RASER: Recoverability-Aware Selective Escalation Router for Multi-Hop Question Answering

Anonymous ACL submission

Abstract

Multi-hop question-answering systems often use expensive retrieval on every question. They may decompose the question, run several retrieval rounds, or search through bridge entities before answering. All of these strategies rely on repeated LLM calls to rewrite or decompose the question, which increases extra token cost, and it is not fitting when the LLM budget is tight. However, our analysis shows that lots of multi-hop questions are already answered correctly by a single one-shot RAG, so running an extra retrieval on every question wastes the budget. We introduce **RASER** (**R**ecoverability-**A**ware **S**elective **E**scalation **R**outer), a family of cheap routers built on one-shot RAG and six features from it. RASER-2 decides whether to stop or escalate to the extra-retrieval action PRUNE. RASER-3 chooses among one-shot RAG, PRUNE, and iterative retrieval IRCoT, using the same features but adding an explicit cost-accuracy trade-off. Neither router makes an extra LLM call to decide. Across six LLMs and three multi-hop QA benchmarks, both routers stay competitive with the other state-of-the-art (SOTA) baselines in F_1 while spending only 41–49% of always-prune's tokens and also less than the iterative and decomposition retrieval baselines. Code is available at https://anonymous.4open.science/r/RASER-2026.

1 Introduction

Multi-hop question answering (QA) requires a system to integrate evidence, which are retrieved passages that support the answer from multiple texts, to produce an answer. A passage is a text chunk returned by the retriever. For example, answering a question such as "Who is the spouse of the director of *Inception*?" requires first identifying the director of *Inception* and then retrieving evidence about that person's spouse. This makes multi-hop QA different from single-hop QA: the difficulty is not only finding a relevant passage but also deciding which intermediate entity or fact should guide the next retrieval step. Benchmarks such as HotpotQA, 2WikiMultiHopQA, and MuSiQue were introduced to evaluate this kind of multi-step reasoning, with MuSiQue in particular designed to reduce shortcut solutions and require connected reasoning (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022).

In a retrieval-augmented generation (RAG) system, the answer is produced by a large language model (LLM) after retrieving passages. And a common solution to this challenge is *multi-step retrieval*: instead of retrieving once and answering immediately, the system repeatedly expands the retrieved passages. These methods are effective because they can recover evidence that a one-shot retriever misses. However, they also apply an extra retrieval for every question: every question pays the cost of extra retrieval, extra LLM calls, or query rewriting, even when the one-shot answer is already correct or when no tested retrieval strategy can recover the answer.

This paper asks: *When is additional retrieval actually necessary?* To answer it, we perform a recoverability analysis over multiple retrieval strategies. For each question, we run a one-shot dense retrieval path and several bridge-style retrieval paths and then compare their answer F_1 after the fact. This analysis measures the potential benefit if a router always chose the best strategy for each question. The analysis reveals only a minority of questions improve by at least 0.1 F1 from bridge-style retrieval, while many are already solved by one-shot retrieval, and another 14–27% remains unrecoverable under all tested strategies. This suggests that the main challenge in multi-hop QA is not simply how to retrieve more evidence but how to decide *when* bridge retrieval is worth paying for.

We therefore define multi-hop QA as a **recoverability-aware selective escalation** problem. Rather than always applying iterative retrieval, our method first produces a one-shot answer from

dense retrieval, then predicts whether that answer is sufficiently supported by the retrieved evidence or whether the question should be escalated to a bridged retrieval path. This framing is related to prior adaptive retrieval and routing work (Jeong et al., 2024; Guo et al., 2026; Liu et al., 2026), but differs in two important respects. First, the decision target is *recoverability*: will bridge retrieval help this question under the current retrieved passages? Second, the router is implemented as **a lightweight Gradient Boosting Machine (GBM)-based classifier** over cheap answer-side, retrieval-side, and evidence-support features, introducing zero router-side LLM tokens.

Our method, **RASER** (**R**ecoverability-**A**ware **S**elective **E**scalation **R**outer), is a family of routers built on the same one-shot RAG and six features. The first one-shot RAG returns the first draft answer and top-k chunks, and then it produces the six features: retrieval similarity scores, draft-answer length, question type, and bridge gaps. RASER-2 is a 2-action router; it is a gradient-boosted binary classifier over six features to pick between one-shot RAG and an extra retrieval route, "PRUNE," under a fixed threshold. RASER-3 is a 3-action cost-aware router. It uses the same six features to predict the answer quality of one-shot RAG, PRUNE, and an iterative retrieval route, "IRCoT," and then chooses the best route based on a token-cost penalty. Both routers make their decision without an extra LLM call. Under a unified setup with the same retriever, LLMs, corpus, and chunking text across all baselines, RASER achieves *competitive F1 at a substantially lower token cost* than strong iterative baselines. Our paper has the following contributions:

- We formulate multi-hop QA as a **recoverability-aware selective escalation** problem and show through recoverability analysis that only a minority of questions improve by at least 0.1 F1 from additional retrieval.
- We propose **RASER**, a family of lightweight routers that includes RASER-2 and RASER-3. They use zero router-side LLM tokens and cheap evidence-support features.
- We provide a **unified comparison** among all the baselines and a route-wise analysis showing that RASER attains competitive F_1 at much lower token cost overall.

2 Related Work

Multi-hop QA Benchmarks Multi-hop QA datasets move beyond single-evidence question answering by requiring systems to connect entities, facts, or supporting passages. HotpotQA (Yang et al., 2018) introduced diverse, explainable multi-hop QA with supporting facts and remains the most widely used benchmark in this area. 2WikiMultiHopQA (Ho et al., 2020) was designed to ensure multi-hop reasoning through structured generation over Wikidata and Wikipedia, while also exposing reasoning paths and supporting evidence. MuSiQue (Trivedi et al., 2022) was proposed specifically to reduce shortcut solutions by composing connected single-hop questions, making it a stronger testbed for genuine multi-hop reasoning.

Extra Retrieval for Multi-hop QA A large part of previous work improves multi-hop QA by adding retrieval or reasoning steps. IRCoT interleaves chain-of-thought reasoning with retrieval queries (Trivedi et al., 2023), and ITER-RetGen alternates between retrieval and generation in the document (Shao et al., 2023). Decomposition approaches such as Self-Ask, Least-to-Most Prompting, and Decomposed Prompting make intermediate questions explicit(Press et al., 2023; Zhou et al., 2023; Khot et al., 2023). In the recent work, Chain-RAG (Zhu et al., 2025) mitigates lost-in-retrieval failures through progressive retrieval, sub-question answering, and query rewriting. KiRAG (Fang et al., 2025) instead performs knowledge-driven iterative retrieval by decomposing documents into triples and dynamically retrieving bridge knowledge that fills information gaps. Both methods show that one-shot retrieval is often insufficient for multi-hop reasoning.

Adaptive Retrieval Several systems have already avoided using the same retrieval method for every input. Adaptive-RAG (Jeong et al., 2024) sees retrieval as a complexity-based decision and learns to select among no-retrieval, single-step retrieval, and iterative retrieval strategies. DeepSieve (Guo et al., 2026) uses an LLM as a knowledge router across heterogeneous knowledge sources; FLARE retrieves when a forward-looking generation appears uncertain (Jiang et al., 2023), and DRAGIN triggers retrieval when token-level signals suggest the model needs outside information (Su et al., 2024).

Reader	one-shot RAG correct	no method helps	small / unclear gain	bridge helps
GPT-OSS-120B	**53%**	14%	19%	14%
Gemma-3-31B	**48%**	19%	13%	20%
Mistral-S-119B	37%	20%	9%	**34%**
Phi-4-mini	31%	25%	15%	29%
Llama-3.1-8B	27%	27%	8%	**38%**
Llama-3-8B	**24%**	26%	12%	**37%**

Table 1: What happens to each question if we try deeper retrieval? We assign a question by the best result among three expensive bridge methods: PRUNE, iterative IRCoT*, and decomposition SELF-ASK*. The first two columns are questions that should *not* be escalated: either the one-shot answer is already correct, or no methods fix it. The last column is a clear target for bridging retrieval: questions where *some* expensive bridges improve the answer by ≥ 0.1 F_1 more than a one-shot retrieval.

Selective prediction and cost control The closest idea is selective prediction, where a model may abstain from examples likely to be wrong (Geifman and El-Yaniv, 2017). The system still answers every question; it only abstains from paying for a more expensive retrieval action when that action is unlikely to change the answer. With a 3-action router, the same idea becomes action selection: if paying more is worthy, the system must also decide whether a bridge or an iterative action is the better use of the budget.

