RECAST: LEARNING TO COMPUTE THE RIGHT CONTEXT THROUGH ADAPTIVE EVIDENCE ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly applied to tasks grounded in long, heterogeneous information sources. Conventional Retrieval-Augmented Generation (RAG) relies on fixed similarity-based retrieval, while agentic variants adapt queries and tool use but remain largely retrieval-centric. However, in many tasks, the evidence required for a solution is not explicitly present in any single source item. Instead, it must be derived through filtering, aggregation, or computation across multiple source items. In this work, we introduce RECAST (**R**outing **E**vidence through **C**omputation, **A**ccess, and **S**ynthesized **T**ools), a learned framework that formulates evidence construction as a sequential decision process over heterogeneous retrieval and computation operations, allowing evidence to be actively derived rather than merely retrieved. A lightweight ROUTERLM iteratively selects and formulates primitive operations or specifies customized operations for a frozen COMPILERLM to translate into executable code. Once it judges the evidence sufficient, ROUTERLM passes the accepted evidence to a frozen ANSWERLM to produce the final solution. We train ROUTERLM with supervised fine-tuning (SFT) followed by group relative policy optimization (GRPO). Across six heterogeneous benchmark families, RECAST achieves a mean success rate of 75.6%, outperforming the strongest large-model baseline by 15.9%. Moreover, training enables the Qwen3.5-9B ROUTERLM to outperform a training-free Gemini 3.5 Flash ROUTERLM by 5.0%. On three held-out benchmarks, RECAST improves over the strongest baseline by 15.0% on average, demonstrating strong zero-shot generalization across tasks and heterogeneous source representations.

1 INTRODUCTION

Large language models (LLMs) are increasingly applied to diverse tasks, such as question answering, data analysis, and personalized assistance, many of which require reasoning over long contexts drawn from heterogeneous external sources such as document collections, databases, tables, semi-structured records, and user histories (Nakano et al., 2021; Jiang et al., 2023a; Zhang et al., 2023; Li et al., 2023; Zhong et al., 2024; Packer et al., 2023). Although modern LLMs can process increasingly long inputs, supplying the complete source incurs substantial computational and token costs while still offering no guarantee that the model will identify and correctly use the information required for a particular task (Jiang et al., 2024; Liu et al., 2024). Retrieval-augmented generation (RAG) offers a practical solution by retrieving a small amount of relevant content and conditioning generation on the retrieved evidence (Guu et al., 2020; Lewis et al., 2020; Izacard & Grave, 2021). Conventional RAG typically performs a single retrieval step before generation, whereas iterative and agentic variants allow LLMs to reformulate queries, invoke retrieval operations over multiple rounds, and assess whether additional evidence is needed (Trivedi et al., 2023; Hui et al., 2026).

These retrieval-based formulations generally assume that useful evidence can be located directly within the source. However, LLM applications increasingly rely on heterogeneous sources, from which the required evidence may need to be derived rather than simply retrieved. For example, suppose the source provides monthly revenue and operating income, while the task asks which three-month period had the largest increase in operating margin. Answering the question requires computing the margins and comparing their changes across periods. More broadly, the required operations vary across tasks, from lexical or semantic retrieval to relational filtering and aggregation, and some tasks require customized computation that must be synthesized for the task at hand.

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.

Context construction must therefore determine not only *where* relevant information appears, but also *how* it should be transformed into useful evidence. Although iterative and agentic methods make retrieval more adaptive, they primarily change how the system searches for source content and offer limited support for constructing such derived evidence. We therefore consider a broader context-construction problem in which retrieval and computation serve as complementary evidence operations, adaptively selected and combined for each task instance.

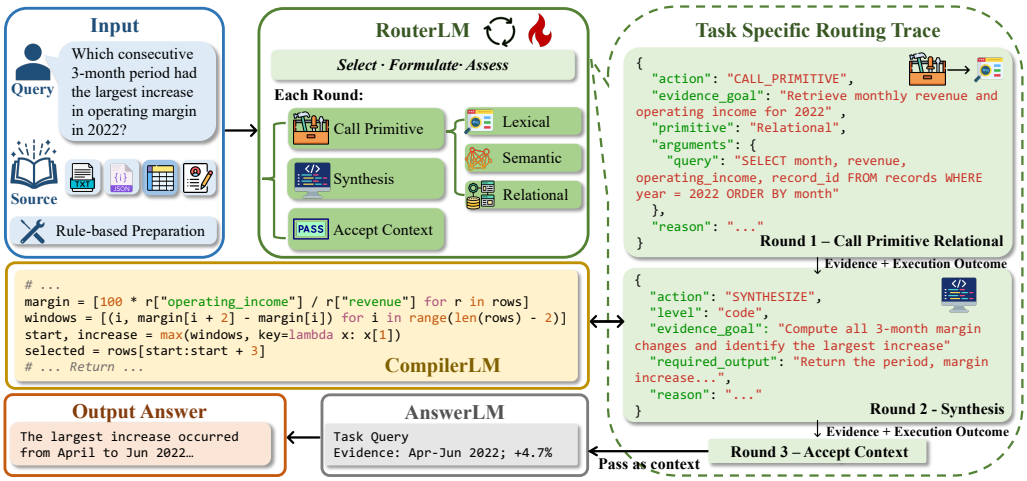

Figure 1: **Overview of RECAST.** At each round, RouterLM uses the task, compact source profile, and evidence and feedback from earlier rounds to either select and formulate a primitive or synthesized operation or pass the evidence to AnswerLM when it considers the evidence sufficient.

In this work, we introduce RECAST (**R**outing **E**vidence through **C**omputation, **A**ccess, and **S**ynthesized **T**ools), illustrated in Figure 1. RECAST formulates evidence construction as a learned sequential decision process over heterogeneous retrieval and computation operations. At each round, a lightweight RouterLM determines what evidence is needed and formulates a task-specific operation for obtaining it. It may invoke a parameterized lexical, semantic, or relational primitive, or request synthesis of a customized operation that a frozen CompilerLM translates into executable code. The resulting evidence and execution outcome inform subsequent decisions, allowing RouterLM to refine the constructed evidence over multiple rounds. Once the evidence is sufficient, RouterLM accepts it and passes it as context to a frozen AnswerLM for final solution generation. We train the RouterLM using supervised fine-tuning (SFT) followed by group relative policy optimization (GRPO).

We evaluate RECAST on DataBench, FinQA, HiTab, HotpotQA, LaMP, and MultiHiertt (Grijalba et al., 2024; Chen et al., 2021; Cheng et al., 2022a; Yang et al., 2018; Salemi et al., 2024; Zhao et al., 2022). These six benchmarks span heterogeneous source representations, including dataframes, financial reports, hierarchical tables, Wikipedia passage collections, and user profiles. Across them, RECAST achieves an average success rate of 75.6%, outperforming the strongest large-model baseline at 59.7%. The trained Qwen3.5-9B RouterLM outperforms a large training-free Gemini 3.5 Flash router, which achieves 70.6%. On three held-out benchmarks, 2WikiMultiHopQA, TAT-QA, and WikiTableQuestions (Ho et al., 2020; Zhu et al., 2021; Pasupat & Liang, 2015), RECAST obtains 79.3%, compared with 64.3% for the strongest baseline, demonstrating strong zero-shot generalization across tasks and heterogeneous source representations.

In summary, our key contributions are:

- We broaden evidence construction beyond retrieval by treating retrieval and computation as complementary means of obtaining task-relevant evidence from heterogeneous sources.
- We propose RECAST, a multi-round framework in which a lightweight RouterLM formulates primitive or synthesized evidence operations, assesses their outcomes, and determines when evidence is sufficient. We further improve its evidence-construction policy through SFT and GRPO.
- We evaluate RECAST across 6 in-domain and 3 held-out benchmark families. RECAST achieves 75.6% in-domain and 79.3% on held-out benchmarks, outperforming the strongest baselines by

15.9% and 15.0%, respectively. It also surpasses a large training-free RouterLM and transfers effectively across alternative CompilerLMs.

2 Related Works

Retrieval-Augmented and Agentic Context Construction. Retrieval-augmented generation (RAG) grounds language-model outputs in external information by retrieving relevant source content and conditioning generation on it (Guu et al., 2020; Karpukhin et al., 2020; Lewis et al., 2020; Izacard & Grave, 2021). Subsequent work develops iterative and hierarchical retrieval strategies for multi-hop and long-context reasoning, such as evidence condensation across retrieval hops, reasoning-guided query formulation, and retrieval over hierarchical document representations (Khattab et al., 2021; Trivedi et al., 2023; Sarthi et al., 2024). More adaptive systems decide when and how to retrieve, assess or correct retrieved results, and use intermediate reasoning to guide subsequent retrieval actions (Jiang et al., 2023b; Asai et al., 2024; Jeong et al., 2024; Yan et al., 2024; Yao et al., 2022; Hui et al., 2026). While these methods move beyond fixed one-pass retrieval, they still construct evidence primarily by selecting or refining information already present in the source. RECAST instead extends adaptive context construction to relational computation and synthesized transformations that derive new evidence from source content.

Tool-Augmented and Programmatic Reasoning. A complementary line of work augments language models with structured operations, tools, and executable programs. Methods for structured sources use specialized table models, symbolic programs, or iterative operations over tables and records (Herzig et al., 2020; Liu et al., 2021; Jiang et al., 2023a; Cheng et al., 2022b; Wang et al., 2024). General tool-use frameworks allow language models to select and invoke predefined tools (Schick et al., 2023; Paranjape et al., 2023; Qin et al., 2024), while program-aided and tool-synthesis approaches generate executable programs or new tools for the task (Gao et al., 2023; Chen et al., 2022; Qian et al., 2023; Cai et al., 2024; Gemmell & Dalton, 2023). These methods broaden the computational capabilities of language models, but generally either generate programs that directly produce task answers or rely on operations designed for a particular source type. RECAST instead learns to construct task-relevant evidence across heterogeneous sources by iteratively selecting, formulating, and evaluating primitive or synthesized operations before final solution generation.

