokens used to estimate how much compute a query should receive are included in the total token cost. If that decision uses m partial reasoning traces of length k, those traces cost

$$C_{\text{probe}} = mk. \tag{6}$$

If one of the probe traces continues from k to $k' > k$, we count the additional $k' - k$ tokens. Tokens spent on probe traces that are later discarded still count toward the total. The traces used to decide how much compute to allocate are separate from the traces used to produce the final answers whose accuracy we evaluate.

4 Results

We first test whether structural similarity is more informative than semantic similarity. We then compare two ways of using these relationships: (i) allocating compute across queries and (ii) reusing shared intermediate answers. Finally, we test whether reusable shared reasoning across queries can be identified automatically.

4.1 Structural similarity is more informative than semantic similarity

The left panel of Figure 2 compares four ways of pairing math problems. Problems that share the same template have strongly correlated outcomes ($r = 0.876$, 95% CI $[0.767, 0.952]$). Pairs with similar text embeddings have a much smaller correlation ($r = 0.304$), while random pairs ($r = 0.081$) and pairs from the same subtopic ($r = -0.011$) show essentially no relationship. This suggests that shared problem structure is much more informative than semantic similarity alone.

The same pattern remains after accounting for what the target query already tells us about its own difficulty. Adding the outcome of a same-template problem increases predictive power by $\Delta R^2 = +0.049$, compared with $+0.027$ for an embedding-nearest problem and essentially zero for random or same-subject pairs, shown in the left panel of Figure 2. Thus, semantic similarity provides some information, but substantially less than shared reasoning structure.

MuSiQue lets us test this more directly because we know which intermediate sub-questions two queries share. The relationship between query outcomes grows with this overlap; $\rho = -0.39$,

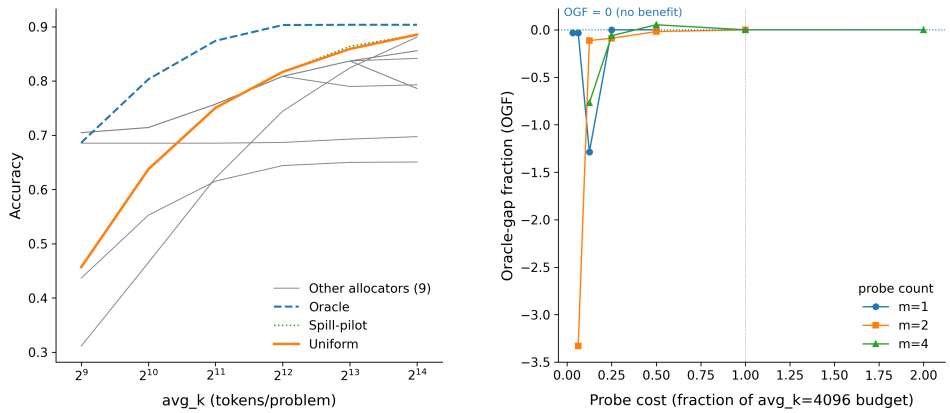


Figure 3: Using cross-query information does not improve compute allocation. (left) None of the 10 non-oracle, non-uniform methods reliably outperforms uniform allocation. (right) Signals based on partial reasoning become useful only after substantial compute has already been spent.

$+0.30$, and $+0.58$ for pairs sharing 0, 1, and 2+ sub-questions, respectively, as shown in the right panel of Figure 2. This pattern shows that the model's success on one query is more informative about its success on another when the two queries share more intermediate reasoning.

We also test whether simply providing the full solution to a related math problem helps. In a pilot on 99 problems, the effect is small and not significant at $k = 512$ and $k = 1024$, and becomes negative at larger budgets; -2.0 points at $k = 2048$ and -1.6 points at $k = 4096$ (Appendix Table 16). This suggests that providing an entire related solution is not a substitute for reusing the specific intermediate reasoning that two queries share.

4.2 Cross-query information does not improve compute allocation

As a secondary comparison, we test whether cross-query information can help decide how much compute each query should receive. The left panel of Figure 3 shows that the possible gain from unequal allocation shrinks quickly as the available budget grows. An oracle that knows exactly how much each problem benefits from additional tokens beats uniform allocation by 22.9 points at 512 tokens/query, but by only 1.8 points at 16,384 tokens/query.

Practical methods face another problem. Obtaining a reliable signal also costs compute. At $k = 256$, 55.2% of cases where four partial reasoning traces agree are still wrong. The error rate falls below 10% only at $k = 2048$, when the four traces have already cost 8,192 tokens per query (Appendix Figure 6). In other words, the signal becomes reliable only after much of the compute it was meant to allocate has already been spent.

A signal may tell us that a problem is generally easy, but that does not mean it can tell whether a particular reasoning attempt is already correct. Under problem-disjoint evaluation, the signals we test do not reliably make this second distinction (Appendix Figure 7; Table 6). The end-to-end results reflect these limitations. None of the 10 non-oracle, non-uniform math allocation methods reliably beats uniform allocation (Figure 3). On MuSiQue, none of the 9 methods using information from other queries improves over deciding each query's budget independently (Appendix Table 18). Across the practical methods we test, cross-query information therefore does not reliably improve compute allocation.

4.3 Reusing shared intermediate answers improves accuracy and efficiency

Instead of using shared structure to decide where to spend compute, we directly reuse the intermediate answers that multiple queries need. Table 1 summarizes the main result.

Using benchmark-provided shared answers raises accuracy from 0.495 to 0.574, a gain of 8.0 points. Most of this gain remains when Qwen3-8B generates the shared answers itself: accuracy rises from

Table 1: Reusing a shared intermediate answer improves accuracy on 400 MuSiQue problems. Gains are relative to the matched token-length controls.

Condition	Accuracy	Control	Gain
No reuse	0.499	–	–
Benchmark-provided answer	0.574	0.495	+8.0 pts
Model-generated answer	0.568	0.497	+7.2 pts

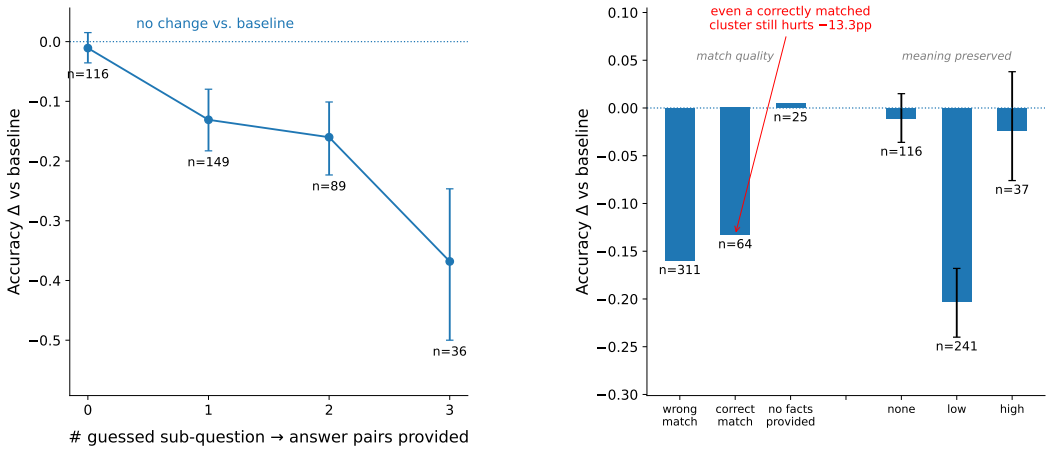


Figure 4: Automatically matching sub-questions across queries and providing the guessed shared answer hurts accuracy. (left) Accuracy gets worse as more guessed sub-question → answer pairs are provided (95% CI). (right) Even a verified-correct match still hurts, because the model's rewritten sub-question can drift from its original meaning; the first three bars are point estimates only (no CI), the last three include 95% CIs.