3 Recoverability Analysis

Before introducing RASER, we first ask a diagnostic question: *how many multi-hop questions actually need additional retrieval?* Although multi-hop QA benchmarks are designed to require reasoning over multiple pieces of evidence, not every question necessarily benefits from running an expensive iterative retrieval pipeline. Some questions are already answered correctly by one-shot retrieval, while others remain difficult even after additional retrieval. To quantify this, we perform a per-sample oracle analysis over three representative strategies: ONE-SHOT RAG, Prune, IRCoT, and SELF-ASK.

For each question, we run all strategies and compute the answer F1 produced by each one. It tells us how the dataset decomposes into questions that are already solved, questions that can be helped by bridge retrieval, and questions that none of the tested methods can recover.

Table 1 shows that lots of questions do not clearly benefit from extra retrieval. This recoverability analysis reframes the task: the central problem is not *how to always retrieve more*, but *how to detect when bridge retrieval is worth paying for*.

4 Approach

Section 3 shows that extra retrieval is useful only for some questions and differs by LLMs and datasets. But running extra retrieval on every question wastes budget, because lots of questions do not need it. What we want instead is a way to spend the expensive budget where it pays off. This section describes two routers. Our method, **RASER** (**R**ecoverability-**A**ware **S**elective **E**scalation **R**outer), is not a new retriever or a new LLM. It is a small decision layer after the first one-shot retrieval-and-answer. It is used to answer one practical question: Return the one-shot retrieved answer, or spend more tokens on one of the extra retrieval routes and answer again? The 2-action RASER is the low-budget router; it decides whether to stop or run one bridge step. The 3-action RASER is the cost-aware router: it decides whether to stop, bridge once, or run iterative retrieval. Figure 1 shows one example question through the whole pipeline; the rest of this section will explain each part.

4.1 2-action RASER: two-route recoverability classifier

A 2-action RASER chooses between two actions. The cheap action, one-shot RAG, just retrieves once and asks the LLM for an answer. The expensive action, PRUNE, tries to find a bridge entity, which is the missing entity that links the question's two hops. For example, to answer, "Who is the spouse of *Young Man Luther*'s author?" You first have to identify the bridge entity, *Erik Erikson* (the author of the book). Then you can look up his spouse with this bridge entity. PRUNE has four steps: ask the LLM to propose up to two candidate bridge entities, re-retrieve using each one, drop weak entities with a lightweight rule-based verifier, and ask the reader for a final answer.

2-action RASER runs four steps: 1. Run one-shot RAG. Retrieve the top-k chunks once and ask the LLM for a draft answer. 2. Summarize the first RAG results. Build a six-feature vector from the retrieved scores, the draft answer, and the question text. 3. Predict whether extra retrieval will help. A GBM (Gradient Boosting Machine) classifier estimates p(BRIDGEABLE | s), the probability

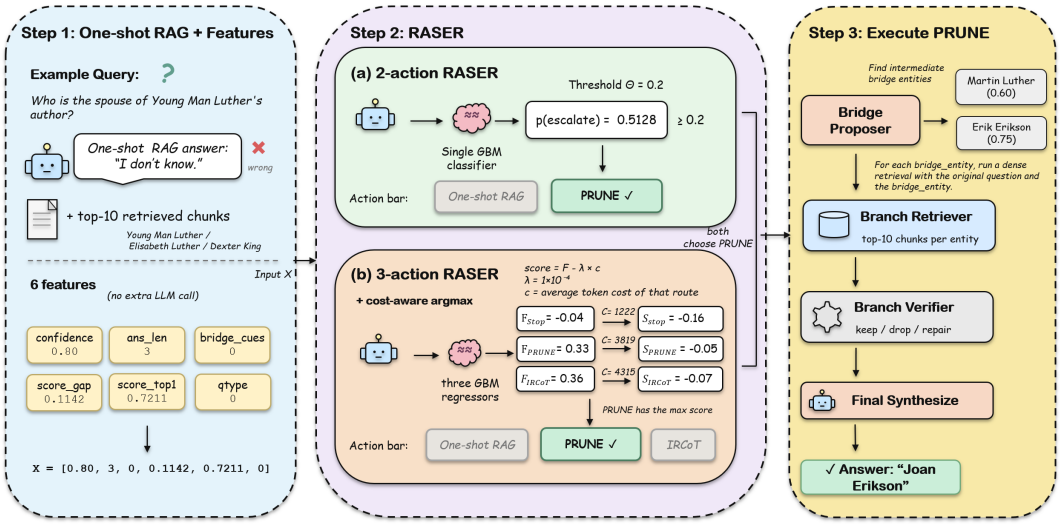

Figure 1: **RASER workflow.** The example question needs two reasoning hops to answer (the author of *Young Man Luther* is Erik Erikson, whose spouse is Joan Erikson), and the one-shot RAG gets the wrong answer (*"I don't know"*). It also produces six features. RASER reads those same six features to decide which action to choose. The **2-action RASER** (a) is a single GBM classifier: it picks between stopping directly after one-shot RAG and PRUNE by checking bridge entities and doing extra retrieval. The **3-action RASER** (b) is three predictors plus a token-cost penalty: it picks among one-shot RAG, PRUNE, and IRCoT* by weighing predicted accuracy against the tokens each action would spend. For this question, both routers choose PRUNE. Step 3 shows the Bridge Proposer proposing two bridge candidates (*Martin Luther* and *Erik Erikson*); the Branch Retriever re-retrieves chunks for each, the Branch Verifier keeps the useful ones, and the Final Synthesizer combines the bridges with the chunks to return the correct answer *Joan Erikson*. Prompts are in Appendix B.1.

that a PRUNE action will improve the draft answer. 4. Choose an action. If the probability is at least Threshold θ, run the bridge action PRUNE and return its answer; otherwise, return the draft answer from one-shot RAG. Appendix G shows why we select $\theta = 0.20$. RASER is trained by six features (Table 2). Two features describe the draft answer, two features describe the retrieval scores, one feature looks for bridge-like words in the question, and one feature records the question type. All of them are available after one-shot RAG, so adding the router does not require any language model calls or retrieval methods.

For the *Young Man Luther* question above (Figure 1), one-shot RAG answers *"I don't know"*, which is not the gold answer *Joan Erikson*. The first-shot RAG also gives the router useful signals: the answer is short and an "I Dont know"(`ans_len` = 3, `confidence` = 0); the top retrieved chunks are about Luther family members but not the name of the author of the book (`score_gap` = 0.1142, no chunk clearly wins); and the question asks for a person (`qtype` = entity). From these, the classifier assigns $p(\text{BRIDGEABLE}) = 0.5128$, above the threshold $\theta = 0.20$, so RASER-2 runs PRUNE. The Bridge Proposer extracts *Erik Erikson* as the missing fact, the Branch Retriever pulls Erikson-related chunks, and the Final Synthesizer returns *Joan Erikson*. The details of the prompts are shown in the Appendix B.1.

4.2 3-action RASER: three-route cost-aware classifier

The 2-action RASER answers only, "Should we escalate?" That works as long as there is one obvious expensive action to escalate. But there isn't. Which expensive route helps change with the LLM? PRUNE wins on some of them and IRCOT* on others. A 3-action router handles that by extending the routes and adding a price tag. The routes grow to three choices: one-shot RAG, PRUNE, and the iterative route IRCOT*—and the router learns with each question *whether* to escalate and *which* expensive route is worth it.

The iterative route. IRCOT* extracts knowledge triples from the retrieval results, asks what information is still missing, then it retrieves again, and repeats for up to three iterations (around 5–7 LLM calls, 3,500–5,500 tokens). It follows the IRCoT (Trivedi et al., 2023) and is the most expensive

Feature	Definition
`confidence`	Rule-based confidence score from one-shot RAG and the question type.
`ans_len`	Number of words in the draft answer.
`bridge_cues`	Did the question text match any multi-hop linguistic pattern? e.g., *"X of the Y of Z"*, *"who ...was born/lived/located in ...where/which/what,"* or *"the country where"*.
`score_gap`	Gap between the top-1 and top-5 dense retrieval scores. Big gap = clear winner; tiny gap = ambiguous.
`score_top1`	Top-1 dense retrieval score. Cosine similarity between the question and the top-1 retrieved chunk
`qtype`	Question type (entity = 0 / date = 1 / yes-no = 2 / count = 3 / other = 4), assigned by simple rules.

Table 2: The six input features to the RASER. They come after the one-shot RAG: the draft answer, the top-10 retrieved chunks' similarity scores, and the question itself. Computing them doesn't require extra LLM calls.

one. The exact prompts are in the Appendix B.1.