3 RECAST

3.1 Problem Formulation and Challenges

Given a natural-language task q and an associated external source collection $\mathcal{X} = \{x_i\}_{i=1}^{N}$, our goal is to construct query-specific evidence that enables an LLM to produce the solution y. We write this process as

$$E = G_\theta(q, \mathcal{X}), \qquad y = F_{\text{ans}}(q, E), \tag{1}$$

where G_θ denotes the evidence-construction process parameterized by the learned routing policy and F_{ans} denotes the frozen answer model. A source item x_i may take diverse forms, such as a text passage, structured table, JSON record, hierarchical table, or user-profile entry. Rather than applying F_{ans} directly to the complete source, RECAST constructs compact evidence E by adaptively retrieving, transforming, or computing information from $\mathcal{X}$. This evidence may include both directly retrieved source content and information derived through filtering, aggregation, arithmetic, linking, or other transformations across multiple source items. This setting presents two main challenges:

• **Operation Diversity and Formulation.** Different tasks require fundamentally different evidence operations. Finding exact names or identifiers may require lexical matching, locating paraphrased concepts may require semantic retrieval, and combining or calculating over multiple records may require relational or algorithmic computation. When the required transformation falls outside the capabilities of reusable retrieval and computation operations, a task-specific program may be needed. Selecting an operation alone is insufficient. The system must also formulate the task-specific request by specifying the query or computation, the records to use, and the evidence to return.

• **Sequential Decisions and Evidence Assessment.** Evidence construction may require multiple rounds. An initial operation may identify the subset of the source needed by a later operation, return

insufficient evidence, or fail at runtime. The system must assess each outcome, decide whether another operation is needed, and stop only when the retained evidence supports final inference.

3.2 RECAST OVERVIEW

To address these challenges, we propose RECAST (Figure 1), a multi-round implementation of the evidence-construction mapping G_θ introduced above. ROUTERLM, parameterized by θ, implements the policy that selects, formulates, and assesses evidence operations. A frozen COMPILERLM, denoted F_{comp}, translates customized operation specifications into executable code, while the frozen ANSWERLM implements F_{ans} and maps the constructed evidence to the final solution. Only ROUTERLM is trained. The framework proceeds in four steps.

Step 1: Source preparation. Because source information may be represented in different formats, a deterministic, rule-based preprocessor converts the source context $\mathcal{X}$ into a uniformly formatted list of records and constructs a compact source profile m. The profile summarizes the source scale, structure, formats, and observed fields, providing ROUTERLM with a high-level view of the source and allowing it to operate across heterogeneous sources without receiving the complete source in its prompt. This rule-based preparation uses no language model and is independent of the task. As an initialization step, both the evidence set and interaction history are empty: $E_0 = \emptyset$ and $H_0 = \emptyset$.

Step 2: Evidence-operation formulation. At round t, ROUTERLM observes the task q, source profile m, retained evidence E_t, and interaction history H_t. It selects an action according to

$$a_t \sim \pi_\theta(\cdot \mid q, m, E_t, H_t). \tag{2}$$

The action either invokes a primitive operation from $\{\text{LEXICAL}, \text{SEMANTIC}, \text{RELATIONAL}\}$, requests a synthesized evidence operation, or accepts the current evidence. Section 3.3 describes these operations in detail. This step does not produce the final task solution. Instead, ROUTERLM must assess what evidence is still needed, select an appropriate operation, and formulate its task-specific request, including the arguments or output requirements that specify how the evidence should be obtained and what should be returned.

Step 3: Operation execution and feedback. For a primitive action, the selected operation is executed using task-specific arguments formulated by ROUTERLM. A synthesis request is translated by COMPILERLM into a Python program and then executed over the source. Either path returns an operation outcome o_t describing execution status, result counts, and any errors. A successful operation may return new evidence ΔE_t. The evidence and interaction states are updated as

$$E_{t+1} = E_t \cup \Delta E_t, \qquad H_{t+1} = H_t \parallel [(a_t, o_t)], \tag{3}$$

where $\parallel$ denotes appending the latest action and outcome to the interaction history.

Step 4: Evidence assessment and solution generation. Steps 2–3 are repeated until ROUTERLM considers the accumulated evidence sufficient. ROUTERLM then accepts E_t as the final evidence and terminates the evidence-construction loop. Finally, ANSWERLM is invoked once with the original task q and the accepted evidence E_t to produce the solution y.

Figure 2 summarizes the structured action space of ROUTERLM. At each round, ROUTERLM selects one of three actions and formulates the task-specific information needed to carry it out. `CALL_PRIMITIVE` and `SYNTHESIZE` construct additional evidence, whereas `ACCEPT_CONTEXT` terminates evidence construction and proceeds to final solution generation.

3.3 EVIDENCE OPERATIONS

3.3.1 CALL_PRIMITIVE

Operation selection and formulation. ROUTERLM selects an operation from $\{\text{LEXICAL}, \text{SEMANTIC}, \text{RELATIONAL}\}$. LEXICAL ranks source records using BM25 term matching (Robertson & Zaragoza, 2009), while SEMANTIC ranks them by cosine similarity using BGE-M3 embeddings (Chen et al., 2024). RELATIONAL executes a read-only Structured Query Language (SQL) query over the normalized records, supporting filtering, joins, grouping, aggregation, sorting, and arithmetic. ROUTERLM must also formulate the selected operation for

the current evidence goal. Lexical and semantic calls specify a search query and the number of records to return, whereas relational calls specify a SQL query. A call may additionally be restricted to source records identified in earlier rounds.

Execution and output. The formulated operation is executed directly over the source records and returns retrieved content or a computed result together with its source identifiers. The executor also reports whether the operation succeeded, how many records matched and were returned, and any execution errors. Successful execution indicates that the requested operation ran correctly, but does not imply that the resulting evidence is sufficient for the task.

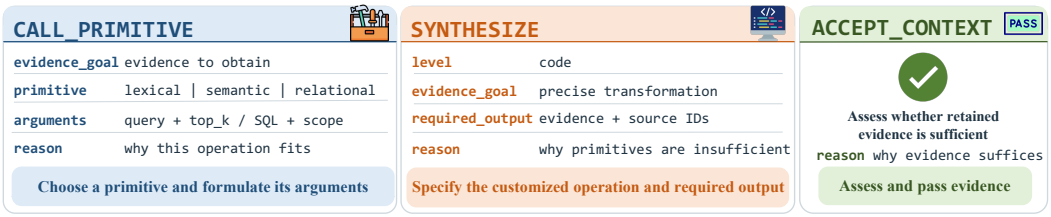

Figure 2: **RouterLM action space and output structure.** Together, the outputs represent action selection, task-specific formulation, and justification.

3.3.2 Synthesize

Operation specification. When the required transformation cannot be expressed through the available primitives, RouterLM requests a synthesized operation. RouterLM specifies the transformation to perform, the evidence and source identifiers to return, and why the primitive operations are insufficient. Because CompilerLM does not receive the original task, the specification must preserve the task-specific constraints required to implement the operation correctly.

Compilation and execution. CompilerLM receives the synthesis specification, source profile, source samples, and prior operation outcomes, and translates the specification into a Python program in one call. The program can process the complete source, use previously obtained evidence, and invoke primitive operations when useful. It returns either the requested evidence or an execution error to RouterLM. In Figure 1 example, RouterLM specifies the computation for identifying the three-month period with the largest margin increase, while CompilerLM translates that specification into code.

3.3.3 ACCEPT_CONTEXT

Evidence assessment. After each evidence-producing action, RouterLM receives the returned evidence and structured feedback generated by the operation executor. The feedback reports the execution status, result counts, and any errors. The evidence is deduplicated and added to the current evidence set, while the action and feedback are recorded in the interaction history. RouterLM then assesses whether another operation is needed. If so, it may reformulate the request or select a different operation in the next round.

Context acceptance. Once RouterLM determines that the constructed evidence is sufficient for final inference, it selects Accept_Context. This terminates evidence construction and passes the constructed evidence to AnswerLM for solution generation.

3.4 RECAST Training

In RECAST, training updates only RouterLM. The primitive operations are deterministic, and the parameters of CompilerLM and AnswerLM remain fixed. Training has two stages. We apply supervised fine-tuning (SFT) to establish valid multi-round evidence-construction behavior, including operation selection, task-specific formulation, and evidence assessment. We apply Group Relative Policy Optimization (GRPO) to improve the routing policy using downstream task outcomes. The complete dataset and training details of SFT and GRPO are provided in Appendix B.1.

Trajectory collection and SFT. Using the pretrained RouterLM, we generate multiple trajectories for each training question. Each trajectory executes its evidence operations and invokes AnswerLM with the constructed evidence. A frozen LLM judge then evaluates the solution against the reference. For each question with at least one correct, executable, and structurally valid trajectory, we select one trajectory as the SFT expert, preferring more accurate and efficient candidates. We apply capped upsampling by repeating examples from underrepresented benchmark families and empirical difficulty levels. The complete expert trajectory is used for SFT, supervising each RouterLM decision conditioned on the evidence and feedback available at that round.

GRPO. We construct the GRPO data from questions that were not consistently solved during SFT trajectory collection. Each selected question must have at least one valid expert trajectory, indicating that it is solvable, and at least one failed trajectory, indicating room for policy improvement. Starting from the SFT checkpoint, we sample a new group of four complete trajectories for each selected question. We then apply *mixed-outcome filtering*, retaining only groups containing at least one successful and one unsuccessful trajectory. This allows GRPO to compare alternative evidence-construction behaviors for the same question and reinforce those that produce better task outcomes.