0.497 to 0.568, a gain of 7.2 points (Table 1). Because adding an unrelated fact of the same length leaves accuracy near the no-reuse baseline (0.495 and 0.497 vs. 0.499), the improvement comes from the shared answer itself rather than from simply adding more text (Section 3.3; Table 1).

Reuse also reduces total compute. Generating the shared answers costs an additional 1.7% of overhead, but each answer can then be used by every query that needs it. As a result, total token use per 10-query group falls by 9.4%. Reuse therefore improves both accuracy and token efficiency.

When the shared sub-question is known in advance, Qwen3-8B answers it correctly 83.1% of the time (Appendix A.4). Incorrect shared answers do not produce a significant reduction in downstream accuracy. Because the sub-question itself is still correct, the downstream model can sometimes recover from an incorrect intermediate answer using the surrounding context. The reuse gain also remains similar after accounting for differences in how much room different problems have to improve (Appendix Figure 9).

These results establish that shared intermediate reasoning is useful once we know which reasoning steps are shared across various queries. The remaining question is whether those steps can be identified reliably without benchmark annotations.

4.4 Automatic discovery fails when generated sub-questions change meaning

So far, we have assumed that we already know which sub-questions are shared across queries, so their answers can be reused. We next test whether the model can itself identify these shared sub-questions automatically by generating intermediate sub-questions for each query and matching similar ones across queries.

Our results show that automatic discovery performs poorly. At $\tau = 0.70$, which gives the best matching F_1 score, accuracy drops from 0.499 to 0.353 (-14.5 points). Even at $\tau = 0.95$, where

matching precision rises to 0.916, accuracy is only 0.372 (-12.7 points) (Figure 4; Appendix Table 12). This suggests that improving matching precision alone is not enough.

To test whether the failure comes from incorrect matching or from the generated sub-questions themselves, we focus on 472 cases where the matcher identifies the correct intermediate reasoning step. For each case, we ask Qwen3-8B to answer both the original benchmark sub-question and the model-generated version. The model answers the benchmark sub-question correctly 83.9% of the time, but the generated version only 12.5% of the time (Appendix Table 13). Moreover, even when all matched sub-questions are correct matches, using their answers still reduces downstream accuracy by 13.3 points (Figure 4).

The problem is that a generated sub-question can refer to the intended reasoning step without faithfully preserving what that step asks. It may drop an important qualifier, lose the referent of a pronoun, or omit part of the required reasoning. These changes can make the generated sub-question look similar enough to the benchmark sub-question to be matched, even though it is actually asking something different. The main bottleneck is not just matching the right sub-questions; the generated sub-questions also need to preserve what the original reasoning step is actually asking.

5 Conclusion

Our results suggest that the main opportunity in multi-query inference lies in identifying and reusing shared intermediate reasoning across queries. On four math benchmarks and MuSiQue, structural similarity (queries sharing intermediate reasoning steps) is substantially more informative about whether one query can help another than semantic similarity (queries sharing words or topics). When the shared intermediate sub-questions are known, answering each once and reusing its model-generated answer improves Qwen3-8B accuracy by 7.2 percentage points while reducing total token use by 9.4%.

We also find that how shared reasoning is used matters. Across the practical allocation methods we test, using information from other queries to decide how much compute each query should receive does not reliably improve accuracy. Directly reusing a shared intermediate answer, by contrast, improves both accuracy and efficiency. This suggests that shared structure is most useful for avoiding repeated reasoning rather than for redistributing reasoning compute across queries.

To discover shared reasoning automatically, we first ask Qwen3-8B to break each query into intermediate sub-questions, then match similar generated sub-questions across queries. Our experiments reveal that even when this matching step identifies the correct reasoning step, the generated sub-question may not faithfully capture what that step is asking. In 472 such correctly matched cases, Qwen3-8B answers the original benchmark sub-question correctly 83.9% of the time, but the generated sub-question only 12.5% of the time. This shows that reliable reuse requires not only finding the right shared reasoning step, but also representing it faithfully.

Broader impact. Shared reasoning could make language-model inference more efficient by solving repeated intermediate steps once and reusing the result, reducing inference cost while potentially improving accuracy. However, an incorrect intermediate answer could be reused across multiple queries and spread the same error. Our automatic-discovery experiments reveal another source of risk: even when the correct reasoning step is matched, the generated sub-question may not faithfully represent what that step is asking. Practical systems should therefore verify both that the correct reasoning step has been identified and that the reused information correctly answers it.

Limitations and future work. Our experiments use only one primary model, Qwen3-8B, and focus on math and multi-hop question answering, where shared reasoning can be measured using templates or benchmark-provided sub-questions. It remains unclear whether the same results hold for other models or domains, such as code generation, scientific reasoning, or queries that share source documents. Our automatic-discovery pipeline also uses a single decomposition method and embedding-based matcher, so other approaches may identify and represent shared reasoning more reliably. Future work should develop methods that can identify, verify, and reuse shared reasoning automatically without relying on benchmark annotations to identify which reasoning steps are shared.

AI USE STATEMENT

Generative AI tools were used during manuscript preparation for writing and editing support, literature search and organization, research brainstorming, and discussion of experimental design and result presentation. They were also used to assist with LaTeX, formatting, and code-related tasks. All experimental choices, analyses, citations, reported results, and final manuscript text were checked by the authors.

ETHICS STATEMENT

This work studies how intermediate reasoning can be shared and reused across multiple language-model queries. Such reuse may reduce inference cost, but it can also propagate an incorrect intermediate result to multiple downstream queries. Our automatic-discovery results further show that generated sub-questions may fail to faithfully represent the reasoning step they are intended to capture. The methods studied here should therefore not be treated as guarantees of correctness, and practical systems should verify shared reasoning before reusing it in settings where errors may have meaningful consequences.

REPRODUCIBILITY STATEMENT

The main text and appendices describe the datasets, model and generation settings, evaluation protocols, data splits, token accounting, and construction of the reuse and allocation experiments. We also report the random seed and validation procedures used in our experiments. Code, experiment configurations, and scripts for reproducing the reported results and figures are released at `https://anonymous.4open.science/r/shared-reasoning-llm/`. All linked materials are anonymized for double-blind review.

REFERENCES

Peter Baile Chen, Yi Zhang, Dan Roth, Samuel Madden, Jacob Andreas, and Michael Cafarella. Log-augmented generation: Scaling test-time reasoning with reusable computation. In *International Conference on Learning Representations*, 2026.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. Batch prompting: Efficient inference with large language model APIs. In *Proceedings of EMNLP: Industry Track*, 2023. arXiv:2301.08721.

Hen Davidov, Nachshon Cohen, Oren Kalinsky, Yaron Fairstein, Guy Kushilevitz, Ram Yazdi, and Patrick Rebeschini. Knowing when to quit: A principled framework for dynamic abstention in LLM reasoning. *arXiv preprint arXiv:2604.18419*, 2026.

Aniket Didolkar, Nicolas Ballas, Sanjeev Arora, and Anirudh Goyal. Metacognitive reuse: Turning recurring LLM reasoning into concise behaviors. *arXiv preprint arXiv:2509.13237*, 2025.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: LLM learns when to think. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.13379.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling LLM reasoning with certaindex. *arXiv preprint arXiv:2412.20993*, 2024.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics (ACL)*, 2025. arXiv:2412.18547.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*, 2023.