Three score predictors. A yes/no classifier cannot express "Route B is slightly better than A, but Route C is even better," so with three routes, we change the target. We train three small *score predictors*, one predictor for each route. Each one looks at the same six first-pass features as a 2-action RASER and answers a what-if question: if we sent this question to route r, what F1 would it get? We write these predictions: "$\hat{f}_{\text{ONE-SHOT RAG}}(\mathbf{s})$, $\hat{f}_{\text{PRUNE}}(\mathbf{s})$, and $\hat{f}^*_{\text{IRCoT}}(\mathbf{s})$. "The model class and training protocol are described together with the 2-action RASER in Section 5.

Inference (cost-aware argmax). Once we can estimate each route's F1, it becomes easy to pick a route: for each route, take its predicted F1 and subtract a penalty for how many tokens it spends, then choose the route with the best score:

$$r^* = \arg\max_{r \in \mathcal{R}} \left[\hat{f}_r(\mathbf{s}) - \lambda \bar{c}_r \right], \quad \mathcal{R} = \{\text{ONE-SHOT RAG}, \text{PRUNE}, \text{IRCoT}^*\}. \tag{1}$$

The λ is an *exchange rate* between tokens and accuracy about how much predicted F_1 an extra token has to "buy" before the router is willing to spend it. At $\lambda = 0$, tokens are free, so the router just takes the highest predicted F_1; as λ grows, an expensive route must promise an even higher F_1 to be chosen, and at a large enough λ the router always stops. Turning this λ from low to high traces out the cost-accuracy curve in Figure 2: rather than providing a single fixed system, we give the operator choices and let them pick the token budget they are willing to pay.

On the same *Young Man Luther* question used in Figure 1, the three predictors output $\hat{f}_{\text{ONE-SHOT RAG}} = -0.04$, $\hat{f}_{\text{PRUNE}} = 0.33$, and $\hat{f}^*_{\text{IRCoT}} = 0.36$. IRCoT* would win slightly on raw F_1. But IRCoT* also costs about 500 more tokens than PRUNE, so after the cost penalty $\lambda \bar{c}_r$ (with $\lambda = 10^{-4}$ and route costs $\bar{c}_{\text{ONE-SHOT RAG}} = 1{,}222$, $\bar{c}_{\text{PRUNE}} = 3{,}819$, $\bar{c}^*_{\text{IRCoT}} = 4{,}315$) the scores become -0.16, -0.05, and -0.07: PRUNE wins and RASER-3 picks the same route as RASER-2. IRCoT*'s 0.03 extra predicted F_1 was not worth its extra tokens. ($\bar{c}_r$ and λ are computed from training-fold data only; details in Section 5.) An analysis varying the cost budget from 0.33 to 1.00 is in Appendix G.

A second example, where the IRCoT finds the results, is shown in Figure 3 (Appendix B). On the Achaemenid question (Llama-3.1-8B / MuSiQue), RASER-2's escalation probability is 0.164, just below the threshold $\theta = 0.20$, so it keeps the one-shot *"I don't know"*. RASER-3's regressors predict $\hat{f}^*_{\text{IRCoT}} = 0.50$, much higher than ONE-SHOT RAG or PRUNE, and after the cost penalty IRCoT* still wins; two retrieve-extract rounds find the answer *323 BC* (F1= 1). This is the case where RASER-2's threshold gates the question out and RASER-3's cost-aware argmax keeps IRCoT* available. Prompts are in Appendix B.1.

RASER-2 and RASER-3 share the same features and the same one-shot RAG results; they differ in classifier (single classifier vs. three regressors) and decision rule (threshold vs. cost-aware argmax). Both are the main systems in this paper.

5 Experimental Setup

Benchmarks We evaluate three multi-hop QA benchmarks with complementary properties. **MuSiQue** (Trivedi et al., 2022) is our main benchmark because it was explicitly designed to reduce shortcut solutions and enforce connected reasoning. **2WikiMultiHopQA** (Ho et al., 2020) serves as a second benchmark with explicit reasoning paths and supporting evidence. **HotpotQA** (Yang et al., 2018) is used as a control benchmark: many questions can already be handled well by one-shot retrieval, making it useful for testing whether a routing method remains restrained when bridge

retrieval is less consistently necessary.

Retrieval All methods use the same dense retriever: Nomic-Embed-Text-v1.5 with cosine similarity (Nussbaum et al., 2025). Passages are sentence-segmented and chunked at 128 tokens with 16-token overlap.

LLMs We evaluate six LLMs: GPT-OSS-120B (OpenAI, 2025), Mistral-Small-119B (Mistral AI, 2026), Gemma-3-31B-it (Team et al., 2025), Llama-3.1-8B-Instruct, Llama-3-8B-Instruct (Grattafiori et al., 2024), and Phi-4-mini-instruct (Microsoft et al., 2025). All of them are queried through an OpenAI-compatible chat API at temperature 0.

GBM and hyperparameters Both RASER routers are Gradient Boosting Machines (GBM) as implemented in scikit-learn (Pedregosa et al., 2018), with the same hyperparameters: 100 trees of max depth 3, learning rate 0.1, and subsample 0.8. RASER-2 is a single Gradient Boosting Classifier trained with log loss; RASER-3 is three Gradient Boosting Regressors, one for each action, each trained with mean squared error (MSE).

Training label For RASER-2, we label a question BRIDGEABLE ($y = 1$) only when the bridge route PRUNE improves the one-shot RAG answer by more than $\tau = 0.1$ F_1:

$$y = \mathbb{I}\left[F_{1\,\text{PRUNE}} - F_{1\,\text{ONE-SHOT RAG}} > \tau\right]. \quad (2)$$

If ONE-SHOT RAG is already correct or no route helps, $y = 0$; $y = 1$ only when the PRUNE route repairs a wrong first-RAG answer. For RASER-3, each regressor $\hat{f}_r$ is trained to predict the per-question F_1 that route r actually achieved on the training data. More details about how we trained RASER are provided in the appendix.

6 Results

Table 3 summarizes the main result: fixed expensive retrieval policies often improve answer F_1, but they pay that cost on every question. RASER keeps high accuracy while avoiding many unnecessary, expensive LLM calls. We report the answer F_1 and the average LLM tokens for each benchmark. The visualization of this table is Figure 6, shown in the Appendix D.

Fixed expensive baselines improve some questions, but they pay that cost for every question. Always running PRUNE spends roughly $3\times$ the tokens of one-shot RAG, while its aggregate F_1 gains are modest and uneven. RASER-2 targets exactly this regime: across six readers and three benchmarks, it stays within 0.026 F_1 of always-PRUNE while using only 39–57% of its tokens. This shows that a cheap recoverability decision can preserve most of the bridge benefit while removing much of the bridge cost. RASER-3 adds an iterative route for settings where one-round bridging is not enough. Its value is LLM-dependent. Frontier LLMs IRCoT* often add little beyond cheaper routes, and RASER-3 stays close to RASER-2. Mid-tier open LLMs, especially MuSiQue and 2Wiki, IRCoT* has real marginal value; there RASER-3 captures much of the always-IRCoT* gain while spending far fewer tokens. Thus RASER-3 should not be read as a universal replacement for RASER-2, but as a higher-budget operating point when iterative retrieval is worth paying for.

Finally, we check whether this conclusion depends on using a lightweight decomposition primitive. Replacing SELF-ASK* with a controlled ChainRAG reimplementation under the same retriever, reader, and corpus changes GPT-OSS-120B F_1 by at most 0.020 per dataset at a similar token cost and does not change the method ranking. The result supports the same conclusion: RASER's gain comes from selective routing, not from inventing a stronger, more expensive route.

6.1 Why does iterative retrieval help small-size LLMs more?

The LLMs comparison in Table 3 shows that larger-sized LLMs gain little from IRCoT*, while middle- and small-sized LLMs benefit from it. One explanation is that larger-sized LLMs rely less on retrieved evidence. They may answer some questions from their memory even when the retrieved passage changes. Because these LLMs are trained with that information.

We test this on 200 MuSiQue questions. For each question, we edit the retrieved passages that contain the answer, we replace the correct answer with a plausible incorrect one, and then we run ONE-SHOT RAG again. If the LLM outputs the new, incorrect answer, it follows the retrieved evidence. If it keeps the original gold answer, it is likely relying on internal memory.