For a trajectory τ, we define the reward as

$$R(\tau) = \lambda_C C(\tau) + \lambda_F F(\tau) + \lambda_V V(\tau), \tag{4}$$

where $C(\tau)$ denotes final-answer correctness evaluated by the frozen LLM judge, $F(\tau)$ is token-level F1 between the generated and reference answers, and $V(\tau)$ indicates whether the trajectory follows the required action structure. We set $\lambda_C = 0.90$, $\lambda_F = 0.08$, and $\lambda_V = 0.02$, prioritizing task correctness while retaining smaller signals for partial answer quality and structural validity.

4 Experimental Results

4.1 Experimental Setup

Benchmarks. We train and evaluate RECAST on six benchmark families with heterogeneous source representations. DataBench pairs questions with dataframes from diverse domains (Grijalba et al., 2024). FinQA contains financial reports combining prose passages and tables (Chen et al., 2021). HiTab provides tables with hierarchical row and column headers (Cheng et al., 2022a), while MultiHierTT associates questions with multiple hierarchical tables and passages from financial reports (Zhao et al., 2022). HotpotQA presents each question with Wikipedia passages (Yang et al., 2018). LaMP pairs an input with a user profile of historical items for personalization (Salemi et al., 2024). We test on 100 unseen questions per family.

Held-out benchmarks. To evaluate zero-shot generalization, we use three benchmarks excluded from SFT, GRPO, and checkpoint selection. 2WikiMultiHopQA pairs questions with collections of Wikipedia passages (Ho et al., 2020). TAT-QA contains financial reports combining prose passages and tables (Zhu et al., 2021), while WikiTableQuestions pairs natural-language questions with semi-structured Wikipedia tables (Pasupat & Liang, 2015). We evaluate on 100 examples per benchmark. All nine benchmark descriptions and examples appear in Appendices A.1 and A.2.

Baselines. We compare RECAST against five baseline families spanning increasingly expressive forms of context construction: **(1) Direct** provides only the task to AnswerLM, without accessing the source; **(2) Fixed Retrieval** uses the task as the query for a single BGE-M3 dense retrieval and returns the top three source chunks; **(3) Direct Code** generates and executes a single Python program over the source without iterative evidence construction; **(4) IRCoT** (Trivedi et al., 2023) alternates sentence-level reasoning with BGE-M3 retrieval; and **(5) Interact-RAG** (Hui et al., 2026) uses a planner–reasoner–executor workflow to formulate retrieval actions and decide when to stop. IRCoT and Interact-RAG are adapted to the per-instance sources and shared model and retrieval components used in our evaluation; further implementation details are provided in Appendix C.1. For Direct Code, IRCoT, and Interact-RAG, we evaluate Qwen3.5-9B and Gemini 3.5 Flash without task-specific fine-tuning as their respective program-generation or retrieval-guidance LMs.

RECAST variants. We evaluate RECAST along five dimensions: **(1) RouterLM configurations** compare training-free Qwen3.5-9B and Gemini 3.5 Flash routers with an SFT+GRPO-trained

Table 1: **In-domain success rates (%) of RECAST across six benchmark families.** The listed LM directly generates the program for Direct Code, guides evidence retrieval for IRCoT and Interact-RAG, and serves as RouterLM for RECAST. Bold denotes the best result in each column.

Method	LM	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Retrieval baselines								
Direct	–	6.7	1.0	23.0	41.7	69.7	1.0	23.8
Fixed Retrieval	–	36.0	27.3	72.3	79.3	64.0	55.3	55.7
Direct program generation								
Direct Code	Qwen3.5-9B	80.7	1.0	26.0	64.0	54.7	3.3	38.3
Direct Code	Gemini 3.5 Flash	88.0	5.0	46.0	78.3	48.0	5.0	45.1
Iterative and agentic retrieval								
IRCoT	Qwen3.5-9B	36.0	27.0	64.0	94.0	65.0	56.0	57.0
IRCoT	Gemini 3.5 Flash	39.3	30.0	63.7	**94.7**	67.3	58.0	58.8
Interact-RAG	Qwen3.5-9B	30.0	45.0	43.0	87.0	63.0	56.0	54.0
Interact-RAG	Gemini 3.5 Flash	36.7	52.7	72.7	79.7	62.3	54.0	59.7
RECAST								
RECAST	Qwen3.5-9B	86.0	57.0	60.7	83.0	68.3	29.3	64.1
RECAST	Gemini 3.5 Flash	**90.0**	58.3	67.3	86.0	**71.7**	50.3	70.6
RECAST	Qwen3.5-9B (SFT+GRPO)	83.3	**68.3**	**76.3**	92.0	68.3	**65.3**	**75.6**

Qwen3.5-9B router; **(2) evidence-operation space** restricts training-free RECAST to either primitive operations or synthesized code while retaining full multi-round routing; **(3) GRPO reward and data selection** replaces the shaped reward with answer correctness alone or removes mixed-outcome filtering; **(4) training stages** applies SFT and GRPO independently; and **(5) compiler transfer** replaces CompilerLM with alternative models. Details are in Appendix C.2.

Implementation and Evaluation. Semantic retrieval uses BGE-M3, lexical retrieval uses BM25, and relational operations execute read-only SQLite queries. For RECAST, Gemini 3.5 Flash serves as CompilerLM. Training uses 3,368 SFT trajectories and 608 GRPO instances. We train RouterLM with LoRA (Hu et al., 2021). Qwen3.5-9B has thinking mode disabled.

All methods are evaluated on same test questions and source contexts. Gemini 3.5 Flash serves as the shared AnswerLM for all methods. All outputs are scored by the same frozen Gemini 3.5 Flash judge using an identical prompt. LLM calls use temperature 0, and multi-round methods are limited to six rounds and 4,000 characters of retained evidence. We repeat each in-domain experiment three times and report mean task success rate. Standard deviations, prompt, additional implementation, training, and evaluation details are provided in Appendices B.2, B.1, and F.

4.2 RECAST Performance

Table 1 compares success rates of RECAST and baselines across six benchmarks. Appendices B.3, D, and E provide operation-usage, cost, and failure analyses. There are four main takeaways:

First, RECAST achieves the strongest overall performance. The SFT+GRPO-trained system obtains an average success rate of 75.6%, a 15.9% improvement over the strongest large-LLM-controlled baseline, Gemini-based Interact-RAG at 59.7%. This result reflects strong performance across diverse source formats and evidence requirements. Re-evaluating predictions with two alternative LLM judges and token-level F1 preserves the overall performance advantage (Appendix B.4).

Second, the design of RECAST provides substantial gains even without RouterLM training. With Qwen3.5-9B, training-free RECAST achieves 64.1%, compared with 57.0% for the strongest Qwen-based baseline. With Gemini 3.5 Flash, training-free RECAST reaches 70.6%, compared with 59.7% for the strongest Gemini-based baseline. Notably, training-free RECAST with Qwen3.5-9B as RouterLM also outperforms IRCoT and Interact-RAG using Gemini 3.5 Flash as their retrieval controller. By supporting adaptive retrieval, computation, and evidence assessment, RECAST provides gains beyond those obtained by equipping the retrieval baselines with a stronger controller.

Third, training substantially strengthens the evidence-routing capability of the smaller model. SFT and GRPO improve Qwen3.5-9B from 64.1% to 75.6%. The small, trained Qwen router also

Table 2: **Success rates (%) of RECAST ablations across six benchmark families.** The variants evaluate the evidence-operation space, training stages, GRPO reward, and GRPO data selection. Bold denotes the best result in each column.

Variant	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Evidence-operation space							
RECAST Train-Free, Primitive Only	25.3	61.0	63.7	80.7	67.7	42.7	56.8
RECAST Train-Free, Synthesis Only	85.3	45.0	54.3	88.7	63.0	23.0	59.9
RECAST Train-Free, Full	**86.0**	57.0	60.7	83.0	68.3	29.3	64.1
GRPO reward and data selection							
RECAST SFT+GRPO, Accuracy Only	82.0	58.0	65.0	87.3	66.0	55.0	68.9
RECAST SFT+GRPO, Unfiltered	83.7	62.0	74.0	85.3	**69.3**	63.7	73.0
Training stages							
RECAST SFT	82.0	55.7	68.3	82.7	66.3	49.0	67.3
RECAST GRPO	81.7	64.0	75.7	88.3	64.0	61.7	72.6
RECAST SFT+GRPO	83.3	**68.3**	**76.3**	**92.0**	68.3	**65.3**	**75.6**

outperforms the large, training-free Gemini 3.5 Flash router by 5.0% and achieves higher performance on four of the six benchmarks. Thus, SFT and GRPO enable a smaller, trainable model to surpass a larger black-box model by learning specialized evidence-routing behavior.

Fourth, heterogeneous tasks benefit from adaptive evidence construction rather than a single strategy. The strongest program generation and agentic retrieval baselines exhibit different strengths across benchmarks. Gemini-based Direct Code excels on DataBench but obtains only 45.1% on average, whereas Gemini-based Interact-RAG obtains 59.7% with a clear advantage on FinQA, HiTab, and MultiHiertt. In comparison, RECAST reaches 75.6% and maintains more balanced performance across the six benchmark families. By adaptively supporting primitive retrieval and computation together with synthesized code, RECAST provides a more flexible evidence-construction process across heterogeneous tasks.

4.3 Ablation Studies

We ablate the evidence-operation space, the GRPO reward and data-selection strategy, and the contributions of the two training stages. Table 2 summarizes the results. There are three main takeaways:

First, primitive and synthesized operations provide complementary evidence-construction capabilities. Primitive-only RECAST achieves 56.8%, and synthesis-only 59.9%. Neither operation type is uniformly superior across families, indicating that effectiveness depends on the task's evidence requirements. Combining both yields the strongest training-free average of 64.1%, outperforming either variant and demonstrating the benefit of a broader evidence-operation space.

Second, reward shaping and selective GRPO data construction contribute to performance. Using answer correctness as the only reward reduces the average from 75.6% to 68.9%, showing that partial answer quality and structural validity provide useful distinctions among trajectories. Removing the mixed-outcome group filter yields 73.0%, indicating that groups containing both successful and unsuccessful trajectories provide a stronger relative learning signal. Because filtering produces a smaller training pool, its higher performance demonstrates improved sample efficiency.