Jungseob Lee, Seongtae Hong, Seungjun Lee, Jaehyung Seo, Junyoung Son, Sugyeong Eo, Chanjun Park, Hyeongju Park, Hyeonseok Moon, and Heuiseok Lim. Dart: Draft-agreement routing for training-free adaptive thinking budgets in hybrid reasoning models. *arXiv preprint arXiv:2606.23181*, 2026.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. Selfbudgeter: Adaptive token allocation for efficient LLM reasoning. *arXiv preprint arXiv:2505.11274*, 2025.

Junhong Lin, Xinyue Zeng, Jie Zhu, Song Wang, Julian Shun, Jun Wu, and Dawei Zhou. Plan and budget: Effective and efficient test-time scaling on reasoning large language models. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.16122.

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

Mathematical Association of America. American invitational mathematics examination. MAA Invitational Competitions, 2026. Accessed September 2026.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Manh Nguyen, Sunil Gupta, and Hung Le. Uncertainty-aware budget allocation for adaptive test-time reasoning. *arXiv preprint arXiv:2605.26849*, 2026.

Qwen Team. Qwen3 technical report, 2025. arXiv:2505.09388.

Luis Gaspar Schroeder, Aditya Desai, Alejandro Cuadron, Kyle Chu, Shu Liu, Mark Zhao, Stephan Krusche, Alfons Kemper, Matei Zaharia, and Joseph E. Gonzalez. vcache: Verified semantic prompt caching. In *International Conference on Learning Representations*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022.

Martina G. Vilas, Safoora Yousefi, Besmira Nushi, Eric Horvitz, and Vidhisha Balachandran. Tracing the traces: Latent temporal signals for efficient and accurate reasoning. *arXiv preprint arXiv:2510.10494*, 2025.

Xu Wan, Speed Zhu, Jianwei Cai, Guang Chen, XiMing Huang, Wiggin Zhou, and Mingyang Sun. The shadow price of reasoning: Economic perspective on optimal budget allocation for LLMs. *arXiv preprint arXiv:2606.03092*, 2026. ICML 2026.

Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. Entropy after think for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*, 2025.

Yilin Xiao, Chuang Zhou, Qinggang Zhang, Su Dong, Shengyuan Chen, and Xiao Huang. Lag: Logic-augmented generation from a cartesian perspective. *arXiv preprint arXiv:2508.05509*, 2025.

Xuan Yang, Furong Jia, Roy Xie, Xiong Xi, Hengwei Bian, Jian Li, and Monica Agrawal. Batch-of-thought: Cross-instance learning for enhanced LLM reasoning. *arXiv preprint arXiv:2601.02950*, 2026.

Junchi Yu, Ran He, and Rex Ying. Thought propagation: An analogical approach to complex reasoning with large language models. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.03965.

Zhiyuan Zhai, Bingcong Li, Bingnan Xiao, Ming Li, and Xin Wang. Adaptive test-time compute allocation for reasoning LLMs via constrained policy optimization. *arXiv preprint arXiv:2604.14853*, 2026.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2506.12721.

Bowen Zuo, Dongruo Zhou, and Yinglun Zhu. Adaptive test-time compute allocation with evolving in-context demonstrations. In *Findings of the Association for Computational Linguistics (ACL)*, 2026. arXiv:2604.21018.

A Appendix

A.1 Setup and corpus

Table 2 gives the full dataset statistics for the 1,435 problems used in our trace corpus. Average output length varies substantially across datasets, from 2,882 tokens on GSM-Symbolic to 16,113 on AIME. This is why we compare methods using explicit token budgets rather than a fixed number of reasoning steps: the same token budget covers very different amounts of reasoning across datasets.

The table also reports how often the model was still reasoning when it reached the 31,908-token maximum. This happens for 0.0% of GSM-Symbolic traces, 0.4% of MATH traces, 2.7% of OlympiadBench traces, and 9.1% of AIME traces. Thus, the maximum budget is sufficient for almost all traces, although some AIME traces may have benefited from more tokens. This means that the largest-budget accuracy on AIME may slightly understate what could be achieved with a still larger budget.

Table 3 gives additional statistics for MuSiQue. Within the 40 groups, there are 119 query pairs sharing no sub-questions, 163 sharing one, and 180 sharing two or more. This variation within the same groups is what allows us to measure how query outcomes change as the amount of shared reasoning increases in Figure 2.

The table also shows that 375 of the 400 MuSiQue problems retain at least one usable shared fact after we remove facts that contain the final answer. The headline reuse results include all 400 problems; the remaining 25 receive no injected shared fact and therefore contribute no reuse effect. Analyses broken down by the number of shared facts use the 375 problems with at least one usable fact.

Table 4 records the exact model and generation settings used to produce the traces. We use one fixed model revision and one fixed seed throughout the corpus. Therefore, the confidence intervals in this paper capture variation across problems and traces under this fixed setup; they do not measure how much the results would change if the entire corpus were regenerated with different seeds.

Table 5 checks the forced-answer procedure used when a stored trace is cut off at a given token budget. At the full token budget, forced-answer accuracy differs from natural completion by at most 0.5 percentage points on every dataset. The largest differences occur on AIME and OlympiadBench, which are also the datasets where more traces reach the maximum generation length. Overall, this check suggests that asking the model for a short final answer after truncation introduces little change at large budgets.

Figure 5 compares the two procedures across token budgets. The points lie on or above the diagonal, especially at smaller budgets, meaning that truncating a longer stored trace tends to overestimate the accuracy obtained when the model is given the same small budget from the start. The gap becomes smaller at larger budgets. We therefore treat simulated accuracy at small token budgets as an approximate and somewhat optimistic estimate.

A.2 Checks for compute allocation

Figure 6 expands the result summarized in Section 4.2. At $k = 256$, 55.2% of problems for which all four partial traces agree are still wrong. This falls to 32.1% at $k = 512$, 15.7% at $k = 1024$,

Table 2: Corpus statistics, 1435 problems total. 16 traces/problem by design, not separately reported per dataset.

Dataset	n	Mean tok	Median tok	Cap-hit %	pass@1	pass@16
GSM-Symbolic	400	2882	1989	0.0	0.882	0.948
MATH	400	4882	3286	0.4	0.969	0.993
OlympiadBench	175	10747	7710	2.7	0.812	0.897
AIME	60	16113	13605	9.1	0.701	0.9
MuSiQue	400	1339	1070	0.1	0.499	0.692
Total	1435					

Table 3: MuSiQue group statistics and shared-sub-question overlap. Pair counts are computed within groups, so a group can contribute pairs to multiple overlap categories.

Metric	Value
Problems	400
Groups	40
Problems per group	10
Traces per problem	16
Unique sub-questions	543
Shared sub-questions	236
Problems with usable shared facts	375
Problems with no usable shared facts	25
Maximum output tokens	31,908
Mean thinking tokens/query	1,339
Query pairs sharing 0 sub-questions	119 pairs / 13 groups
Query pairs sharing 1 sub-question	163 pairs / 19 groups
Query pairs sharing 2+ sub-questions	180 pairs / 20 groups

Table 4: Generation configuration used for all experiments.

Field	Value
Model	Qwen/Qwen3-8B
Revision	b968826d9c46dd6066d109eabc6255188de91218
Data type	bfloat16
Thinking mode	Enabled
Maximum model length	32,768
Maximum output tokens	31,908
Samples per problem	16
Temperature / top-p / top-k	0.6 / 0.95 / 20
Seed	20260910

Table 5: Forced-answer validation: graded vs forced pass@1 at the full token cap, per dataset.