The results show GPT-OSS-120B keeps the original answer on 38% of edited questions, while Llama-3-8B only 9%. This explains why larger LLMs gain less from iterative retrieval; many of

	ONE-SHOT RAG		a-PRUNE		IRCoT*		SELF-ASK*		ChainRAG†		**RASER-2**		RASER-3	
Reader / dataset	F_1	tk	F_1	tk	F_1	tk	F_1	tk	F_1	tk	F_1	tk	F_1	tk
GPT-OSS-120B (N=500 per dataset)														
MuSiQue	0.488	1.5k	0.528	4.8k	0.455	4.3k	0.482	4.6k	0.465	4.6k	**0.502**	**2.1k**	**0.510**	**2.2k**
2Wiki	0.754	1.1k	0.761	3.3k	0.781	3.0k	0.743	3.5k	0.763	3.6k	**0.763**	**1.3k**	**0.774**	**1.6k**
HotpotQA	0.777	1.4k	0.769	3.7k	0.763	3.6k	0.724	4.1k	0.727	3.6k	**0.779**	**1.5k**	**0.787**	**1.8k**
Mistral-S-119B (N=300/300/200)														
MuSiQue	0.283	1.2k	0.397	3.8k	0.499	4.3k	0.380	3.3k	0.464	2.3k	**0.306**	**1.9k**	**0.408**	**2.4k**
2Wiki	0.450	970	0.570	2.9k	0.658	2.9k	0.562	2.8k	0.643	1.7k	**0.485**	**1.5k**	**0.451**	**979**
HotpotQA	0.699	1.3k	0.716	3.4k	0.773	3.4k	0.638	3.3k	0.662	1.6k	**0.702**	**1.4k**	**0.723**	**2.2k**
Gemma-3-31B (N=300/300/200)														
MuSiQue	0.393	1.2k	0.503	3.7k	0.577	3.8k	0.360	3.0k	0.445	2.3k	**0.458**	**2.1k**	**0.502**	**2.4k**
2Wiki	0.630	962	0.694	2.8k	0.722	2.5k	0.664	2.5k	0.625	1.8k	**0.656**	**1.2k**	**0.700**	**1.6k**
HotpotQA	0.787	1.3k	0.797	3.4k	0.810	3.0k	0.734	2.8k	0.740	1.6k	**0.796**	**1.4k**	**0.812**	**1.8k**
Llama-3-8B (N=300/300/200)														
MuSiQue	0.223	1.2k	0.256	3.7k	0.354	4.7k	0.292	3.2k	0.209	2.8k	**0.229**	**1.6k**	**0.309**	**2.8k**
2Wiki	0.367	932	0.406	2.8k	0.507	3.5k	0.460	2.2k	0.377	2.1k	**0.380**	**1.2k**	**0.472**	**2.1k**
HotpotQA	0.632	1.2k	0.656	3.3k	0.688	4.1k	0.621	3.5k	0.474	2.5k	**0.635**	**1.3k**	**0.655**	**2.2k**
Llama-3.1-8B (N=300/300/200)														
MuSiQue	0.236	1.2k	0.264	3.8k	0.376	5.1k	0.276	3.2k	0.312	2.6k	**0.253**	**1.5k**	**0.330**	**3.0k**
2Wiki	0.307	954	0.387	2.9k	0.532	4.0k	0.353	2.3k	0.518	1.9k	**0.355**	**1.4k**	**0.487**	**2.2k**
HotpotQA	0.612	1.3k	0.626	3.4k	0.676	4.8k	0.625	3.6k	0.596	2.2k	**0.624**	**1.5k**	**0.654**	**2.4k**
Phi-4-mini (N=300/300/200)														
MuSiQue	0.331	1.1k	0.353	3.6k	0.372	5.4k	0.407	3.0k	0.293	1.9k	**0.335**	**1.4k**	**0.357**	**2.7k**
2Wiki	0.391	904	0.459	2.6k	0.532	3.8k	0.457	2.6k	0.475	1.6k	**0.402**	**1.1k**	**0.474**	**2.3k**
HotpotQA	0.634	1.2k	0.643	3.2k	0.670	4.7k	0.577	3.2k	0.564	1.6k	**0.635**	**1.4k**	**0.652**	**2.2k**

Table 3: Main results: answer F_1 and mean tokens per question (k $= 10^3$) for six LLMs and three datasets under the same settings. Held-out N is shown in each LLM: GPT-OSS-120B uses N=500 per dataset; the other readers use N=300/300/200 for MuSiQue/2Wiki/HotpotQA. The routers this paper contributes to; RASER-2 keeps most of the bridge accuracy at a much lower cost, and RASER-3 adds the iterative route, which has real improvements.

LLM	Edited	Gold	Abstain	Other
GPT-OSS-120B	41%	**38%**	12%	9%
Gemma-3-31B	52%	21%	18%	9%
Llama-3-8B	67%	9%	12%	12%

Table 4: Memory check on 200 MuSiQue questions. *Edited* means the reader follows the corrupted retrieved passage and outputs the edited wrong answer. *Gold* means the reader keeps the original gold answer instead of edited evidence, suggesting it relies on internal memory rather than retrieved evidence.

Feature	Importance	Plain meaning
`score_gap`	0.38	top-1 vs. top-5 score gap
`score_top1`	0.35	top-1 retrieved passage's score
`confidence`	0.12	rule-based confidence
`ans_len`	0.12	lengthen of the draft answer
`bridge_cues`	0.02	bridge-like words in the question
`qtype`	0.01	question type

Table 5: Feature importance for RASER.

their one-shot RAG successes are not true retrieval successes, so extra retrieval has less space to help. Smaller LLMs follow the retrieved text more often, making retrieval depth more important.

6.2 Features Ablation

Which input features mainly drive the RASER Table 5 shows that tree features are important in all LLMs. Higher importance means the router uses that feature more often when making routing decisions. The `score_gap` and `score_top1`, account for 0.73 of the total importance, showing that the router primarily relies on whether the first retrieval step produced clear and reliable evidence. One-shot RAG answer features are secondary: `confidence` and `ans_len` together contribute 0.24. By contrast, `bridge_cues` and `qtype` contribute only 0.03, suggesting that RASER learns to escalate when the retrieved evidence and draft answer jointly indicate that the one-shot RAG is unreliable.

6.3 Classifiers Ablation

We also tried other classifiers. We compare the two-route GBM with XGBoost, LightGBM, CatBoost, scaled logistic regression, and a small MLP. RASER might also work in other classifiers. We applied Gradient Boosting Machine because it is the simplest and most widely available default. Some

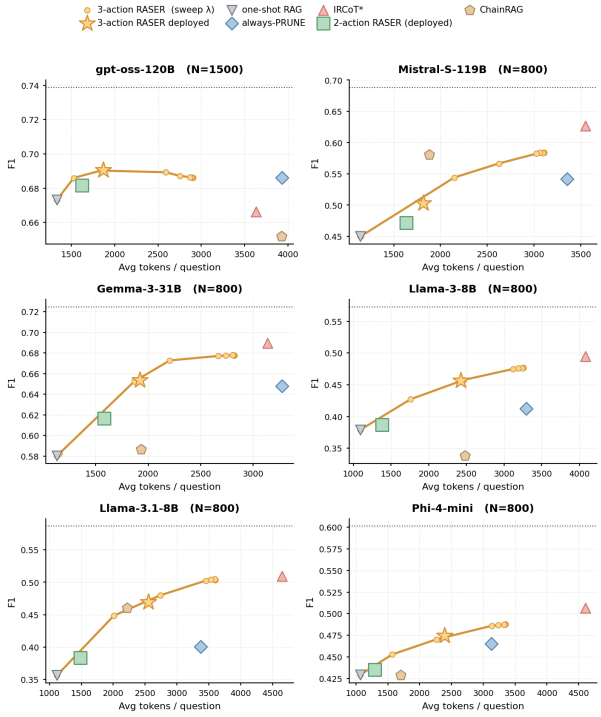

Figure 2: Cost-F_1 trade-off for each LLM across MuSiQue, 2WikiMultiHopQA, and HotpotQA (weighted by N for each dataset). The **orange curve** is RASER-3 as we sweep the cost penalty λ from low to high; each point is one λ setting. The **orange star** marks the deployed operating point (the RASER-3 column of Table 3). Other methods include one-shot RAG, always-PRUNE, IRCoT*, ChainRAG, and the deployed RASER-2. The *oracle*: the F_1 a perfect router would be obtained if it could always pick the best of the three routes for each question. On small LLM Llama-3-8B and Phi-4-mini, RASER-3 gets more F_1 for the same tokens.

results are shown in Table 6. Full classifier comparisons are reported in Appendix F.

Router	Model	F_1	Tokens	Route
RASER-2	GBM	0.520	1514	86/14
	LogReg	0.531	1549	85/15
	MLP	0.530	1597	82/18
RASER-3	GBM	0.562	2157	64/12/24
	XGBoost	0.562	2094	67/10/23
	Ridge	0.546	1844	76/5/18

Table 6: Part of Classifier ablation. GBM is not the best, but it is within the same F_1 range as the alternatives while being simple and reproducible.