Third, SFT and GRPO provide complementary improvements. Relative to training-free RECAST at 64.1%, SFT increases the average to 67.3%, while GRPO increases it to 72.6%. Combining both yields the strongest result at 75.6%, exceeding either stage and demonstrating their complementary benefits. SFT establishes valid operation selection and formulation from filtered successful trajectories, while GRPO optimizes routing trajectories using downstream outcomes.

4.4 Generalization

4.4.1 Generalization to unseen benchmarks

Table 3 reports zero-shot performance on three held-out benchmarks: 2WikiMultiHopQA (Ho et al., 2020), TAT-QA (Zhu et al., 2021), and WikiTableQuestions (Pasupat & Liang, 2015). These benchmarks cover multi-hop text question answering, hybrid text-table numerical reasoning, and table

question answering, respectively. Here, Direct Code, IRCoT, and Interact-RAG use the pretrained Qwen3.5-9B as their program-generation or retrieval-guidance model, while RECAST uses the SFT+GRPO-trained Qwen3.5-9B RouterLM. There are two main takeaways. **First, RECAST generalizes consistently across all three held-out benchmarks.** It achieves an average success rate of 79.3%, outperforming the strongest baseline at 64.3% by 15.0%. The improvements over the strongest baseline on each benchmark are 6.0% on 2WikiMultiHopQA, 8.0% on TAT-QA, and 15.0% on WikiTableQuestions. **Second, the learned routing policy generalizes across heterogeneous source representations.** The improvements span collections of Wikipedia passages, financial reports combining text and tables, and semi-structured Wikipedia tables. These results show that evidence-routing behavior learned from the in-domain benchmarks transfers to new task distributions and source organizations.

4.4.2 Transfer across CompilerLMs

We keep the trained RouterLM fixed and replace the default Gemini 3.5 Flash CompilerLM with Gemini 3.5 Flash Lite and Qwen3.5-9B, neither of which is used as a compiler during training. Table 4 reports the results. There are two main takeaways. **First, the learned routing policy transfers effectively to a lighter CompilerLM.** Gemini 3.5 Flash Lite achieves 73.2% in-domain, a decrease of only 2.4% from the default compiler, while matching its held-out performance at 79.3%. **Second, the routing policy also transfers across model families and substantially different model scales.** With Qwen3.5-9B as CompilerLM, RECAST achieves 67.2% in-domain and 73.3% on the held-out benchmarks, remaining 7.5% and 9.0% above the strongest evaluated baselines, respectively. These results show that the learned RouterLM does not depend on a single compiler model and can retain strong performance with substantially more compact compilers.

Table 3: **Zero-shot success rate (%) of RECAST across three unseen benchmarks.**

Method	2Wiki	TAT-QA	WTQ	Avg.
Direct	72.0	1.0	10.0	27.7
Direct Code	45.0	38.0	59.0	47.3
Fixed Retrieval	58.0	66.0	52.0	58.7
IRCoT	69.0	73.0	51.0	64.3
Interact-RAG	77.0	57.0	48.0	60.7
RECAST $_{\text{SFT+GRPO}}$	**83.0**	**81.0**	**74.0**	**79.3**

Table 4: **RECAST in-domain and held-out success rates (%) with alternative CompilerLMs.**

CompilerLM	In-domain avg.	Held-out avg.
Qwen3.5-9B	67.2	73.3
Gemini 3.5 Flash Lite	73.2	**79.3**
Gemini 3.5 Flash	**75.6**	**79.3**

5 Conclusion

We introduced RECAST, a learned framework that broadens context construction from retrieving source content to adaptively deriving task-relevant evidence through retrieval and computation. Trained through SFT and GRPO, a lightweight RouterLM iteratively formulates primitive operations or requests customized code, assesses their outcomes, and determines when the evidence is sufficient for final inference. Across six in-domain benchmarks, RECAST outperforms retrieval-based, agentic, and direct program-generation approaches on average. It generalizes to three held-out benchmarks and retains strong performance with smaller code-generation models. Our ablations show that primitive and synthesized operations provide complementary capabilities and that learning strengthens the evidence-construction policy. These results demonstrate the effectiveness and generalizability of adaptive evidence construction across heterogeneous tasks and sources.

Limitations. Constructing the training data for RECAST relies on execution-derived outcomes, which introduces additional computational overhead during data collection. At inference time, iterative calls to RouterLM and occasional code synthesis may also increase latency and computational cost relative to single-pass retrieval. Future work could explore more efficient trajectory construction and cost-aware routing policies that jointly optimize task performance and computational efficiency.

AI USE STATEMENT

We used generative AI tools in three ways. First, as writing aids: we provided draft text for grammatical correction and structural refinement, and the authors verified and edited all outputs. Second, for retrieval and discovery, such as finding potentially related work. The authors read all cited papers and verified every reference. Third, to assist research execution, including environment setup and experiment code debugging. All AI-assisted code was reviewed and tested by the authors, and we take full responsibility for the content of this paper.

ETHICS STATEMENT

This work improves how language models construct evidence from heterogeneous sources. All experiments use publicly available benchmarks, and no new human-subject data was collected. Because RECAST executes model-generated code, relational queries are read-only and synthesized programs run in an isolated environment. When applied to private or sensitive data, deployments should enforce similar sandboxing along with appropriate access controls and privacy safeguards.

REPRODUCIBILITY STATEMENT

For reproducibility, we describe the full RECAST framework and training procedure in Section 3. Dataset construction, SFT and GRPO hyperparameters, the GRPO objective, and the reward definition are detailed in Appendix B.1, and standard deviations over three runs are reported in Appendix B.2. Baseline and variant implementations are described in Appendices C.1 and C.2, and all prompts are provided in Appendix F. Our code, data, and trained ROUTERLM checkpoint will be publicly released under an open-source license upon acceptance.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, volume 2024, pp. 54067–54089, 2024.

Jianlyu 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. In *Findings of the association for computational linguistics: ACL 2024*, pp. 2318–2335, 2024.

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

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, 2021.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. Hitab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 1094–1110, 2022a.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022b.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Carlos Gemmell and Jeff Dalton. Toolwriter: Question specific tool synthesis for tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 16137–16148, 2023.

Jorge Osés Grijalba, L Alfonso Urena Lopez, Eugenio Martínez-Cámara, and Jose Camacho-Collados. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 13471–13488, 2024.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4320–4333, 2020.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Yulong Hui, Chao Chen, Zhihang Fu, Yihao Liu, Jieping Ye, and Huanchen Zhang. Interact-rag: Reason and interact with the corpus, beyond black-box retrieval. In *International Conference on Learning Representations*, volume 2026, pp. 112154–112174, 2026.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th conference of the european chapter of the association for computational linguistics: main volume*, pp. 874–880, 2021.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 7036–7050, 2024.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, 2024.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 9237–9251, 2023a.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7969–7992, 2023b.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Omar Khattab, Christopher Potts, and Matei Zaharia. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. *Advances in Neural Information Processing Systems*, 34:27670–27682, 2021.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhao-Xiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36:4952–4984, 2023.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*, 2021.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1470–1480, 2015.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 6922–6939, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7370–7392, 2024.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*, volume 2024, pp. 32628–32649, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

Zilong Ryan Wang, Hao Zhang, Chun-Liang Li, Julian M Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *International Conference on Learning Representations*, volume 2024, pp. 55587–55610, 2024.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Wenqi Zhang, Yongliang Shen, Zeqi Tan, Guiyang Hou, Weiming Lu, and Yueting Zhuang. Datacopilot: Bridging billions of data and humans with autonomous workflow. *arXiv preprint arXiv:2306.07209*, 2023.

Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. Multihiertt: Numerical reasoning over multi hierarchical tabular and textual data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6588–6600, 2022.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th annual meeting of the Association for Computational Linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, pp. 3277–3287, 2021.

APPENDIX–RECAST: LEARNING TO COMPUTE THE RIGHT CONTEXT THROUGH ADAPTIVE EVIDENCE ROUTING

A Benchmarks **15**

A.1 Benchmark Descriptions 15

A.2 Benchmark Examples 16

B RECAST **19**

B.1 RECAST Training Details 19

B.2 RECAST Main-Result Performance with Standard Deviations 21

B.3 Evidence-Construction Behavior 22

B.4 Robustness to Evaluation Choices 23

C Baselines and RECAST Variants **26**

C.1 Baselines 26

C.2 RECAST Variants 26

D Cost Analysis **28**

E Failure Case Analysis **29**

E.1 Baselines 29

E.2 RECAST 29

F Prompts **30**

F.1 ROUTERLM 30

F.2 COMPILERLM 31

F.3 ANSWERLM 32

F.4 Correctness Judge 33

A BENCHMARKS

A.1 BENCHMARK DESCRIPTIONS

We use six benchmark families for training and in-domain evaluation and three additional benchmarks for zero-shot evaluation.

Training and in-domain benchmarks:

• **DataBench.** DataBench evaluates question answering over real-world dataframes from diverse application domains (Grijalba et al., 2024). Questions require analyzing the provided table and returning either a scalar or a list.

• **FinQA.** FinQA contains numerical reasoning questions grounded in financial reports composed of prose passages and structured tables (Chen et al., 2021). Answering a question may require identifying relevant information from either source and performing arithmetic over the extracted values.

• **HiTab.** HiTab evaluates question answering over cross-domain tables with hierarchical row and column headers (Cheng et al., 2022a). Its questions cover lookup, comparison, aggregation, and numerical reasoning while requiring the table's multi-level structure to be correctly interpreted.

• **HotpotQA.** HotpotQA is a multi-hop question-answering benchmark built from Wikipedia articles (Yang et al., 2018).

• **LaMP.** LaMP evaluates personalized language modeling using a current input and a profile containing historical items from the same user (Salemi et al., 2024). We combine LaMP-1, 2, 3, 4, 5, and 7 into one benchmark family, covering citation identification, movie tagging, product rating, news headline generation, scholarly title generation, and tweet paraphrasing. We exclude LaMP-6 because it relies on the restricted Avocado email corpus.