Dataset	n	Graded pass@1	Forced pass@1 (max k)	Diff
aime	60	0.701	0.706	0.005
gsm_symbolic	400	0.882	0.882	0.0
math	400	0.969	0.97	0.001
olympiadbench	175	0.812	0.817	0.005

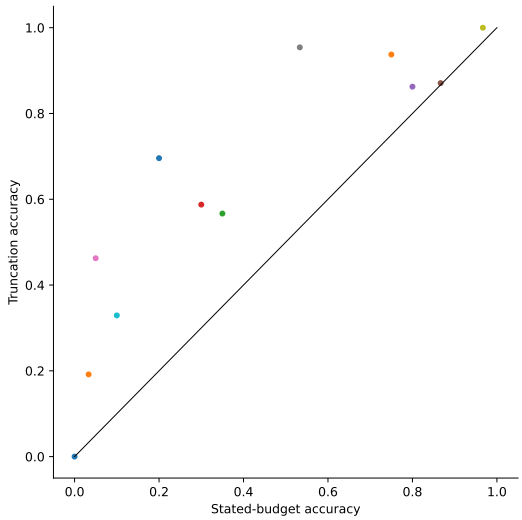


Figure 5: Truncation accuracy compared with accuracy when the model is given the same token budget from the start. Points on the diagonal indicate agreement; points above the diagonal mean that truncation gives higher accuracy.

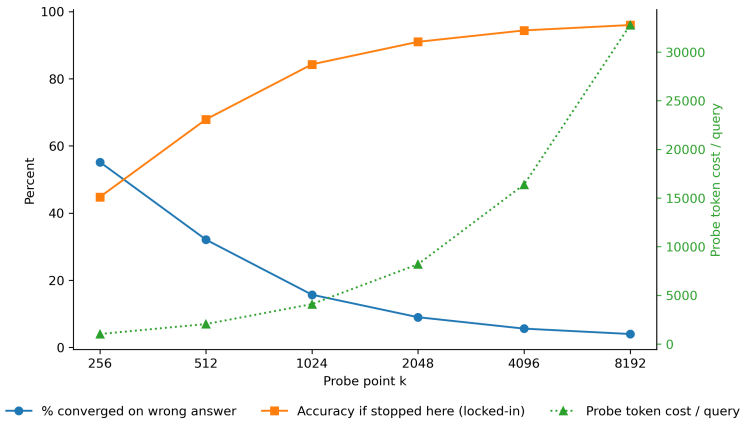


Figure 6: Agreement among four partial reasoning traces becomes more reliable as more tokens are used. Early agreement is often incorrect, while later agreement is more accurate but costs substantially more tokens.

9.0% at $k = 2048$, 5.6% at $k = 4096$, and 4.0% at $k = 8192$. Thus, agreement becomes a much better signal as more reasoning is observed. The problem is that obtaining this more reliable signal also requires spending more tokens. By the time agreement is highly reliable, much of the available compute has already been used.

Figure 7 explains why we use problem-disjoint evaluation. If different traces from the same problem appear in both training and evaluation, the prediction model can partly learn which problems are usually easy or hard instead of learning whether a particular reasoning attempt is going well. The difference is especially large on AIME, which contains only 60 distinct problems, and smaller on GSM-Symbolic, which contains 400. This supports keeping all traces from one problem in the same split, as described in Section 3.6.

Table 6 checks whether this conclusion depends on the particular prediction model or on which point in the reasoning trace is used. Overall problem-disjoint AUC can be high, showing that the predictor can distinguish generally easier from harder problems. However, within-problem performance remains close to chance across the tested prediction models and token positions. Thus, the tested

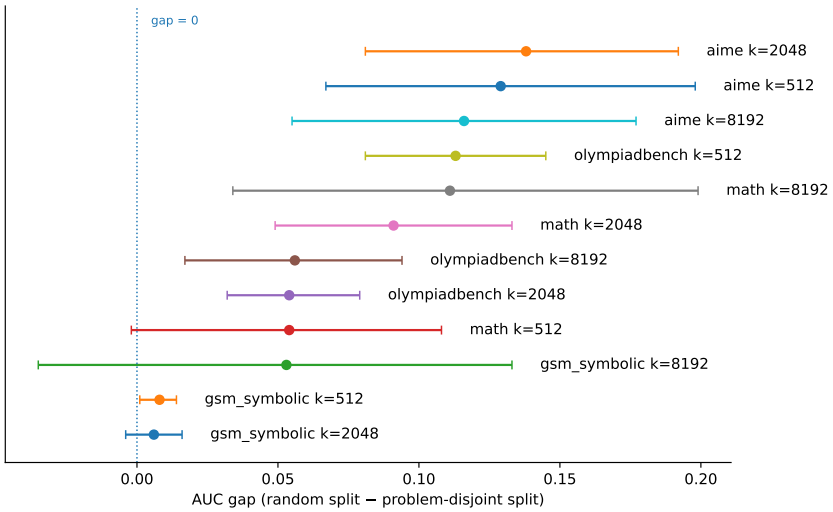


Figure 7: Keeping traces from the same problem in both training and evaluation makes reasoning progress appear easier to predict. Problem-disjoint evaluation removes this source of information.

features do not reliably distinguish a successful reasoning attempt from an unsuccessful one on the same problem.

Table 7 gives a concrete example of why the evaluation split matters. On QuALITY, an earlier evaluation that allowed information from the same group to appear across splits produced $\rho = +0.998$. Using the corrected split on the same data and prediction method gives $\rho = -0.007$. We do not use QuALITY in the main results, but this example shows how strongly the evaluation procedure can change the apparent predictability of reasoning outcomes.

Table 6: Within-problem prediction remains near chance across probe architectures and token positions under problem-disjoint evaluation. Random-split AUC can be high, but within-problem AUC is generally close to 0.5, indicating that the apparent signal does not reliably track progress within a problem.