7 Discussion

When Should We Use RASER-3? 3-action RASER is the cost-aware router. Its purpose is not to replace 2-action but to choose among one-shot RAG, PRUNE, and IRCOT* when the application is willing to spend more for accuracy. This matters when iterative retrieval is useful. On the middle-sized LLMs, IRCOT* improves F1 over one-shot RAG and bridging retrieval PRUNE (for example, Llama-3.1-8B on MuSiQue: IRCOT* 0.376 vs. PRUNE 0.264). In this case, 3-action RASER reaches 87–97% of always-IRCOT* F1 while spending only 47–62% of its tokens.

The RASER-3 results in Table 3 are the results of the specific λ we deployed. not RASER-3 itself. In our setup λ is the cost penalty and it is in the cost-aware argmax (Eq. 1); Figure 2 shows turning λ higher also applies more tokens. Small λ has higher F_1 with higher token cost; large λ performs worse than RASER-2. RASER-3 is not a fixed system at all, but it is a cost dial. It behaves based on your budget tokens. Appendix E checks both routers. RASER-2 escalates the questions where PRUNE actually helps. RASER-3 uses IRCoT* more often for the LLMs and datasets where IRCoT* actually gives an F_1 gain.

Comparison to other Baselines RASER is lightweight by design. The router reads six features already available after the one-shot RAG pass and makes no additional LLM calls before deciding whether to escalate. It also does not require pre-built knowledge graphs, pre-trained retrievers, or an LLM judge for routing. This distinguishes RASER from heavier iterative-retrieval systems. In our experiments, IRCoT* and SELF-ASK* are controlled methods based on literature, not full reproductions of KiRAG or ChainRAG. The contribution of RASER is not a new, expensive retrieval method; it is a recoverability-aware decision layer that decides when bridge retrieval is worthy.

8 Conclusion

Multi-hop QA is usually treated as a "retrieve more" problem, but our analysis shows that only a small portion of questions benefit from extra retrieval. Then RASER asks the research question, "When is extra retrieval worth it?" It is a small classifier that runs after one-shot RAG, reads six features, and decides whether to escalate to a bridge step (RASER-2) or to iterative retrieval (RASER-3), with no extra LLM calls. Across six LLMs and three benchmarks, RASER-2 and RASER-3 reach F_1 comparable to and sometimes higher than the strongest baselines, while spending far fewer tokens. It shows deciding *when* to retrieve more is cheaper and more useful than always retrieving more.

Limitations

Baselines are simplified versions We did not completely run the original KiRAG or ChainRAG, but a simplified version of them. Our IRCoT*, Self-Ask*, and ChainRAG baselines share the same retriever and LLM as RASER, so the comparison is fair in our setup. But the numbers should not be compared directly to the original papers' results. KiRAG, for example, has trained some heavy reasoning components that we did not reproduce.

Small evaluation size and reader memory We evaluate 200–500 questions for each LLM and dataset, so the per-cell F_1 numbers have some noise. The memory check in §6.1 also shows that large-sized LLMs sometimes answer from memory instead of the retrieved passage. So F_1 on those LLMs reflect both retrieval and the LLM's own knowledge, not retrieval alone. But RASER performs well on smaller-sized LLMs, which have less memory.

References

Jinyuan Fang, Zaiqiao Meng, and Craig MacDonald. 2025. KiRAG: Knowledge-driven iterative retriever for enhancing retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18969–18985, Vienna, Austria. Association for Computational Linguistics.

Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Minghao Guo, Qingcheng Zeng, Xujiang Zhao, Yanchi Liu, Wenchao Yu, Mengnan Du, Haifeng Chen, and Wei Cheng. 2026. DeepSieve: Information sieving via LLM-as-a-knowledge-router. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 3054–3077, Rabat, Morocco. Association for Computational Linguistics.

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

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. 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)*, pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. *Preprint*, arXiv:2210.02406.

Jiate Liu, Zebin Chen, Shaobo Qiao, Mingchen Ju, Danting Zhang, Bocheng Han, Shuyue Yu, Xin Shu, Jingling Wu, Dong Wen, Xin Cao, Guanfeng Liu, and Zhengyi Yang. 2026. A2rag: Adaptive agentic graph retrieval for cost-aware and reliable reasoning. *Preprint*, arXiv:2601.21162.

Microsoft, :, Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, and 1 others. 2025. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *Preprint*, arXiv:2503.01743.

Mistral AI. 2026. Introducing mistral small 4. Mistral AI Blog. Accessed: 2026-05-26.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. 2025. Nomic embed: Training a reproducible long context text embedder. *Preprint*, arXiv:2402.01613.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2018. Scikit-learn: Machine learning in python. *Preprint*, arXiv:1201.0490.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. 2024. DRAGIN: Dynamic retrieval augmented generation based on the real-time information needs of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12991–13013, Bangkok, Thailand. Association for Computational Linguistics.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, and 1 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. ♪ MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. 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)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

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

Rongzhi Zhu, Xiangyu Liu, Zequn Sun, Yiwei Wang, and Wei Hu. 2025. Mitigating lost-in-retrieval problems in retrieval augmented multi-hop question answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22362–22375, Vienna, Austria. Association for Computational Linguistics.

A Supplemental Material Statement

The anonymized code and scripts needed to reproduce the experiments are available at:`https://anonymous.4open.science/r/RASER-2026`. The repository contains the implementation of the RASER pipeline, retrieval methods and routing systems, prompts, and configuration files, which are used to reproduce the main experiments and RASER system.

B RASER workflow

We show RASER's full pipeline on three example questions on MuSiQue with Llama-3.1-8B, one question for each route: one example is about the one-shot RAG answer being already correct, one example is about extra bridge steps fixing the answer (PRUNE), and the final example is about only iterative retrieval being able to find the missing fact and generate the correct answers(IRCoT*). Figure 3 above visualizes the third example (the IRCoT case); the other two are discussed in the text only. All F1 and token costs come directly after the runs.

B.1 Workflow prompts

The prompts below are what we actually send to the LLM.

Step 1—One-shot RAG. After running the first time RAG, we get the top 10 retrieved texts from the dense retriever; then, we transfer this information as the following example prompts to the LLM:

Prompt: One-shot RAG

Answer the following question concisely based on the given context. If you cannot find the answer, reply "I don't know".
Context: *{retrieved chunks}*
Question: *{q}*
Answer (be concise, just the answer):

PRUNE– Bridge Proposer. After RASER, we ask LLM to extract up to two bridge candidates as JSON; the prompts are shown as follows:

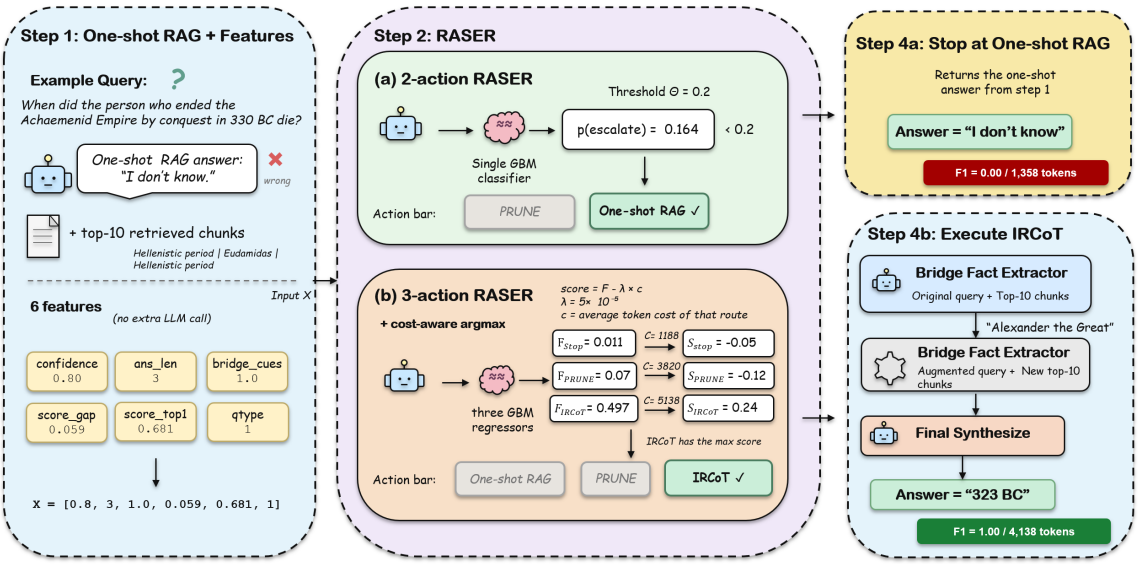

Figure 3: **RASER on a question where the two routers disagree.** Same pipeline as Figure 1, on the Achaemenid question (Llama-3.1-8B / MuSiQue). RASER-2's escalation probability 0.164 is below the threshold $\theta = 0.20$, so it returns the one-shot *"I don't know"* (Step 4a, F1= 0). RASER-3 picks IRCoT* via the cost-aware argmax ($\hat{f}^*_{\text{IRCoT}}$ stays highest even after the cost penalty), and two retrieve-extract rounds with the Bridge Fact Extractor find *323 BC* (Step 4b, F1= 1). Full step-by-step trace in Example 3 below.