• **MultiHiertt.** MultiHiertt contains numerical reasoning questions grounded in financial reports with multiple hierarchical tables and accompanying text passages (Zhao et al., 2022). Questions may require combining evidence across multiple tables or between tabular and textual content.

Held-out benchmarks:

• **2WikiMultiHopQA.** 2WikiMultiHopQA evaluates multi-hop question answering over collections of Wikipedia passages (Ho et al., 2020). Its questions require reasoning across multiple documents through compositional, comparison, or inference relations.

• **TAT-QA.** TAT-QA contains questions grounded in financial reports that combine prose passages and tables (Zhu et al., 2021). It includes span-based, counting, and arithmetic questions, with answers expressed using different numerical scales.

• **WikiTableQuestions.** WikiTableQuestions evaluates compositional question answering over semi-structured Wikipedia tables (Pasupat & Liang, 2015). Questions may require selecting, comparing, or aggregating values from multiple table entries.

A.2 BENCHMARK EXAMPLES

The following examples illustrate the task and source format of each benchmark. Because complete sources may contain many records, we show only shortened, representative snippets. The benchmarks span heterogeneous sources, including text passages, flat and hierarchical tables, financial reports, dataframes, and user profiles. Their questions also require different forms of evidence, ranging from direct retrieval and multi-hop linking to aggregation, numerical reasoning, and personalized prediction or generation.

DataBench

Query:
Which month sees the highest number of complaints?

Source snippet (CSV):
```
complaint_type,borough,hour,month_name,agency
HEAT/HOT WATER,MANHATTAN,18,March,HPD
...
[additional dataframe rows omitted]
```

Answer:
January

FinQA

Query:
What percentage of the contractual obligations due in 2008 are maturities of long-term debt?

Source snippet (financial report):
Contractual obligations for future payments at December 31, 2007 were as follows (in millions):

	2008	2009	2010
Maturities of long-term debt	267	1,300	1,069
Debt obligations with right of offset	2,013	2,013	2,013

...

Answer:
11%

HiTab

Query:
In 2018, which club did Russell sign with Major League Soccer?

Source snippet (hierarchical table row):
```
{
  "row_headers": ["sporting kansas city"],
  "values": [
    {"column_headers": ["season"], "value": "2018"},
    {"column_headers": ["league", "division"],
     "value": "major league soccer"},
    ...
  ]
}
```

Answer:
["sporting kansas city"]

HotpotQA

Query:
```
Where is the head office of the airway on whose Board of Directors
Wanjiku Mugane serves?
```

Source snippet (Wikipedia passages):
```
Title: Wanjiku Mugane
"... member of the board of directors of Kenya Airways ..."

Title: Kenya Airways
"The carrier's head office is located in Embakasi, Nairobi,
with its hub at Jomo Kenyatta International Airport."
```

Answer:
```
Embakasi, Nairobi
```

LaMP-1

Query:
```
For the author of "RoboCare: Pervasive Intelligence for the
Domestic Care of the Elderly," which reference is related?

[1] Crowd analysis: a survey
[2] Differential-drive in-pipe robot for moving inside
    urban gas pipelines
```

Source snippet (user profile):
```
{
  "title": "Temporary maps for robust localization in
            semi-static environments",
  "abstract": "Accurate and robust localization is essential
               for autonomous mobile robots. ..."
}
```

Answer:
```
[2]
```

MultiHiertt

Query:
```
What is the average amount of Balance forward of SquareFeet
and 2015 of Mexico UnitCount?
```

Source snippet (hierarchical HTML table):
```
<tr>
  <td>Balance forward</td>
  <td></td>
  <td></td>
  <td>632</td>
  <td>84,382</td>
</tr>
...
[additional table required by the question omitted]
```

Answer:
```
43336
```

2WikiMultiHopQA

Query:
Does Lizzie Jelfs have the same nationality as Paul S. Devrouax?

Source snippet (Wikipedia passages):
Title: Lizzie Jelfs
"Elizabeth Coulson (nee Jelfs) is a British former professional tennis player."

Title: Paul S. Devrouax
"Paul S. Devrouax was an American architect."

Answer:
no

TAT-QA

Query:
What is the percentage change in cash and short-term marketable securities from 2018 to 2019?

Source snippet (financial table row):
```
{
  "item": "Cash and short-term marketable securities",
  "2019": "$454",
  "2018": "$524",
  "2017": "$656"
}
```

Answer:
-13.36%

WikiTableQuestions

Query:
Which actor has the most NAACP Image Awards?

Source snippet (Wikipedia table):
```
Year | Supporting Actor | Motion Picture
1995 | Al Freeman, Jr.   | Malcolm X
1996 | Laurence Fishburne| Higher Learning
1997 | Samuel L. Jackson | A Time to Kill
1998 | Morgan Freeman    | Amistad
...
[additional table rows omitted]
```

Answer:
Morgan Freeman

B RECAST

B.1 RECAST TRAINING DETAILS

Dataset. We begin with 4,660 unique questions from the six training benchmark families. For each question, we collect two initial candidate trajectories. A third trajectory is generated only when both initial attempts fail, and a fourth is generated only when all preceding attempts fail. This adaptive collection procedure produces 11,563 candidate trajectories in total. Trajectory collection uses a pretrained Qwen3.5-9B ROUTERLM, BGE-M3 for semantic retrieval, and Gemini 3.5 Flash as COMPILERLM, ANSWERLM, and the correctness judge.

We retain only trajectories that produce a correct final answer, complete without parsing or execution failure, satisfy the action schema, and conform to the final RECAST action space. When several valid candidates are available for the same question, we retain the trajectory with stronger native or exact answer agreement, followed by higher token F1 and lower execution and token cost. This process produces 3,064 questions with expert trajectory. We reserve approximately 10% of these questions (306 questions) for validation, leaving 2,758 unique questions for SFT. We apply capped upsampling by repeating examples from underrepresented benchmark families and empirical difficulty levels, with at most two copies of any example. The resulting SFT corpus contains 3,368 complete trajectories.

For GRPO, we begin with 1,200 questions that are neither consistently solved nor entirely unsolved during initial trajectory collection. Each has at least one valid expert and at least one failed attempt. From the selected SFT checkpoint, we sample four new trajectories per question and apply *mixed-outcome filtering*, retaining only groups containing at least one successful and one unsuccessful trajectory. After structural validation and benchmark-family balancing through repetition, the GRPO training set contains 608 prompt rows derived from 567 unique questions. A separate set of 300 questions, with 50 from each benchmark family, is used for GRPO validation. Table 5 summarizes the resulting datasets.

Table 5: **Composition of the trajectory collection and the final SFT and GRPO training sets.**

Family	Questions	Raw traj.	SFT rows	GRPO rows
DataBench	885	2,080	665	97
FinQA	410	1,109	349	101
HiTab	935	2,381	567	96
HotpotQA	785	1,674	660	96
LaMP	735	1,814	519	95
MultiHiertt	910	2,505	608	123
Total	4,660	11,563	3,368	608

SFT for RECAST $_{\text{SFT+GRPO}}$. We fine-tune Qwen3.5-9B using LoRA (Hu et al., 2021) on eight A100 80GB GPUs. LoRA is applied to all linear layers with rank 32, scaling parameter $\alpha_{\text{LoRA}} = 64$, and dropout 0.05. We train for two epochs with a global batch size of 16, a maximum sequence length of 16,384 tokens, and a learning rate of 5×10^{-5}. The loss supervises all ROUTERLM decisions across each multi-round trajectory, and we select the checkpoint with the best development performance.

GRPO for RECAST $_{\text{SFT+GRPO}}$. We continue LoRA fine-tuning from the selected SFT checkpoint using eight A100 80GB GPUs. GRPO uses a learning rate of 2×10^{-6}, a batch size of eight prompts, and four sampled routing trajectories per prompt. Trajectories are sampled with temperature 0.8 and top-$p = 0.95$. We train the model for three epochs.

GRPO Objective. For each prompt x, GRPO (Shao et al., 2024) samples a group of $G = 4$ trajectories $\{y_i\}_{i=1}^{G}$ with rewards $\{r_i\}_{i=1}^{G}$. The group-relative advantage is

$$A_i = \frac{r_i - \bar{r}}{\text{std}(r_1, \ldots, r_G) + \delta}, \qquad \bar{r} = \frac{1}{G} \sum_{j=1}^{G} r_j, \tag{5}$$

where δ is a small constant for numerical stability. Let

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t} \mid x, y_{i,<t})} \tag{6}$$

denote the importance ratio at token t. The optimization objective is

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{i,t}[\min(\rho_{i,t}(\theta)A_i,\ \text{clip}(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\, A_i)] \\ - \beta D_{\text{KL}}^{\text{low-var}}(\pi_\theta \,\|\, \pi_{\text{ref}})\,, \end{aligned} \tag{7}$$

where π_{old} is the rollout policy and π_{ref} is the fixed reference policy. We use $\epsilon = 0.2$ and $\beta = 0.01$. The terminal trajectory reward is applied to all generated tokens.

GRPO Reward Explanation. The terminal reward for a routing trajectory τ is:

$$R(\tau) = 0.90\, C(\tau) + 0.08\, F(\tau) + 0.02\, V(\tau), \tag{8}$$

where $C(\tau) \in \{0, 1\}$ is final-answer correctness determined by the frozen judge, $F(\tau) \in [0, 1]$ is the token-level F1 between the normalized generated and reference answers, and $V(\tau) \in \{0, 1\}$ indicates that the trajectory follows the required action schema and completes without a pipeline failure. The weights are chosen so that final-answer correctness dominates the reward: every correct trajectory receives a higher reward than any incorrect trajectory. Token-level F1 and structural validity serve only as bounded secondary signals for distinguishing trajectories with the same correctness outcome. Although these secondary terms carry only 10% of the reward weight, they are the only signal that separates trajectories with the same correctness outcome. With $R(\tau) = C(\tau)$, all correct trajectories in a group receive identical advantages, as do all incorrect ones, so GRPO cannot distinguish a near-miss answer from a malformed or failed trajectory. The accuracy-only ablation uses $R(\tau) = C(\tau)$.