Dataset	k	Probe	Rand.	Disjoint	Gap	Within-rand.	Within-disjoint
AIME	512	Linear	0.743	0.608	0.135	0.498	0.499
AIME	512	MLP (16)	0.757	0.626	0.131	0.557	0.552
AIME	512	MLP (32)	0.766	0.635	0.131	0.547	0.583
AIME	512	MLP (64, dropout)	0.775	0.614	0.161	0.570	0.542
AIME	2048	Linear	0.744	0.597	0.146	0.507	0.540
AIME	2048	MLP (16)	0.758	0.658	0.099	0.528	0.573
AIME	2048	MLP (32)	0.754	0.614	0.140	0.506	0.550
AIME	2048	MLP (64, dropout)	0.766	0.609	0.157	0.532	0.532
AIME	8192	Linear	0.623	0.561	0.062	0.531	0.583
AIME	8192	MLP (16)	0.653	0.601	0.052	0.544	0.584
AIME	8192	MLP (32)	0.666	0.548	0.118	0.537	0.576
AIME	8192	MLP (64, dropout)	0.639	0.505	0.133	0.508	0.556
GSM-Symbolic	512	Linear	0.912	0.884	0.027	0.491	0.483
GSM-Symbolic	512	MLP (16)	0.928	0.918	0.010	0.507	0.510
GSM-Symbolic	512	MLP (32)	0.933	0.926	0.007	0.532	0.512
GSM-Symbolic	512	MLP (64, dropout)	0.940	0.931	0.009	0.535	0.512
GSM-Symbolic	2048	Linear	0.871	0.860	0.011	0.418	0.421
GSM-Symbolic	2048	MLP (16)	0.892	0.888	0.003	0.393	0.420
GSM-Symbolic	2048	MLP (32)	0.893	0.888	0.006	0.384	0.417
GSM-Symbolic	2048	MLP (64, dropout)	0.901	0.889	0.012	0.390	0.398
GSM-Symbolic	8192	Linear	0.508	0.493	0.015	0.470	0.523
GSM-Symbolic	8192	MLP (16)	0.579	0.560	0.019	0.593	0.484
GSM-Symbolic	8192	MLP (32)	0.541	0.488	0.053	0.565	0.526
GSM-Symbolic	8192	MLP (64, dropout)	0.577	0.480	0.097	0.592	0.517
MATH	512	Linear	0.679	0.664	0.015	0.490	0.491
MATH	512	MLP (16)	0.704	0.650	0.053	0.502	0.520
MATH	512	MLP (32)	0.735	0.680	0.055	0.474	0.496
MATH	512	MLP (64, dropout)	0.768	0.662	0.106	0.466	0.491
MATH	2048	Linear	0.682	0.599	0.083	0.485	0.548
MATH	2048	MLP (16)	0.717	0.664	0.053	0.562	0.544
MATH	2048	MLP (32)	0.738	0.647	0.092	0.501	0.497
MATH	2048	MLP (64, dropout)	0.759	0.650	0.109	0.538	0.507
MATH	8192	Linear	0.578	0.507	0.070	0.555	0.529
MATH	8192	MLP (16)	0.670	0.557	0.113	0.553	0.500
MATH	8192	MLP (32)	0.654	0.539	0.115	0.511	0.489
MATH	8192	MLP (64, dropout)	0.675	0.558	0.117	0.519	0.498
OlympiadBench	512	Linear	0.735	0.630	0.105	0.425	0.463
OlympiadBench	512	MLP (16)	0.767	0.655	0.113	0.434	0.433
OlympiadBench	512	MLP (32)	0.769	0.656	0.113	0.441	0.448
OlympiadBench	512	MLP (64, dropout)	0.772	0.671	0.101	0.440	0.430
OlympiadBench	2048	Linear	0.788	0.729	0.059	0.506	0.474
OlympiadBench	2048	MLP (16)	0.815	0.752	0.063	0.442	0.465
OlympiadBench	2048	MLP (32)	0.814	0.760	0.054	0.473	0.484
OlympiadBench	2048	MLP (64, dropout)	0.824	0.765	0.059	0.460	0.469
OlympiadBench	8192	Linear	0.701	0.658	0.043	0.557	0.577
OlympiadBench	8192	MLP (16)	0.717	0.618	0.099	0.571	0.582
OlympiadBench	8192	MLP (32)	0.706	0.649	0.057	0.583	0.578
OlympiadBench	8192	MLP (64, dropout)	0.702	0.661	0.041	0.546	0.585

Table 7: QuALITY leakage correction, worked example: apparent within-group correlation under a leave-one-group-out protocol before and after correcting a label-leakage bug in the cross-validation folds.

Protocol	ρ
Flawed (leave-one-group-out, group label leaks into CV folds)	ρ = +0.998
Corrected (leave-one-group-out, group-disjoint folds)	ρ = -0.007

A.3 ALLOCATION RESULTS, EXPANDED

Table 8: Oracle and uniform-allocation reference curves on the pooled math benchmarks ($n = 1{,}035$) across token budgets.

Dataset	Budget	Allocator	m	n	Accuracy
pooled	512	Oracle	4	1035	0.6863
pooled	1024	Oracle	4	1035	0.8032
pooled	2048	Oracle	4	1035	0.8741
pooled	4096	Oracle	4	1035	0.9031
pooled	8192	Oracle	4	1035	0.9036
pooled	16384	Oracle	4	1035	0.9036
pooled	512	Uniform	4	1035	0.4569
pooled	1024	Uniform	4	1035	0.6374
pooled	2048	Uniform	4	1035	0.7506
pooled	4096	Uniform	4	1035	0.8168
pooled	8192	Uniform	4	1035	0.8590
pooled	16384	Uniform	4	1035	0.8855

Table 8 reports the oracle and uniform-allocation reference curves across token budgets. The oracle shows the possible gain from unequal allocation if we knew in advance how each problem would respond to additional compute, while uniform allocation provides the baseline. The gap is large at small budgets but becomes much smaller as the available budget increases.

Table 9 asks whether information available before reasoning can predict how difficult a problem will be. Using either the query text representation or Qwen3-8B's representation does not improve prediction over the baseline: the reported R^2 values are -0.066 and -0.060, respectively. Thus, these representations do not provide a useful early signal for deciding which problems should receive more compute in this experiment.

Figure 8 tests another way to reduce compute: choosing between thinking and non-thinking generation for each query. The signal we test has almost no relationship with which choice is better ($\rho = 0.046$, $p = 0.356$, $n = 400$). More importantly, even if we knew the better choice for every query in advance, accuracy would improve over always thinking by only 1.5 percentage points. There is therefore little room for this decision to improve accuracy in this setting.

Table 10 tests whether taking several non-thinking answers and selecting the majority answer can replace one thinking run. It does not. Even with 32 non-thinking samples, accuracy does not exceed thinking-mode accuracy at the tested token budgets. Thus, using many short answers is not a better use of the same compute in this experiment.

Table 11 provides an additional check using math problems where we do not have known shared intermediate sub-questions. Across three ways of grouping these problems and six token budgets, using information from other queries does not improve over the corresponding independent allocation method. This is consistent with the broader result: when there is no known shared reasoning to use directly, cross-query information does not provide a reliable allocation advantage.

A.4 REUSE AND AUTOMATIC DISCOVERY, EXPANDED

Across the 236 shared sub-questions used in our MuSiQue groups, Qwen3-8B answers 196 correctly when given the benchmark-provided sub-question, for an accuracy of 83.1%. This helps explain why model-generated shared answers can still be useful in the main reuse experiment.

Figure 9 checks whether the reuse benefit changes when a query receives more shared facts. The raw gain decreases from $+0.0995$ for problems receiving one fact to $+0.0444$ for problems receiving three facts. However, these groups also differ in their baseline accuracy and therefore in how much improvement is possible.

Table 9: Leave-one-group-out CV R^2 for difficulty prediction from question text alone (n=400 MuSiQue questions; individual CV fold sizes are not reported separately). Negative R^2 means worse than predicting the grand mean.

Predictor	Ridge α	LOGO-CV R^2
MiniLM-L6-v2	10,000	-0.066
Qwen3-8B hidden state	10,000	-0.060

Table 10: Non-thinking sampling and majority voting compared with thinking-mode pass@1. Even non-thinking pass@32 remains below thinking pass@1. Token counts are shown for majority voting and the thinking-mode reference.

Samples	Metric	n	Accuracy	95% CI	Tokens/query
1	Non-thinking pass@k	400	0.285	[0.2424, 0.33]	–
2	Non-thinking pass@k	400	0.3025	[0.2575, 0.3475]	–
4	Non-thinking pass@k	400	0.320	[0.2775, 0.3675]	–
8	Non-thinking pass@k	400	0.3325	[0.2874, 0.38]	–
16	Non-thinking pass@k	400	0.340	[0.295, 0.3825]	–
32	Non-thinking pass@k	400	0.350	[0.3025, 0.3925]	–
1	Non-thinking majority vote	400	0.285	[0.2425, 0.3275]	8.0
2	Non-thinking majority vote	400	0.2625	[0.22, 0.305]	16.0
4	Non-thinking majority vote	400	0.275	[0.23, 0.32]	33.0
8	Non-thinking majority vote	400	0.280	[0.235, 0.325]	72.0
16	Non-thinking majority vote	400	0.275	[0.2325, 0.3175]	164.0
32	Non-thinking majority vote	400	0.275	[0.2325, 0.32]	1258.0
1	Thinking pass@1	400	0.4988	[0.4569, 0.5378]	1339.0