Prompt: Bridge Proposer

You are decomposing a multi-hop question to find intermediate bridge entities. Identify up to 2 intermediate facts that could help answer the question.

For each, output a JSON object on its own line with these fields:

- `bridge_entity`: the intermediate fact ($\leq$ 5 words)
- `bridge_relation`: what role it plays
- `missing_slot`: what is still missing after this bridge
- `confidence`: 0.0–1.0

Passages: *{retrieved chunks}*
Question: *{q}*
Current answer: *{one-shot answer}*
Bridge proposals (one JSON per line):

PRUNE– Branch Verifier. Rule-based verifier, no LLM call. For each bridge, compute (i) *novelty* (fraction of new chunks vs. one-shot), (ii) *support* (fraction of new chunks mentioning the bridge), (iii) *info_gain* (1 – Jaccard overlap). A branch is dropped if novelty <0.05 or support <0.05.

PRUNE– Final Synthesizer. We merge the surviving chunks with the one-shot chunks and tell the LLM to bridge the fact that it can trust:

Prompt: Final Synthesizer

Answer the following question concisely based on the given context. If you cannot find the answer, reply "I don't know".

Intermediate facts already established by decomposition (treat as authoritative bridge entities):
{- bridge_1}
- {bridge_2}

Context: *{merged branch + one-shot chunks}*
Question: *{q}*
Answer (be concise, just the answer):

IRCoT*– iterative retrieval. Up to three rounds: extract a short fact from the current chunks, append it to the query, re-retrieve. The prompt shows below:

Prompt: IRCoT* fact extractor

You are decomposing a multi-hop question. Read the passages and Extract one intermediate fact that helps answer the question. The fact MUST be a short answer string (entity name, date, place — at most 5 words). NOT a sentence. NOT an explanation. NO reasoning.

Examples of valid replies: *Tracy McConnell / 1973 / Mississippi River / DONE*
Examples of INVALID replies: *"The answer is X because…"* / multi-line essays.

Passages: *{retrieved chunks}*
Facts already found: *{prior bridges}*
Question: *{q}*
Fact:

The iteration stops when the LLM says `DONE`, after

3 rounds, or when new chunks overlap the previous round's chunks by Jaccard >0.6. The final answer is then synthesized with the same Final Synthesizer prompt as PRUNE.

B.2 Example 1: one-shot RAG already works (Llama-3.1-8B / MuSiQue)

Many multi-hop questions are actually solved by one-shot RAG, because the top retrieved chunk happens to contain both hops. The question below has a two-hop structure—which state hosts the building, then who that state voted for in 2016—but the top chunk is the Wikipedia page for the building (which names New Hampshire), and the next chunk is the 2016 election page. The LLM reads off Hillary Clinton from these two chunks. Both routers see a high `score_top1` and don't escalate. Running PRUNE or IRCoT* would give the same answer at 3–4× the token cost.

Question. Who did the state where Christian Science Pleasant View Home was located vote for in 2016?
Gold. *Hillary Clinton.*

Step 1 – One-shot RAG. F1 = 1.00 **1,231 tokens**

Top-3 retrieved chunks: *Christian Science Pleasant View Home* (the building, Concord NH); *2016 U.S. presidential election*; *2016 U.S. presidential election in Texas*. The LLM reads off *Hillary Clinton*.

Step 2 – Features.
`confidence` = 0.6, `ans_len` = 2, `bridge_cues` = 1, `score_gap` = 0.149, `score_top1` = 0.778, `qtype` = entity.

$\mathbf{x} = [0.6,\ 2,\ 1.0,\ 0.149,\ 0.778,\ 0]$

Step 3 – Routers.
RASER-2: $p(\text{ESCALATE}) = 0.010 < \theta = 0.20 \Rightarrow$ ONE-SHOT RAG ✓

RASER-3: $\hat{f}_{\text{ONE-SHOT RAG}} = 0.378$, $\hat{f}_{\text{PRUNE}} = 0.173$, $\hat{f}^*_{\text{IRCoT}} = 0.244$.
After cost penalty ($\lambda = 5\times10^{-5}$): $0.32, -0.02, -0.01$ $\Rightarrow$ ONE-SHOT RAG ✓

Final. Both routers choose one-shot RAG. They save 2,727 tokens per question vs. always-PRUNE and 3,991 vs. always-IRCoT* with finding the correct answer.

B.3 Example 2: bridge route PRUNE fixes the answer (Gemma-3-31B / MuSiQue)

Some multi-hop questions need one bridge step to find the answer. The question below asks which province borders the province that hosts Lago District. The top-retrieved chunk is the Lago District page (it says Lago District is in Niassa Province), but no chunk specifies what borders Niassa. So, one-shot RAG answers, "I don't know." Both routers escalate to PRUNE: RASER-2's escalation probability is 0.83, well above the 0.20 threshold; RASER-3's regressor also predicts PRUNE will get the highest F1. The Bridge Proposer extracts *Niassa Province* as the bridge, the Branch Retriever pulls in Niassa-related chunks, and the Final Synthesizer reads off *Cabo Delgado Province*.

Question. What province shares a border with the province where Lago District is located?
Gold. *Cabo Delgado Province.*

Step 1 – One-shot RAG. F1 = 0.00 **1,124 tokens**

Top-3 retrieved chunks: *Lago District* (says it is in Niassa Province), *Lago Verde (Queyras)*, *Gallaratese*. No chunk says which province borders Niassa. LLM answers *"I don't know"*.

Step 2 – Features.
`confidence` = 0.0, `ans_len` = 3, `bridge_cues` = 0, `score_gap` = 0.268, `score_top1` = 0.837, `qtype` = entity.

$\mathbf{x} = [0.0,\ 3,\ 0.0,\ 0.268,\ 0.837,\ 0]$

Step 3 – Routers.
RASER-2: $p(\text{ESCALATE}) = 0.830 \geq \theta = 0.20 \Rightarrow$ PRUNE ✓

RASER-3: $\hat{f}_{\text{ONE-SHOT RAG}} = 0.506$, $\hat{f}_{\text{PRUNE}} = 0.888$, $\hat{f}^*_{\text{IRCoT}} = 0.807$.
After cost penalty ($\lambda = 1\times10^{-4}$): $0.39, 0.52, 0.42$ $\Rightarrow$ PRUNE ✓

Step 4 – PRUNE execution. F1 = 1.00 **3,603 tokens**

Bridge Proposer extracts *Niassa Province* (the province containing Lago District). Branch Retriever pulls Niassa-related chunks. Branch Verifier keeps the branch (support = 0.78). Final Synthesizer combines the bridge with the chunks and answers *Cabo Delgado Province*.

B.4 Example 3: iterative retrieval IRCOT* is needed (Llama-3.1-8B / MuSiQue)

Some questions need more than one bridge step. The question below asks for the death date of *"the person who ended the Achaemenid Empire by conquest in 330 BC"*. The bridge is Alexander the Great, but identifying him is not enough — we still need to retrieve his biography to find when he died. This is exactly where the two routers diverge. RASER-2's escalation probability is 0.164, just below the 0.20 threshold, so it stays with the one-shot *"I don't know"* (F1 = 0). RASER-3 is not gated by a threshold; its three regressors predict $\hat{f}^*_{\text{IRCoT}} = 0.50$, much higher than $\hat{f}_{\text{ONE-SHOT RAG}} = 0.01$ and $\hat{f}_{\text{PRUNE}} = 0.07$, and after the cost penalty IRCoT* still wins. IRCoT*

then iterates: round 1 extracts Alexander the Great, round 2 retrieves his biography and reads off *323 BC* (F1 = 1.00). This is the clearest case in our held-out set where the threshold hurts RASER-2 and the cost-aware argmax saves RASER-3.

Question. When did the person who ended the Achaemenid Empire by conquest in 330 BC die?
Gold. *323 BC* (Alexander the Great).