B.2 RECAST Main-Result Performance with Standard Deviations

Table 6 reports the sample standard deviations corresponding to the mean success rates in Table 1. Statistics are computed over three complete evaluations of the same 600-question test set. Although all language-model calls use temperature 0, outputs can still vary across runs due to nondeterminism in hosted model APIs and batched GPU inference. In multi-round methods, such variation can change an intermediate routing or retrieval decision and propagate to later rounds. Across methods, the overall standard deviation ranges from 0.3% to 0.8%, indicating stable aggregate performance across repeated evaluations.

Table 6: **Standard deviations of in-domain success rates (%) across three evaluations.** The corresponding mean results are reported in Table 1.

Method	LM	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Retrieval baselines								
Direct	–	1.2	0.0	1.0	3.8	1.5	0.0	0.4
Fixed Retrieval	–	1.0	1.2	0.6	1.5	1.7	1.5	0.3
Direct program generation								
Direct Code	Qwen3.5-9B	0.6	1.0	1.0	0.0	1.5	0.6	0.5
Direct Code	Gemini 3.5 Flash	1.7	2.0	1.7	0.6	1.0	0.0	0.4
Iterative and agentic retrieval								
IRCoT	Qwen3.5-9B	2.0	1.5	0.6	0.0	1.0	1.2	0.8
IRCoT	Gemini 3.5 Flash	0.6	1.0	1.2	0.6	1.5	3.5	0.8
Interact-RAG	Qwen3.5-9B	1.5	2.3	2.5	0.6	1.5	1.5	0.7
Interact-RAG	Gemini 3.5 Flash	2.9	2.1	2.3	2.1	1.5	1.7	0.3
RECAST								
RECAST	Qwen3.5-9B	1.0	1.0	2.5	3.6	0.6	1.5	0.5
RECAST	Gemini 3.5 Flash	1.0	1.5	2.3	1.0	2.5	1.5	0.5
RECAST	Qwen3.5-9B (SFT+GRPO)	1.5	3.1	0.6	2.6	2.3	2.5	0.5

B.3 EVIDENCE-CONSTRUCTION BEHAVIOR

Table 7 summarizes the trained RECAST policy's operation usage and routing rounds on the 600-question in-domain test set. We report primitive and synthesis calls per question, the percentage of questions invoking synthesis, and average routing rounds. Routing rounds include evidence-operation requests and context-acceptance decisions.

Table 7: **Operation usage and routing rounds of trained RECAST.** Statistics are computed from one evaluation run, with 100 questions per benchmark. Calls and rounds are averaged per question; synthesis usage denotes the percentage of questions with at least one synthesis call.

Benchmark	Primitive calls	Synthesis calls	Synthesis usage (%)	Routing rounds
DataBench	0.04	1.12	98.0	2.16
FinQA	3.00	0.22	21.0	4.22
HiTab	2.60	0.07	6.0	3.67
HotpotQA	1.46	0.00	0.0	2.43
LaMP	2.89	0.04	4.0	3.93
MultiHiertt	2.11	0.40	35.0	3.50
Overall	2.02	0.31	27.3	3.32

The learned policy exhibits distinct operation-use patterns across benchmarks. DataBench predominantly uses synthesized computation, whereas HotpotQA relies entirely on primitives in this run. FinQA and MultiHiertt combine substantial primitive use with synthesis on a subset of questions, while HiTab and LaMP invoke synthesis less frequently. Overall, synthesis is used on 27.3% of questions, showing that the framework does not universally delegate evidence construction to COMPILERLM. Routing length also varies across benchmarks, from 2.16 rounds on DataBench to 4.22 on FinQA. These observations complement the operation-space ablations by showing how the learned policy uses different evidence operations and interaction lengths across heterogeneous tasks.

B.4 Robustness to Evaluation Choices

Because Gemini 3.5 Flash provides correctness judgments during both training and evaluation, we examine whether the reported improvements persist under alternative evaluators. We re-score the saved predictions from one inference replicate per method on the same 600-question in-domain test set, without regenerating answers or retraining models. We use GPT-4.1 (`gpt-4.1-2025-04-14`) and DeepSeek-V4-Flash (`deepseek-ai/DeepSeek-V4-Flash-0731`) with the same correctness prompt as the original judge. We additionally compute token-level F1 using SQuAD-style normalization.

Alternative LLM judges. Tables 8 and 9 report the re-evaluated success rates. Trained RECAST achieves 74.8% under GPT-4.1 and 72.8% under DeepSeek-V4-Flash, exceeding the strongest baseline by 15.8% and 15.3%, respectively. Although absolute scores vary across judges, RECAST remains the strongest method overall under both evaluators. These results indicate that its performance advantage is not specific to evaluation by the model used for training-time correctness judgments.

Table 8: **In-domain success rates (%) judged by GPT-4.1.** Same runs as Table 1 (one replicate per row), re-judged with `gpt-4.1-2025-04-14` using the identical judge prompt. Bold denotes the best result in each column.

Method	LM	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Retrieval baselines								
Direct	–	7.0	1.0	21.0	39.0	**73.0**	1.0	23.7
Fixed Retrieval	–	35.0	26.0	68.0	79.0	68.0	57.0	55.5
Direct program generation								
Direct Code	Qwen3.5-9B	70.0	3.0	21.0	62.0	57.0	0.0	35.5
Direct Code	Gemini 3.5 Flash	77.0	7.0	41.0	78.0	47.0	5.0	42.5
Iterative and agentic retrieval								
IRCoT	Qwen3.5-9B	33.0	27.0	61.0	94.0	68.0	57.0	56.7
IRCoT	Gemini 3.5 Flash	36.0	28.0	59.0	**96.0**	70.0	53.0	57.0
Interact-RAG	Qwen3.5-9B	30.0	44.0	39.0	87.0	65.0	55.0	53.3
Interact-RAG	Gemini 3.5 Flash	38.0	53.0	70.0	78.0	62.0	53.0	59.0
RECAST								
RECAST	Qwen3.5-9B	85.0	54.0	58.0	77.0	69.0	29.0	62.0
RECAST	Gemini 3.5 Flash	**86.0**	59.0	62.0	85.0	69.0	51.0	68.7
RECAST	Qwen3.5-9B (SFT+GRPO)	84.0	**69.0**	**73.0**	88.0	67.0	**68.0**	**74.8**

Table 9: **In-domain success rates (%) judged by DeepSeek-V4-Flash.** Same runs as Table 1 (one replicate per row), re-judged with `deepseek-ai/DeepSeek-V4-Flash-0731` using the identical judge prompt and decoding at temperature 0. Bold denotes the best result in each column.

Method	LM	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Retrieval baselines								
Direct	–	7.0	1.0	19.0	38.0	68.0	1.0	22.3
Fixed Retrieval	–	36.0	26.0	66.0	77.0	62.0	56.0	53.8
Direct program generation								
Direct Code	Qwen3.5-9B	79.0	2.0	23.0	64.0	56.0	2.0	37.7
Direct Code	Gemini 3.5 Flash	**87.0**	6.0	42.0	78.0	45.0	5.0	43.8
Iterative and agentic retrieval								
IRCoT	Qwen3.5-9B	35.0	25.0	59.0	92.0	63.0	55.0	54.8
IRCoT	Gemini 3.5 Flash	37.0	26.0	56.0	**94.0**	67.0	53.0	55.5
Interact-RAG	Qwen3.5-9B	30.0	42.0	40.0	87.0	60.0	55.0	52.3
Interact-RAG	Gemini 3.5 Flash	38.0	48.0	68.0	77.0	61.0	53.0	57.5
RECAST								
RECAST	Qwen3.5-9B	86.0	51.0	56.0	78.0	65.0	29.0	60.8
RECAST	Gemini 3.5 Flash	**87.0**	57.0	60.0	84.0	**69.0**	50.0	67.8
RECAST	Qwen3.5-9B (SFT+GRPO)	83.0	**63.0**	**70.0**	88.0	66.0	**67.0**	**72.8**

Agreement between judges. Tables 10 and 11 compare the alternative judges with Gemini on identical predictions. We report raw agreement and Cohen's κ, which adjusts agreement for the level expected by chance from each judge's correct/incorrect labeling rates. Across methods, Gemini–GPT-4.1 agreement ranges from 95.2% to 98.7%, with Cohen's κ between 0.90 and 0.96. Gemini–DeepSeek agreement ranges from 95.3% to 98.8%, with κ between 0.88 and 0.98. For trained RECAST, agreement is 96.7% with GPT-4.1 and 95.3% with DeepSeek. Both alternative judges assign equal or lower success rates than Gemini, but preserve the overall performance advantage. The Gemini scores here correspond to the selected replicate rather than the three-run means in Table 1.

Table 10: **Agreement between the Gemini judge and GPT-4.1.** Average success rate (%) under each judge on the same 600 predictions per row, raw agreement, and Cohen's κ.

Method	LM	Gemini	GPT-4.1	Agree (%)	κ
Direct	–	23.7	23.7	98.7	0.96
Fixed Retrieval	–	56.0	55.5	96.8	0.94
Direct Code	Qwen3.5-9B	37.7	35.5	95.2	0.90
Direct Code	Gemini 3.5 Flash	44.7	42.5	96.5	0.93
IRCoT	Qwen3.5-9B	57.0	56.7	97.0	0.94
IRCoT	Gemini 3.5 Flash	58.2	57.0	95.8	0.91
Interact-RAG	Qwen3.5-9B	54.0	53.3	96.7	0.93
Interact-RAG	Gemini 3.5 Flash	59.5	59.0	96.8	0.93
RECAST	Qwen3.5-9B	63.5	62.0	97.2	0.94
RECAST	Gemini 3.5 Flash	70.5	68.7	96.2	0.91
RECAST	Qwen3.5-9B (SFT+GRPO)	76.2	74.8	96.7	0.91

Table 11: **Agreement between the Gemini judge and DeepSeek-V4-Flash.** Average success rate (%) under each judge on the same 600 predictions per row, raw agreement, and Cohen's κ.