Table 11: GroupCLEAR on math under random, embedding, and no-overlap grouping, all three with delta vs. Marginal-gain and 95% CI at every budget. All differences are at or near zero across every condition and budget: no grouping scheme recovers a signal on problems that lack genuine sub-question structure.

avg_k	Grouping	Allocator	n	Accuracy	Δ vs Marginal-gain (95% CI)
512	random	GroupCLEAR(a=0.5)	1035	0.4569	+0.0000 [0.0, 0.0]
1024	random	GroupCLEAR(a=0.5)	1035	0.6374	+0.0000 [0.0, 0.0]
2048	random	GroupCLEAR(a=0.5)	1035	0.7506	+0.0000 [0.0, 0.0]
4096	random	GroupCLEAR(a=0.5)	1035	0.8168	+0.0000 [0.0, 0.0]
8192	random	GroupCLEAR(a=0.5)	1035	0.8552	+0.0010 [-0.0033, 0.0052]
16384	random	GroupCLEAR(a=0.5)	1035	0.8802	-0.0014 [-0.0039, 0.0006]
512	embedding	GroupCLEAR(a=0.5)	1035	0.4569	+0.0000 [0.0, 0.0]
1024	embedding	GroupCLEAR(a=0.5)	1035	0.6374	+0.0000 [0.0, 0.0]
2048	embedding	GroupCLEAR(a=0.5)	1035	0.7506	+0.0000 [0.0, 0.0]
4096	embedding	GroupCLEAR(a=0.5)	1035	0.8168	+0.0000 [0.0, 0.0]
8192	embedding	GroupCLEAR(a=0.5)	1035	0.8542	-0.0000 [-0.0002, 0.0002]
16384	embedding	GroupCLEAR(a=0.5)	1035	0.8818	+0.0002 [-0.0002, 0.0006]
512	no_overlap	GroupCLEAR(a=0.5)	1035	0.4569	+0.0000 [0.0, 0.0]
1024	no_overlap	GroupCLEAR(a=0.5)	1035	0.6374	+0.0000 [0.0, 0.0]
2048	no_overlap	GroupCLEAR(a=0.5)	1035	0.7506	+0.0000 [0.0, 0.0]
4096	no_overlap	GroupCLEAR(a=0.5)	1035	0.8168	+0.0000 [0.0, 0.0]
8192	no_overlap	GroupCLEAR(a=0.5)	1035	0.8542	+0.0000 [0.0, 0.0]
16384	no_overlap	GroupCLEAR(a=0.5)	1035	0.8816	+0.0000 [0.0, 0.0]

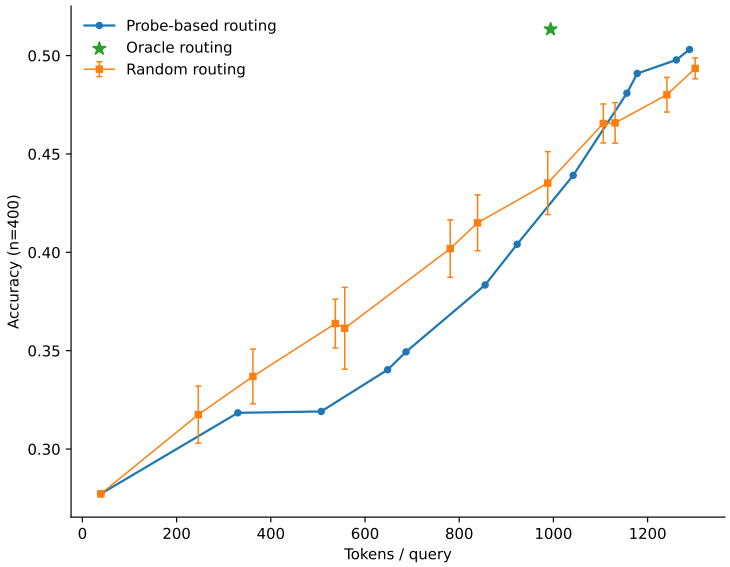


Figure 8: Accuracy and token use when choosing between thinking and non-thinking generation on MuSiQue. Even a choice that always selects the better option improves accuracy over always thinking by only 1.5 percentage points.

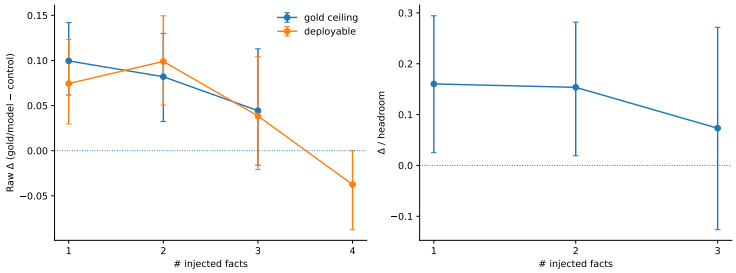


Figure 9: Reuse gains by number of shared facts among the 375 problems retaining at least one usable shared fact. The figure shows both the raw accuracy gain and the gain relative to the amount of accuracy still available to improve.

After accounting for the remaining room to improve, the values are 0.160, 0.154, and 0.073 for one, two, and three facts. The estimate for three facts is much less certain because it contains only 62 problems and its 95% confidence interval includes zero. We therefore do not find clear evidence that reuse systematically becomes less useful as more shared facts are provided.

The model-generated-answer condition also contains only 10 problems that receive four shared facts. Its raw change is -0.038, with a 95% confidence interval of $[-0.088, 0.0]$. Because this group is so small, we do not interpret the negative value as evidence that providing four facts is harmful.

Figure 10 examines the 126 queries that receive at least one incorrect model-generated shared answer. Their accuracy changes are centered near zero, so we do not detect a significant decrease in final accuracy. This should not be read as showing that incorrect shared answers can never hurt. With 126 queries, effects smaller than about 5.8 percentage points would be difficult to detect reliably. The result is therefore that we do not detect harm at this sample size.

A.4.1 Automatic-discovery details

We compare generated sub-questions only within the same 10-query MuSiQue group and never across groups. For each pair of generated sub-questions from different queries, we compute the cosine similarity between their MiniLM-L6-v2 embeddings. We treat the pair as a match when its similarity is at least τ. If several matches connect to one another, we group them together; for example, if A matches B and B matches C, then A, B,

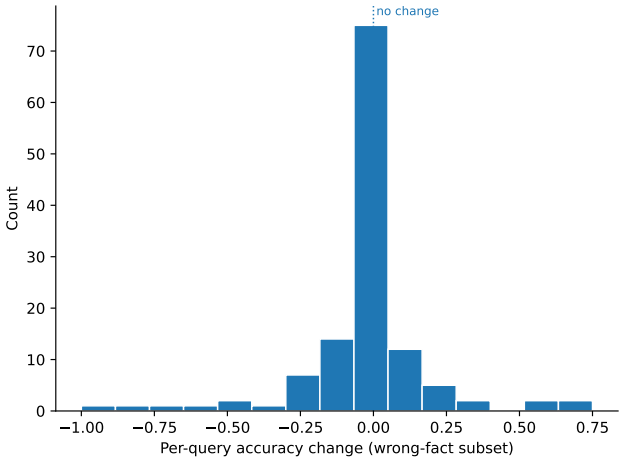


Figure 10: Change in final-query accuracy for queries that receive at least one incorrect model-generated shared answer. The average change is close to zero in this sample.

Table 12: Precision–recall tradeoff across matching thresholds. End-to-end accuracy was evaluated only at $\tau = 0.70$ and $\tau = 0.95$; both tested operating points reduced accuracy. All rows use $n = 400$.