Step 1 – One-shot RAG. F1 = 0.00 **1,358 tokens**

Top-3 retrieved chunks: *Hellenistic period*, *Eudamidas I*, *Hellenistic period* (second chunk) — related to the era but never name Alexander or his death date. LLM answers *"I don't know"*.

Step 2 – Features.
`confidence`=0.8 (date prior), `ans_len`=3, `bridge_cues`=1, `score_gap`=0.059, `score_top1`=0.681, `qtype`=date.

$\mathbf{x} = [0.8,\ 3,\ 1.0,\ 0.059,\ 0.681,\ 1]$

Step 3 – Routers (here they disagree).
RASER-2: $p(\text{ESCALATE}) = 0.164 < \theta = 0.20 \Rightarrow$ ONE-SHOT RAG

RASER-3: $\hat{f}_{\text{ONE-SHOT RAG}} = 0.011$, $\hat{f}_{\text{PRUNE}} = 0.070$, $\hat{f}^{*}_{\text{IRCoT}} = 0.497$.
After cost penalty ($\lambda = 5\times10^{-5}$): $-0.05, -0.12, 0.24$
$\Rightarrow$ IRCoT* ✓

Step 4a – what RASER-2 runs. F1 = 0.00 **1,358 tokens**

RASER-2 keeps the one-shot *"I don't know"*. The probability 0.164 is just below 0.20, so the router does not escalate. F1 = 0.

Step 4b – what RASER-3 runs. F1 = 1.00 **4,138 tokens**

RASER-3 runs IRCoT*. Round 1 retrieves *Hellenistic period* chunks and extracts the fact *"Alexander the Great"*; round 2 re-retrieves with that fact added to the query, hitting Alexander's biography, and extracts *"323 BC"*. Final answer: *323 BC*.

On this question IRCoT* is both cheaper *and* more accurate than PRUNE would have been (4,138 vs. 4,306 tokens; F1 1.00 vs. 0.00), because the second hop's evidence needs a query rewrite that PRUNE's single re-retrieval cannot do.

B.5 Take-away

RASER can do three things with its budget: spend no extra LLM calls (Example 1, ONE-SHOT RAG, 1.2k tokens), spend bridge retrieval calls (Example 2, PRUNE, 2.2k tokens), or run iterative retrieval IRCoT (Example 3, IRCoT*, 4.1k tokens). RASER-2 picks between the first two actions. RASER-3 picks among all three actions. The router's job is to spend the smallest budget that still gets the correct question.

C Baseline Workflows

For comparison with the RASER workflow in Figure 3, Figures 4 and 5 show how the two LLM-driven baselines from Table 3, SELF-ASK* (Self-Ask-style) and ChainRAG, process the same Achaemenid question.

SELF-ASK* (Self-Ask). One LLM call decomposes the question into numbered sub-questions, with later sub-questions referencing earlier answers via `#N`. For each sub-question in order, we rewrite it with prior answers substituted in, retrieve the top-10 chunks, and ask the LLM. The last sub-answer is the final output. On the Achaemenid question, this takes 4 LLM calls and about 3,500–4,500 tokens.

ChainRAG. ChainRAG adds two steps on top of Self-Ask. First, a multi-hop judge checks whether the question is multi-hop; only multi-hop questions go through decomposition. Second, before processing the sub-questions, it builds a sentence-level graph over the top-100 retrieved chunks. The graph has three edge types: similarity (k-nearest neighbors by nomic cosine), positional (adjacent in the same document), and entity (sharing a named entity). Each sub-question is answered by retrieving seed sentences and expanding their 2-hop neighbors on the graph. Later sub-questions are rewritten with prior answers (*entity completion*). On the Achaemenid question, this takes 4–5 LLM calls and about 4,500–5,500 tokens.

D Main results visualization

Figure 6 shows the visualization of Table 3. Which provides a clear vision about how RASER performs compared with other baselines.

E Router Diagnose

RASER-2. Only 9–16% of questions actually benefit from PRUNE over ONE-SHOT RAG. RASER-2 escalates 14% of the time, close to that rate —it neither over- nor under-escalates. Among questions the router assigns high probability, the bridgeable rate is about 3× higher than among low-probability questions, so the score is meaningful.

RASER-3. Pooled across LLMs and datasets, RASER-3 sends 64% of questions to ONE-SHOT RAG, 12% to PRUNE, and 24% to IRCoT*. The

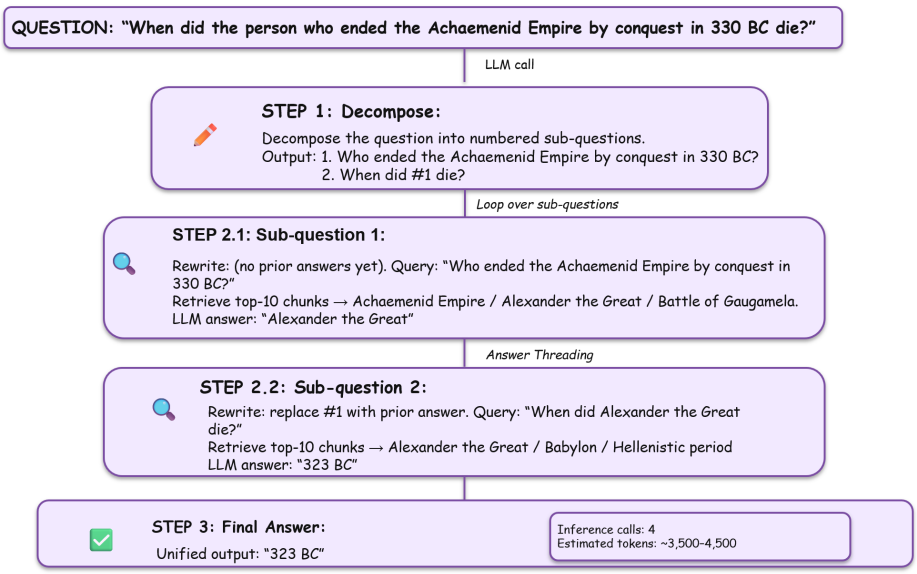

Figure 4: **SELF-ASK* (Self-Ask) baseline on the Achaemenid question.** Step 1: one LLM call decomposes the question into two sub-questions. Step 2.1: Retrieve and answer the first sub-question (*Alexander the Great*). Step 2.2: Rewrite the second sub-question by substituting #1 with the first answer (*answer threading*), retrieve and answer (*323 BC*). Step 3: The last sub-answer is the final output.

mix shifts per cell: cells where IRCoT* is the best fixed route (mid-tier LLMs on MuSiQue/2Wiki) get $> 30\%$ IRCoT*; cells where ONE-SHOT RAG is already strong (frontier LLMs on HotpotQA) stay at $> 80\%$ ONE-SHOT RAG.

F Classifier Ablation

Table 7 is the full version of the classifier ablation we summarized in the main text (Table 6). The main text shows only 3 of the 6 classifiers per router for space; here, we include all of them. The setup is identical: the same six features, the same LLM and dataset, the same threshold $\theta = 0.20$ for RASER-2, same cost-budget rule for RASER-3's λ. The only thing that changes per row is the classifier (for RASER-2) or the route-value regressor head (for RASER-3).

We tried six model families. The tree-based ones are sklearn GBM, XGBoost, LightGBM, and CatBoost, all with the same hyperparameters (100 trees, max depth 3, learning rate 0.1, and subsample 0.8). The linear baseline is logistic regression for RASER-2 and Ridge Regression for RASER-3. The MLP baseline is a single hidden layer of 32 ReLU units, the same architecture for both routers. For the linear models and the MLP, we standardized the features first. Tree models are scale-invariant, so we did not.

The take-away is the same as the main text. For RASER-2, the routed F1 spans only 0.011 across all six classifiers (0.520 to 0.531). For RASER-3, it spans 0.016 (0.546 to 0.562). The deployed sklearn GBM is not strictly the best in either case; every alternative falls within the 95% bootstrap CI on F1. What drives RASER's performance is not the choice of model but the six features.

The full table also makes one thing visible that the main text does not: the linear and shallow models do not really learn the middle route. In RASER-3, Ridge sends only 5% of questions to PRUNE, and the MLP only 6%; the four tree-based regressors send 7–12%. Ridge and the MLP flatten the middle option, defaulting to either ONE-SHOT RAG or IRCoT*. The tree models learn to use PRUNE as a real cheap middle choice, and that is where the extra 0.013–0.016 F1 comes from.