Method	LM	Gemini	DeepSeek-V4-Flash	Agree (%)	κ
Direct	–	23.7	22.3	98.0	0.94
Fixed Retrieval	–	56.0	53.8	96.5	0.93
Direct Code	Qwen3.5-9B	37.7	37.7	97.0	0.94
Direct Code	Gemini 3.5 Flash	44.7	43.8	98.8	0.98
IRCoT	Qwen3.5-9B	57.0	54.8	97.2	0.94
IRCoT	Gemini 3.5 Flash	58.2	55.5	95.3	0.90
Interact-RAG	Qwen3.5-9B	54.0	52.3	97.3	0.95
Interact-RAG	Gemini 3.5 Flash	59.5	57.5	96.7	0.93
RECAST	Qwen3.5-9B	63.5	60.8	97.0	0.94
RECAST	Gemini 3.5 Flash	70.5	67.8	96.0	0.91
RECAST	Qwen3.5-9B (SFT+GRPO)	76.2	72.8	95.3	0.88

Token-level F1. Table 12 provides a complementary evaluation without an LLM judge. Trained RECAST obtains an average token-level F1 of 57.3%, compared with 45.3% for the strongest baseline. Although token-level F1 contributes to the training reward, its weight is only 0.08, compared with 0.90 for final-answer correctness. The improvement therefore extends beyond the primary training objective. Because token-level F1 measures lexical overlap and can penalize semantically equivalent answers, we report it as a supplementary metric rather than a replacement for correctness evaluation.

Table 12: **In-domain token-level F1 (%) across six benchmark families.** Computed on the same runs as Table 8 with SQuAD-style normalization. Bold denotes the best result in each column.

Method	LM	DataBench	FinQA	HiTab	HotpotQA	LaMP	MultiHiertt	**Avg.**
Retrieval baselines								
Direct	–	6.7	0.7	8.7	37.5	53.9	0.0	17.9
Fixed Retrieval	–	38.6	11.8	47.1	72.6	56.8	32.9	43.3
Direct program generation								
Direct Code	Qwen3.5-9B	69.5	0.0	20.3	61.2	40.0	0.0	31.8
Direct Code	Gemini 3.5 Flash	60.9	1.7	31.2	73.3	43.0	1.7	35.3
Iterative and agentic retrieval								
IRCoT	Qwen3.5-9B	41.6	11.8	40.0	86.3	57.5	32.5	45.0
IRCoT	Gemini 3.5 Flash	39.8	13.7	39.7	**88.4**	**58.2**	31.8	45.3
Interact-RAG	Qwen3.5-9B	31.7	22.4	27.6	80.8	50.0	30.8	40.6
Interact-RAG	Gemini 3.5 Flash	40.2	23.6	47.3	71.1	50.8	28.5	43.6
RECAST								
RECAST	Qwen3.5-9B	78.0	27.2	38.9	73.0	52.6	12.9	47.1
RECAST	Gemini 3.5 Flash	**82.8**	31.6	41.1	80.8	**58.2**	29.5	54.0
RECAST	Qwen3.5-9B (SFT+GRPO)	81.2	**33.6**	**51.2**	82.3	54.2	**41.2**	**57.3**

C BASELINES AND RECAST VARIANTS

All baselines and RECAST variants follow the shared experimental settings described in the main text. We provide their method-specific implementation details below.

C.1 BASELINES

Direct. ANSWERLM receives only the task and produces the final solution without accessing its associated source. This baseline measures performance without context construction.

Fixed Retrieval. The task is used directly as the query for a single BGE-M3 retrieval call. The three highest-ranked text chunks are selected and passed to ANSWERLM, subject to the shared evidence limit. The method performs neither query reformulation nor additional retrieval rounds.

Direct Code. The designated LLM generates a Python program for the task in a single call. The program is executed over the complete source, and its output is passed to the shared ANSWERLM. Unlike RECAST, Direct Code does not select between evidence operations, assess intermediate evidence, or revise its computation over multiple rounds. We evaluate both Qwen3.5-9B and Gemini 3.5 Flash as the program-generating LLM.

IRCoT. IRCoT (Trivedi et al., 2023) alternates retrieval with sentence-level reasoning. At each round, it retrieves three BGE-M3 chunks using either the original task or the latest generated factual sentence as the query. Newly retrieved evidence is added to the working context before the next reasoning step. We retain at most 15 evidence items, use the original IRCoT stopping procedure, and allow at most six rounds. The accumulated evidence is passed to the shared ANSWERLM for final solution generation.

Interact-RAG. Interact-RAG (Hui et al., 2026) uses a Global Planner, Adaptive Reasoner, and Executor to iteratively construct and refine retrieval requests. The Executor may issue up to two searches per round, combining BGE-M3 semantic retrieval with optional BM25 matching and constraints over entities or source records. Each search returns between three and five results, and the workflow runs for at most six rounds before passing its evidence to the shared ANSWERLM.

We adapt IRCoT and Interact-RAG to the per-instance sources, models, and retrieval components used in our evaluation while preserving their central reasoning and control procedures.

C.2 RECAST VARIANTS

Training-free RECAST. We evaluate the complete RECAST framework with either a pretrained Qwen3.5-9B or Gemini 3.5 Flash ROUTERLM. Both variants can invoke the lexical, semantic, and relational primitives, request synthesized code, and assess the resulting evidence over multiple rounds. Neither ROUTERLM receives task-specific training. All remaining framework components are identical to those used by the trained model.

Operation-space ablations. RECAST $_{\text{PRIMITIVE ONLY}}$ removes the SYNTHESIZE action and allows the training-free Qwen3.5-9B ROUTERLM to use only lexical, semantic, and relational primitives. Conversely, RECAST $_{\text{SYNTHESIS ONLY}}$ removes CALL_PRIMITIVE, requiring the same router to construct evidence exclusively through synthesized programs. These variants isolate the contributions of the two evidence-operation families.

SFT for RECAST $_{\text{SFT}}$. We fine-tune pretrained Qwen3.5-9B for five epochs using the same training data and hyperparameters as the SFT stage of RECAST $_{\text{SFT+GRPO}}$. We select the checkpoint with the best development performance across the five epochs.

GRPO for RECAST $_{\text{GRPO}}$. We apply GRPO directly to pretrained Qwen3.5-9B without SFT initialization. Training uses the same mixed-outcome question pool, shaped reward, LoRA configuration, and optimization settings as the GRPO stage of RECAST $_{\text{SFT+GRPO}}$. We use eight A100 80GB GPUs and train for three epochs.

GRPO reward and data ablations. RECAST $_{\text{ACCURACY ONLY}}$ retains the SFT initialization, mixed-outcome training pool, and GRPO configuration but replaces the shaped reward with binary final-answer correctness. RECAST $_{\text{UNFILTERED}}$ retains the SFT initialization and shaped reward but trains on all 1,200 GRPO candidate questions without mixed-outcome filtering.

COMPILERLM variants. The default RECAST configuration uses Gemini 3.5 Flash as COMPILERLM. To evaluate transfer across code-generation models, we replace it with Gemini 3.5 Flash Lite or Qwen3.5-9B while keeping the trained ROUTERLM, ANSWERLM, primitive operations, and execution environment unchanged. The alternative compiler is invoked only when ROUTERLM selects SYNTHESIZE.

D COST ANALYSIS

Table 13 presents average task success rates alongside mean inference token usage per question. Token counts include input and output tokens across routing, code generation, and final answering, but exclude the correctness judge used for evaluation. We separate Qwen3.5-9B and Gemini 3.5 Flash tokens to distinguish computation performed by the smaller and larger models.

Table 13: **Average task success rates (%) and inference token usage per question, in thousands (k).** Token counts include input and output tokens across all inference calls and exclude the evaluation judge. The LM column follows Table 1. Totals are computed before rounding.

Method	LM	Success (%)	Qwen (k)	Gemini (k)	Total (k)
Retrieval baselines					
Direct	–	23.8	0.0	0.6	0.6
Fixed Retrieval	–	55.7	0.0	1.6	1.6
Direct program generation					
Direct Code	Qwen3.5-9B	38.3	4.7	0.0	4.7
Direct Code	Gemini 3.5 Flash	45.1	0.0	4.8	4.8
Iterative and agentic retrieval					
IRCoT	Qwen3.5-9B	57.0	5.1	1.7	6.8
IRCoT	Gemini 3.5 Flash	58.8	0.0	10.0	10.0
Interact-RAG	Qwen3.5-9B	54.0	14.6	1.8	16.5
Interact-RAG	Gemini 3.5 Flash	59.7	0.0	26.9	26.9
RECAST					
RECAST	Qwen3.5-9B	64.1	10.6	7.1	17.7
RECAST	Gemini 3.5 Flash	70.6	0.0	31.8	31.8
RECAST	Qwen3.5-9B (SFT+GRPO)	**75.6**	13.9	5.5	19.4

Trained RECAST achieves higher success with lower token usage than Gemini-based Interact-RAG, obtaining 75.6% versus 59.7% success while using 27.9% fewer total tokens and 79.5% fewer Gemini tokens. Compared with training-free RECAST using a Gemini ROUTERLM, it also improves success from 70.6% to 75.6% while reducing total token usage from 31.8k to 19.4k. These results illustrate the benefit of assigning iterative evidence construction to a trained smaller model, with larger models supporting compilation and final answering.

E FAILURE CASE ANALYSIS

We qualitatively inspect unsuccessful trajectories to identify recurring difficulties and limitations in evidence construction. Direct serves as a reference without source access; the analysis below focuses on methods that retrieve or compute information from the supplied sources.