τ	Prec.	Rec.	F1	FP/problem	E2E?	Accuracy Δ (95% CI)	
0.50	0.482	0.985	0.647	2.97	N	not run	
0.55	0.553	0.963	0.703	2.66	N	not run	
0.60	0.624	0.939	0.749	2.41	N	not run	
0.65	0.698	0.885	0.781	2.14	N	not run	
0.70	0.765	0.805	0.784	1.88	Y	-0.145	[-0.1766, -0.1144]
0.75	0.817	0.680	0.742	1.26	N	not run	
0.80	0.850	0.547	0.666	0.81	N	not run	
0.85	0.874	0.437	0.583	0.45	N	not run	
0.90	0.893	0.310	0.460	0.36	N	not run	
0.95	0.916	0.185	0.308	0.23	Y	-0.127	[-0.1562, -0.0969]

and C are placed in the same group. We keep only groups containing sub-questions from at least two different queries.

To evaluate these matches, we use MuSiQue's benchmark-provided sub-question identities. Two generated sub-questions count as a correct match when they correspond to the same benchmark sub-question used by multiple queries in the group. Precision, recall, and F_1 in Table 12 are computed over these pairwise matches.

For the analysis in Table 13, we separately ask whether a generated sub-question preserves what the corresponding benchmark sub-question is asking. This is different from asking whether two generated sub-questions were correctly matched: a match can identify the intended intermediate reasoning step while the generated wording still changes its meaning. We therefore compare the generated sub-question with its corresponding benchmark sub-question and directly test whether Qwen3-8B can answer each version correctly.

Table 12 reports matching quality across all tested similarity thresholds. As the threshold increases from $\tau = 0.50$ to $\tau = 0.95$, matching precision increases from 0.482 to 0.916, while fewer true shared steps are found and the average number of incorrect matches per problem decreases from 2.97 to 0.23. Thus, stricter matching produces fewer but more reliable matches.

We run the full downstream experiment only at $\tau = 0.70$, which gives the best F_1 score, and at $\tau = 0.95$, which gives the highest tested precision. Both reduce final accuracy: by 14.5 points at $\tau = 0.70$ and 12.7 points at $\tau = 0.95$. Because the other thresholds were not tested end-to-end, this table alone does not show that every possible threshold would reduce accuracy. The next experiment addresses this more directly by looking only at matches that are known to be correct.

Table 13 compares the original benchmark sub-question with the model-generated sub-question in 472 cases where both refer to the same intermediate reasoning step. Qwen3-8B answers the benchmark sub-question correctly 83.9% of the time but answers the model-generated version correctly only 12.5% of the time. In 341

Table 13: Automatic-discovery matching and generated-sub-question quality. The upper block shows matching statistics across similarity thresholds. The lower block compares benchmark and generated sub-questions for 472 correctly matched shared facts at $\tau = 0.95$.

τ	Matched groups	Problems	Facts/problem	Incorrect facts/problem
0.50	95	396	3.30	2.97
0.55	118	395	3.14	2.66
0.60	135	394	2.97	2.41
0.65	146	394	2.79	2.14
0.70	154	385	2.68	1.88
0.75	169	373	2.54	1.26
0.80	180	361	2.35	0.81
0.85	186	353	2.19	0.45
0.90	190	331	2.04	0.36
0.95	187	290	1.80	0.23

Benchmark vs. generated sub-questions at $\tau = 0.95$	
Benchmark sub-question answered correctly	396/472 (83.9%)
Generated sub-question answered correctly	59/472 (12.5%)
Benchmark correct / generated wrong	341
Generated correct / benchmark wrong	4

cases, the benchmark version is answered correctly while the generated version is answered incorrectly; only 4 cases go in the opposite direction.

The table also shows that increasing the matching threshold reduces how many problems receive any matched shared fact at all, from 396 problems at $\tau = 0.50$ to 290 at $\tau = 0.95$. A stricter matching threshold therefore improves the reliability of the remaining matches but also removes many possible reuse opportunities. More importantly, the large 83.9% versus 12.5% gap remains even when the match itself is correct. This supports the main-text conclusion that matching alone is not the main problem: the generated sub-question often fails to preserve what the original reasoning step is asking.

Table 14 gives concrete examples of model-generated sub-questions changing the meaning of the benchmark sub-question. In some cases, the generated version drops information needed to identify the correct entity. In others, it combines multiple steps into a different question, asks for a different property, or uses a pronoun whose referent is no longer clear. The table also includes a case where both versions fail, showing that changing the meaning of the sub-question does not explain every error. These examples illustrate the same problem measured quantitatively in Table 13.

Table 15 records the exact prompts used to provide shared answers and the construction of the unrelated controls. The shared and control blocks have the same mean length of 32.1 tokens, and 373 of 375 pairs differ in length by less than 10%. This supports the comparison in the main text: the accuracy gain comes from providing the relevant shared answer rather than simply adding more text.

The table also records the code used to remove a shared fact when it contains the final answer. The known-sharing and automatic-discovery experiments use slightly different implementations of this check: the automatic-discovery version does not include the same whitespace and empty-string checks. These differences appear too small to explain the large difference between the two experiments, but we did not rerun the automatic-discovery experiment with exactly the same code. We therefore treat this as an unchecked implementation difference rather than assuming it has no effect.

A.5 Full-solution transfer

Table 16 tests whether providing the complete solution to a related math problem helps solve the target problem. At $k = 512$ and $k = 1024$, the changes are small and their confidence intervals include zero. At $k = 2048$ and $k = 4096$, providing the related solution slightly reduces accuracy by 2.0 and 1.6 percentage points, respectively. This supports the main-text result that a complete related solution is not a substitute for reusing the specific intermediate reasoning that two queries share.

A.6 Rejected Qwen2.5-Math setup

Table 17 explains why we use Qwen3-8B rather than Qwen2.5-Math-7B-Instruct. In a preliminary AIME test, Qwen2.5-Math-7B-Instruct sometimes produced unusable outputs when sampled. Under its default tested sampling setting, 11 of 40 outputs were unusable. Changing the sampling settings reduced this problem but did

Table 14: Examples of generated sub-questions changing the meaning of the benchmark sub-question. Most failures arise because the generated version asks a different question, even when the intended shared reasoning step is matched correctly.

Case	Detail
Aggregate (472 correct matches)	The benchmark sub-question is answered correctly in 83.9% of cases (396/472), compared with 12.5% for the generated version (59/472). In 341 cases the benchmark version is correct and the generated version is wrong; only 4 cases go in the opposite direction. Known-sharing reuse improves accuracy by 7.0 points relative to the no-reuse baseline, while automatic discovery at $\tau = 0.95$ reduces it by 12.7 points.
#1 Markus Zusak	Benchmark: citizenship → Australia (correct). Generated: "Which country is near the country mentioned previously?" → Belgium (wrong). The generated question changes the task entirely.
#3 Fort Hill / Clemson	Benchmark asks which of Clemson or South Carolina has more national championships → South Carolina. Generated asks how many championships the Fort Hill university has → zero. The generated question collapses two steps into a different question.
#5/#6 *Oh Yeah*	Benchmark: performer → Charles Mingus. Generated asks which state the performer is from → Missouri. The generated question asks for a downstream attribute rather than the requested entity.
#9 Columbia County	Benchmark: administrative region → Oregon. Generated: "Which state contains that county?" → Oklahoma. The phrase "that county" has no usable antecedent.
#10 East Timor president	Benchmark: president of East Timor → Francisco Guterres. Generated refers to "the country identified in question 2" → Pedro I of Brazil. The generated question points to the wrong entity.
#2/#4 Atlanta	Both versions are answered incorrectly because the benchmark sub-question itself is ambiguous. Both return Sherman instead of the benchmark answer John Bell Hood.
Dangling reference ($\tau = 0.70$)	Benchmark: Titian place of death → El Escorial. Generated asks how many times the plague occurred in "that city" → three. The generated question introduces an unsupported additional step.
Incorrect match ($\tau = 0.70$)	The benchmark contains no shared sub-question, but automatic discovery injects an unrelated greyhound-bus fact through an incorrect match.