G Threshold and cost-budget sensitivity

RASER has two settings the operator can adjust:

RASER-2's threshold θ controls how confident the router has to be before escalating to PRUNE. Lower θ means the router escalates more often; higher θ means it stays close to one-shot RAG.

RASER-3's cost-budget fraction controls how much the router is allowed to spend. We set this as a percentage of always-IRCoT*'s token cost (on training data); the actual λ in Eq. 1 is then derived to satisfy this budget.

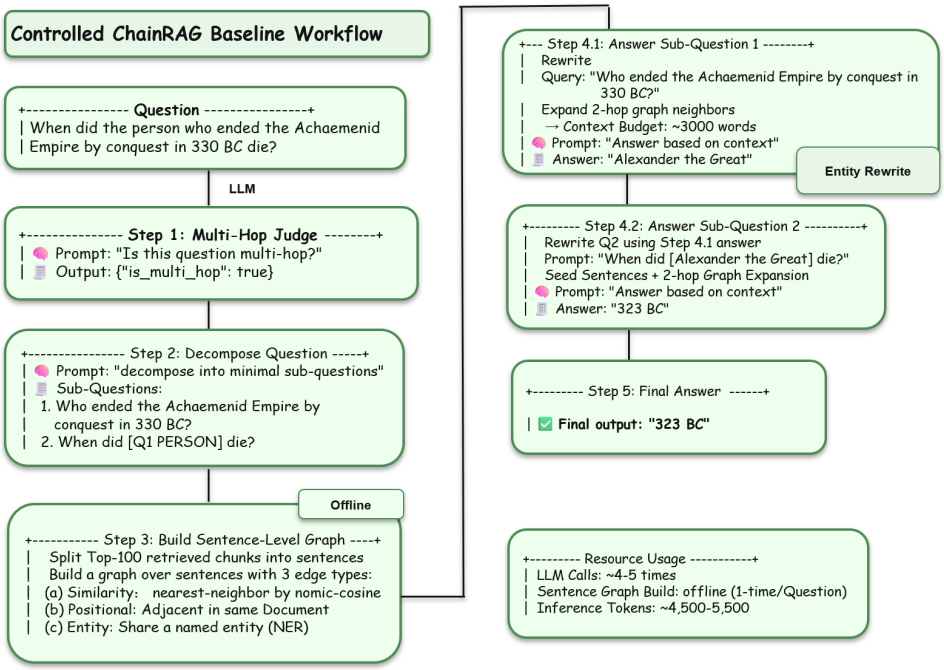

Figure 5: **ChainRAG baseline on the Achaemenid question.** The pipeline adds a multi-hop judge (Step 1) and an offline sentence-level graph build (Step 3) with three edge types (similarity, positional, entity) on top of decomposition. Each sub-question retrieves seed sentences and expands their 2-hop neighbors on the graph. Sub-question 2 is rewritten by replacing the reference to the first sub-answer.

In the paper, we use $\theta = 0.20$ and cost-budget $= 0.60$. This appendix sweeps both settings to show our choices are not cherry-picked: small changes give small differences. Everything else is held fixed (the same six features, the same 5-fold dataset cross-validation, and the same `sklearn` GBM head, pooled across all six LLMs and three datasets).

Threshold θ sweep for RASER-2. Table 8 shows what happens as θ moves from 0.10 (escalate aggressively) to 0.30 (stay close to one-shot RAG). Both F_1 and tokens move smoothly—no cliff, no hidden sweet spot. Across all five settings, F_1 spans only 0.014 and tokens span 349. The deployed $\theta = 0.20$ is not the highest-F_1 setting; $\theta = 0.10$ would gain 0.010 more F_1 but cost 242 extra tokens per question. We picked 0.20 because it gives the smallest token cost while staying within 0.010 F_1 of the maximum.

Cost-budget sweep for RASER-3. The raw λ in the cost-aware argmax (Eq. 1) is hard to interpret on its own: "$\lambda = 5 \times 10^{-5}$" tells you nothing about how much the router will spend. We use a plain-English setting instead: *spend at most $X\%$ of always-*IRCoT*'s tokens on the training fold.* We then pick the largest λ that meets this limit, automatically per (LLM, dataset) cell.

Table 9 sweeps the limit from 0.33 to 1.00. The two ends are sanity checks: at 0.33 the router is forced to be cheap and collapses to ONE-SHOT RAG (99% of questions stop), so RASER-3 behaves like one-shot RAG; at 1.00 it can freely use IRCoT* on almost half the questions and gets the highest F_1 (0.583). The deployed 0.60 sits at the knee of the curve: moving the budget from 0.33 to 0.60 buys $+0.058$ F_1; moving from 0.60 to 1.00 buys only another $+0.021$ but costs 50% more tokens. This is the same trade-off as the Pareto curve in Figure 2, but with the setting expressed as a token-spend percentage instead of a raw λ number.

Router	Model	F_1	Avg. tokens	Route mix (%)
RASER-2	`sklearn` GBM	0.520	1514	86/14
	Logistic Regression	0.531	1549	85/15
	MLP-32	0.530	1597	82/18
	XGBoost	0.526	1575	83/17
	LightGBM	0.526	1607	82/18
	CatBoost	0.529	1555	84/16
RASER-3	`sklearn` GBM	0.562	2157	64/12/24
	Ridge Regression	0.546	1844	76/5/18
	MLP-32	0.559	1908	74/6/19
	XGBoost	0.562	2094	67/10/23
	LightGBM	0.558	1937	73/9/18
	CatBoost	0.562	1989	72/7/21

Table 7: Classifier ablation. All models are evaluated on the same setting over six LLMs and three datasets. For RASER-2, different binary classifiers, produce a narrow F_1 range of 0.011. For RASER-3, different route-value regressors, a similarly narrow F_1 range of 0.016. Thus, the main conclusion is not tied to a particular implementation of the router. We use the `sklearn` gradient boosting model because it is simple, standard, and reproducible, not because it is uniquely the best. S/P/I denotes the percentage of questions routed to ONE-SHOT RAG, PRUNE, and IRCOT*.

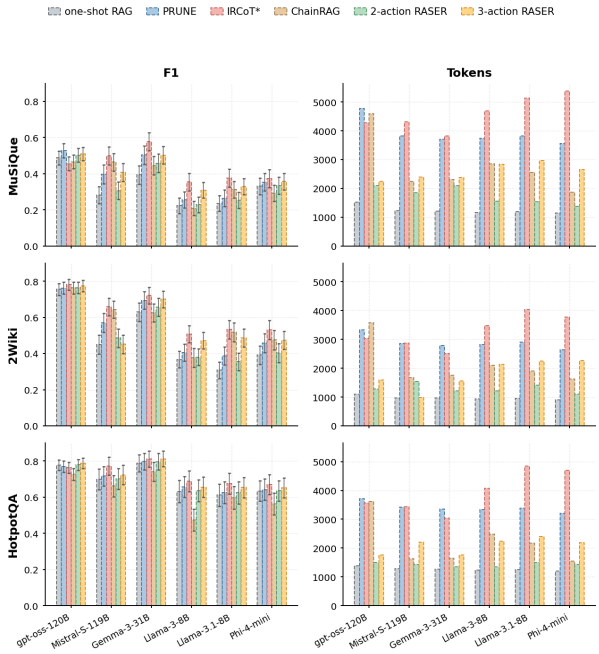

Figure 6: Visualization of Table 3, **Bottom:** average tokens for each benchmark. The Self-Ask* baseline is in the table 3.

θ	F_1	Avg tokens	Escalation %
0.10	0.530	1,756	24.0%
0.15	0.525	1,604	17.9%
0.20 (deployed)	**0.520**	**1,514**	**14.1%**
0.25	0.518	1,455	11.6%
0.30	0.516	1,407	9.7%

Table 8: RASER-2 threshold sensitivity. Across the five θ values the routed F_1 moves by 0.014 total and the token cost moves by 349. The deployed $\theta = 0.20$ is not the F_1 maximum but is within 0.010 of it while spending 242 fewer tokens per question.

Budget	F_1	Avg tokens	Route rate
0.33	0.504	1,190	99/1/0
0.50	0.525	1,653	82/7/11
0.60 (deployed)	**0.562**	**2,157**	**64/12/24**
0.75	0.578	2,665	44/19/37
1.00	0.583	3,141	25/28/47

Table 9: RASER-3 cost-budget sensitivity. Increasing the budget from 0.33 to 1.00 traces the same Pareto curve as Figure 2, but parameterized by interpretable fractions instead of λ values. The deployed 0.60 is balanced: most of the F_1 gain (above always-ONE-SHOT RAG at 0.33) is bought by the first 30 percentage points of budget; the last 40 % points add little F_1 but spend a lot more tokens.