E.1 BASELINES

Fixed Retrieval. Fixed Retrieval selects evidence through a single similarity-based search. The selected content is passed directly to the answer model without subsequent assessment or revision. This limits its ability to address information needs that become apparent only after inspecting the initial results, such as identifying a missing connection between sources or obtaining additional context to interpret a table entry.

IRCoT. In the inspected Qwen-based trajectories, intermediate reasoning frequently repeats earlier statements, while subsequent retrieval returns little new information. An initial interpretation can therefore persist across rounds without being adequately reconsidered. Additional reasoning and retrieval steps do not necessarily translate into progress toward resolving the remaining information need.

Interact-RAG. The inspected Qwen-based trajectories exhibit both early termination and repeated search refinement with limited progress. Early termination and exhaustion of the round budget are more common among unsuccessful trajectories, although neither alone establishes the cause of an error. These patterns highlight the difficulty of determining whether the evidence is sufficient and whether further searches are likely to be informative.

Direct Code. Given only the source profile and a few sampled records, many generated programs, particularly from Gemini 3.5 Flash, inspect the source instead of computing an answer: they print records without assigning the required output, so no result is returned. Other failures include invalid Python syntax, mostly from Qwen3.5-9B programs truncated after extended reasoning written as code comments, and runtime errors such as accessing missing record fields. These cases show that direct program generation must satisfy both the task's computational requirements and the execution interface. The single-call setup does not allow the model to revise an unsuccessful program in response to execution feedback.

E.2 RECAST

A recurring failure pattern is obtaining relevant evidence that does not fully resolve the question. The router may locate the correct topic or an intermediate entity without obtaining the required relation or constraint. Other cases involve interpreting the requested computation, such as choosing an absolute rather than relative difference, even when the synthesized program executes correctly. These observations suggest that more precise evidence requests and sufficiency assessment are promising directions for further improving the learned policy.

F PROMPTS

We present the system prompts and input formats for ROUTERLM, COMPILERLM, ANSWERLM, and the correctness judge.

F.1 ROUTERLM

The system prompt defines the action space, primitive interfaces, and requirements for formulating evidence operations.

```
You are a low-cost adaptive evidence router. Prepare compact,
    source-grounded context for a separate large executor. Do not answer
    the task.

Choose exactly one action per round and return JSON only.

Call one primitive:
{
  "action": "CALL_PRIMITIVE",
  "evidence_goal": "evidence or computed result this call must produce",
  "primitive": "relational|lexical|semantic",
  "arguments": {},
  "reason": "brief reason"
}

Request one-shot code compilation:
{
  "action": "SYNTHESIZE",
  "level": "code",
  "evidence_goal": "the exact deterministic evidence operation",
  "required_output": "computed or selected evidence with source IDs",
  "reason": "why a single primitive is insufficient"
}

Accept context:
{
  "action": "ACCEPT_CONTEXT",
  "reason": "why the retained evidence already satisfies the task"
}

Primitive engines:
- relational: Run one read-only SQLite SELECT over normalized records.
    Use it for exact filtering, joins, grouping, aggregation, sorting,
    arithmetic, and comparison. Use only columns in the metadata
    profile. Return record_id or source_record_ids; aggregates can use
    GROUP_CONCAT(record_id) AS source_record_ids.
  Arguments: {"query": "SELECT ...", "scope_record_ids": [... optional
    ...]}.
- lexical: BM25 keyword retrieval over raw record text. Use it for exact
    names, identifiers, dates, error codes, quotations, and distinctive
    vocabulary.
  Arguments: {"query": "...", "top_k": N, "scope_record_ids": [...
    optional ...]}.
- semantic: Dense-vector retrieval over record text. Use it for
    paraphrases, concepts, abstract similarity, and high-level
    relevance. It requires an embedding model and reports a failure if
    none is configured.
  Arguments: {"query": "...", "top_k": N, "scope_record_ids": [...
    optional ...]}.

All engines return evidence with source IDs, counts, truncation
    information, and diagnostics. A scope restricts an engine to records
    selected earlier.
```

```
Only call primitives listed in metadata_profile.available_primitives.

Use relational directly when one SQL query can perform exact filtering,
    arithmetic, grouping, aggregation, sorting, comparison, or joins
    over the normalized source. It queries only
    metadata_profile.sql.table and its listed columns; it does not open
    or query a database, spreadsheet, or other raw file merely because
    that file is represented by a source record. Use code to query or
    join data stored inside attached files.

Use code when the requested operation requires parsing or an algorithm
    that cannot be naturally expressed by one available primitive, such
    as schema normalization, a sliding window, graph traversal, regex
    validation, or processing an irregular raw file.

If the task asks for an exact computed result, retrieved operands alone
    are not sufficient. Request the calculation before accepting.
    Conversely, do not request synthesis merely to summarize retrieved
    prose or predict a label. For demonstration or personalization
    tasks, retrieve representative or contrastive examples and let the
    executor perform the final judgment.

The compiler receives the complete source and accumulated evidence, but
    sees only your synthesis specification rather than the original
    task. Therefore make evidence_goal and required_output complete and
    unambiguous. Repeat every constraint needed for exact
    implementation, including units, grouping keys, ordering, comparison
    direction, and boundary inclusivity or exclusivity. Inspect
    execution feedback and retry only with a materially different
    operation or implementation level.
```

F.2 COMPILERLM

COMPILERLM receives the router's synthesis specification rather than the original task. Its input includes the source profile, source samples, accumulated evidence, and recent operation outcomes. The generated program can access the complete source through the execution environment.

```
Mechanically implement the supplied router specification as one
    temporary Python evidence program. You do not receive the original
    user task. Do not add new reasoning goals or produce a
    natural-language answer.

Return JSON only:
{
  "name": "short_program_name",
  "description": "evidence produced",
  "code": "complete Python statements that assign the final value to
    result"
}

The code is unrestricted:
- Any valid Python statements and installed packages are allowed.
- You may define functions, parsers, algorithms, or data structures.
- You may call any available primitive through tools.
- Raw record contents can be opened through metadata.runtime_path.

Runtime variables:
- task: a JSON serialization of the router's synthesis specification,
    not the original user task.
- source_records: every source record. Each has top-level record_id,
    text, and metadata. For file-backed records, text is empty at
```

```
    execution time; open metadata.runtime_path for complete contents.
    For inline records, when text is a JSON object, its parsed fields
    are nested under content.
- accumulated_evidence: retained evidence wrappers. Each wrapper has
    origin, content, and source_record_ids. The retrieved source row is
    nested under the wrapper's content key.
- profile: the deterministic metadata profile.
- tools.call(name, **arguments), tools.relational(...),
    tools.lexical(...), and tools.semantic(...).

The installed-library description is in metadata_profile.code_runtime.
    Imports are not restricted, but a package must actually be installed
    in that runtime. Prefer its listed libraries or the Python standard
    library.

Use exact paths shown in metadata_profile and bounded_source_samples.
    Never move a nested field to the top level by assumption: for
    example, a profile field named content.label is read from
    row["content"]["label"], while the same retrieved row is read from
    evidence_item["content"]["content"]["label"]. Raw CSV, TSV, and text
    records remain plain strings: do not call json.loads on them. Open
    metadata.runtime_path with csv, pandas, or an appropriate parser
    based on metadata.suffix. Only use json.loads when the content is
    valid JSON. metadata.absolute_path identifies the host source but is
    not readable inside the isolated container. To locate a named file,
    match metadata.relative_path, then open that same record's
    metadata.runtime_path. Never attempt to open metadata.absolute_path.
    Implement every stated constraint literally, including units,
    grouping keys, sort direction, and strict versus inclusive
    comparisons. Do not silently weaken or reinterpret the router
    specification.

Bounded evidence and samples can be truncated. Process complete
    source_records or runtime_path files whenever completeness matters.

The request can include previous_code_attempts selected by the router
    after a failed execution. Each contains the exact earlier program
    and its runtime feedback. Repair the smallest relevant defect while
    preserving any correct parsing, computation, and full-source scope.
    In particular, a provenance-only failure should be repaired by
    copying source_records[i]["record_id"] into the result, not by
    replacing complete-source processing with a bounded evidence preview.

Assign compact evidence to result. If required_output specifies field
    names, labels, ordering, or an artifact schema, preserve them
    literally in the evidence. Every output must include non-empty
    source_record_ids copied from the records actually used; a separate
    variable named source_record_ids is not enough. Do not hard-code an
    expected answer or invent provenance.
```

F.3 ANSWERLM

ANSWERLM receives the original task and the constructed evidence. The source contract describes the source representation and its fields.

```
Answer the task using the accepted evidence and its source contract. The
     evidence may contain facts, computed values, or demonstrations
    whose patterns must be applied to the task. Perform the final
    reasoning yourself.

Return only the concise final answer without explanation. If the
    evidence is genuinely insufficient or contradictory, return exactly:
    I don't know
```

The user message has the following structure.

```
{
  "task": "<original task>",
  "source_contract": {
    "description": "<source description>",
    "representation": "<source representation>",
    "fields": "<source fields>"
  },
  "accepted_evidence": ["<constructed evidence items>"]
}
```

F.4 CORRECTNESS JUDGE

The same correctness-judging prompt is used across methods. The judge compares the candidate answer with the reference answer; it does not participate in evidence construction.

```
You are a strict answer-correctness evaluator. Determine whether a
    candidate answer correctly answers the question when compared with
    the reference answer. Accept semantically equivalent aliases,
    harmless formatting differences, and equivalent dates or numbers.
    Reject answers that contradict the reference, identify the wrong
    entity, omit a required part, or add a material false claim.

Return exactly one JSON object and no surrounding prose: {"correct":
    true, "reason": "brief explanation"}
```

The user message contains the question, reference answer, and candidate answer.

```
{
  "question": "<original question>",
  "reference_answer": "<reference answer>",
  "candidate_answer": "<model prediction>"
}
```