Table 15: Reuse prompts, controls, and final-answer filtering. Both the known-sharing and automatic-discovery experiments remove shared facts whose answer matches the problem's final answer in either substring direction.

Element	Content
Shared-answer prompt	"You may use the following verified facts:" followed by `[resolved sub-question] → [answer].`
Matched control	Facts from another group's shared sub-questions, using the same format and matched to within 10% of the shared-answer block's token count (Qwen3-8B tokenizer).
Token matching	Both conditions average 32.1 tokens/block; 373/375 blocks fall within the 10% tolerance.
Known-sharing filter	Lowercase and strip the final and shared-fact answers; drop the fact if either is a substring of the other and the shared-fact answer is nonempty. This removes 319 facts across 303 problems.
Automatic-discovery filter	Lowercase the final and shared-fact answers; drop the fact if either is a substring of the other. This removes 30 facts across 29 problems.
Implementation difference	The automatic-discovery filter lacks the known-sharing filter's explicit `strip()` and empty-string check. We did not rerun the experiment with identical filtering.

not remove it consistently. Qwen's recommended settings reduced the count to 1 of 40, with 3 additional traces reaching the maximum length.

Table 16: Pilot results for providing the full solution of a related math problem when solving the target problem ($n = 99$).

k	Baseline	With related solution	Δ	95% CI
512	0.537	0.561	+0.024	$[-0.016, +0.066]$
1024	0.806	0.822	+0.016	$[-0.015, +0.044]$
2048	0.905	0.885	-0.020	$[-0.043, -0.002]$
4096	0.944	0.928	-0.016	$[-0.031, -0.005]$

Table 17: Qwen2.5-Math-7B-Instruct sampling-collapse diagnostic on AIME smoke-test prompts. The model was rejected before the full experiments because sampled decoding frequently produced unusable outputs.

Setting	Garbage traces / total
vLLM greedy	0/10
vLLM T=0.01	1/40
vLLM T=0.7 top_p=0.95 (task default), per-request seeds	11/40
same, no seeds (FlashInfer sampler)	6/40
same, use_fp64_gumbel=True	7/40
HF transformers, T=0.7 top_p=0.95	4/40
vLLM fp64, T=0.5 top_p=0.95	6/40
vLLM fp64, T=0.6 top_p=0.95 top_k=20 rep=1.05	9/40
vLLM fp64, T=0.7 top_p=0.8 top_k=20 (Qwen recommended)	1/40 (3 capped)

Because this behavior would make the stored trace corpus unreliable, we rejected Qwen2.5-Math-7B-Instruct before running the full experiments. This check only explains our model choice. It is not evidence about the behavior of Qwen3-8B, for which we did not observe the same problem in the trace corpus.

A.7 Complete list of tested variants

Table 18 lists every reuse, automatic-discovery, and MuSiQue allocation variant reported in this paper. The two successful reuse experiments—using benchmark-provided shared answers and model-generated shared answers when the shared sub-question is known—were specified before their results were observed. The two automatic-discovery experiments were added afterward to test the natural next question: whether the shared reasoning that made reuse work could also be found automatically.

The table also includes all nine MuSiQue allocation variants. None improves over the corresponding method that decides each query's budget independently. For several of these rows, uncertainty estimates were not exported. Values shown as exactly zero should therefore be read as matching the baseline to the reported numerical precision, not as evidence that the true difference is exactly zero.

A.8 Additional limitations

The main text gives the limitations most directly tied to the central claims. We record several additional details here.

Single model and seed. All stored traces come from one model, Qwen3-8B, under one fixed generation setup and one fixed seed (Table 4). We therefore do not know how much the numerical results would change with a different model, model size, or complete regeneration of the corpus. The confidence intervals reported in the paper measure variation across the problems and traces we generated, not variation across independently regenerated corpora.

Two tested domains. We study shared reasoning using math problems and MuSiQue. These datasets are useful because shared structure can be identified from templates or benchmark-provided sub-questions. We do not test other settings in which queries might share reasoning, such as code generation, scientific reasoning, or queries that use the same source documents. The current results should therefore not be assumed to transfer directly to those settings.

Small effects may be missed. Some experiments do not find a significant effect, but their sample sizes do not allow us to rule out arbitrarily small effects. For example, the incorrect-shared-answer experiment in

Table 18: Variant registry. Bold rows pass; italicized variants were added after the initial reuse results. Reuse/discovery variants report 95% CIs; MuSiQue allocation/routing variants report point estimates only because CIs were not exported. For reuse/discovery, Δ is relative to the no-reuse baseline rather than the comparison control listed in the table. For allocation/routing, Δ is relative to the marginal-gain baseline.

Variant	Shares	Comparison control	Pre-reg.	n	Δ vs. baseline	Verdict
Reuse / automatic discovery						
Benchmark-provided answer	known shared sub-Qs	Token-matched irrelevant facts	Y	400	+0.076 [0.047, 0.105]	**PASS**
Model-generated answer	known shared sub-Qs	Token-matched irrelevant answers	Y	400	+0.070 [0.040, 0.100]	**PASS**
Automatic discovery ($\tau = 0.70$)	matched sub-Qs	Token-matched control	N	400	-0.145 [-0.177, -0.114]	FAIL
Automatic discovery ($\tau = 0.95$)	matched sub-Qs	Token-matched control	N	400	-0.127 [-0.156, -0.097]	FAIL
Allocation / routing (MuSiQue)						
GroupCLEAR	–	Marginal-gain baseline	Y	400	+0.000	FAIL
GroupBudget	–	Marginal-gain baseline	Y	400	+0.000	FAIL
Sequential	–	Marginal-gain baseline	Y	400	+0.000	FAIL
ProbeOne	–	Marginal-gain baseline	Y	400	-0.003	FAIL
ProbeOneSeq	–	Marginal-gain baseline	Y	400	-0.003	FAIL
MiniLM	–	Marginal-gain baseline	Y	400	-0.004	FAIL
HiddenState	–	Marginal-gain baseline	Y	400	-0.006	FAIL
Cascade	–	Marginal-gain baseline	Y	400	-0.004	FAIL
Stretch	–	Marginal-gain baseline	Y	400	-0.004	FAIL

Figure 10 would have difficulty reliably detecting an accuracy change smaller than about 5.8 percentage points. Similarly, the experiment in Figure 8 can rule out moderate relationships but may miss a small one. These results should therefore be read as "no reliable effect detected" rather than as proof that the true effect is exactly zero.

One method for matching generated sub-questions. Our automatic-discovery experiment uses MiniLM-L6-v2 embeddings and cosine similarity to match generated sub-questions. Other matching methods could improve which sub-questions are paired. However, Table 13 shows that poor matching is not the only problem: even among 472 correct matches, the generated sub-question is answered correctly only 12.5% of the time, compared with 83.9% for the benchmark version. A better matching method would therefore still need a way to ensure that generated sub-questions preserve what the original reasoning step asks.

Small difference in the final-answer filtering code. As noted in Appendix A.4, the known-sharing and automatic-discovery experiments use slightly different code for removing shared facts that contain the final answer (Table 15). We do not expect this small implementation difference to explain the large difference in accuracy between the two experiments, but we did not rerun the experiment with identical filtering code. We therefore leave this as an unverified implementation detail.