Why-What-How RAG: SLM-Guided Low-Compute QA

Group Members

1. Akshat Saxena (230099)
2. Akshay Reddy (230517)
3. Om Chaudhari (230715)
4. Himesh Singh (230478)
5. Tattwa Shiwani (231089)

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities on diverse question–answering (QA) tasks when properly prompted. For example, GPT-3 (175B) matches or exceeds many fine-tuned systems on NLP benchmarks using only in-context learning with a few exemplars or task descriptions. Modern prompting techniques such as chain-of-thought (CoT) further improve multi-step reasoning by encouraging step-by-step explanations rather than single-shot answers. However, even strong LLMs still struggle with difficult knowledge-intensive or commonsense tasks without access to external information, especially when benchmarks are designed to be adversarial or "Google-proof."

At the same time, there has been a rapid rise of *small language models* (SLMs) in the 2–7B parameter range. Recent models, such as Phi-3-mini (3.8B) and Gemma 2 (2–9B), achieve surprisingly strong performance on benchmarks like MMLU while being affordable enough to run on a single consumer GPU or even on edge devices. In many settings, they offer a much better quality–cost–latency trade-off than monolithic large models. In parallel, retrieval-augmented generation (RAG) has emerged as a straightforward yet powerful approach to integrate LLM reasoning with explicit evidence from search engines or knowledge bases.

These trends motivate *hybrid* QA pipelines that combine a small local model with a larger cloud model. In this work, we explore such a design. We use a small language model (SLM, ≈3B parameters) to decompose questions, generate web search queries, and summarize retrieved snippets into concise hints. A larger LLM (Gemini) then answers the original multiple-choice question conditioned on those hints. Intuitively, the SLM acts as a low-cost "planner" and retrieval controller, while the LLM is reserved for the final, high-quality decision.

We evaluate this approach on several multiple-choice QA benchmarks: ARC-Challenge (science), HellaSwag (commonsense), GPQA (graduate science), a mixed MMLU subset, MedMCQA (medical), and MUSR (narrative reasoning). We compare this approach to a baseline where Gemini answers directly with CoT prompting, without any SLM or retrieval. Our experiments suggest that SLM-guided decomposition and retrieval can substantially improve accuracy on challenging reasoning-heavy datasets like HellaSwag and MUSR, while being competitive on others. We also discuss where this strategy is beneficial, where it is less effective, and why small models are particularly well-suited for this decomposition and retrieval role.

2 Related Work

2.1 Instruction and Chain-of-Thought Prompting

Modern LLMs exhibit strong performance via in-context learning. Large models such as GPT-3 can often match or exceed fine-tuned systems on many tasks by leveraging instruction exemplars in the prompt. For reasoning tasks, chain-of-thought prompting has been shown to greatly enhance multi-step problem-solving. Providing intermediate reasoning steps or even a simple instruction, such as "let's think step by step," improves accuracy on arithmetic and logical tasks. These methods demonstrate that LLMs can perform complex reasoning when guided to generate intermediate thoughts; however, they do not address the issue of missing factual knowledge.

2.2 Retrieval-Augmented QA

A complementary line of work enriches LLM inputs with external knowledge. Retrieval-augmented generation (RAG) combines a parametric language model with a non-parametric retriever over sources such as Wikipedia or the web. By retrieving supporting passages and conditioning answer generation on them, RAG models produce more specific and factual answers than unguided generation, and reduce hallucinations on open-domain QA tasks. Variants of RAG use multi-hop retrieval, decomposition into sub-questions, or iterative refinement of queries to handle complex questions.

2.3 Small Language Models and Hybrid Pipelines

Small language models have emerged as an attractive alternative to monolithic LLMs. Recent families, such as Phi-3 and Gemma-2, show that carefully trained models with 2–7B parameters can approach the quality of much larger systems across language, coding, and math benchmarks, while being 10–30 times cheaper to run and easier to deploy on edge hardware. In parallel, several works have investigated whether *SLM + retrieval* can replace or complement large models in specialized domains such as education or software engineering. These studies report that small models equipped with RAG can match or even outperform larger models on domain-specific tasks while using a fraction of the compute budget.

Our work aligns with this trend, focusing specifically on multiple-choice QA. Rather than using an SLM as the final answerer, we use it as a *decomposer and retriever* that generates targeted sub-questions and distilled hints for a larger model. This division of labor lets us benchmark how much value a small model can add *on top of* an already strong LLM, and when that extra structure leads to measurable gains.

2.4 DRAG and Distillation-based RAG for SLMs

Recent advances in retrieval-augmented generation (RAG) show that combining parametric models with explicit external evidence greatly improves factual accuracy and reduces hallucination. DRAG introduces an evidence- and knowledge-graph–based distillation pipeline that transfers RAG-style capabilities from very large teacher models to smaller student models, demonstrating strong accuracy gains while aiming to reduce hallucination. DRAG's multi-stage recipe—teacher evidence generation, LLM-based evidence ranking, graph construction, and student conditioning—establishes a high-performing template for making SLMs more factual and robust.

2.5 Prior Approaches

Existing RAG methods span a spectrum of design choices:

- **Naive RAG:** retrieve documents from a large corpus and condition generation on the retrieved passages. This is simple and effective but requires a high-quality retriever and large context windows.
- **Graph-based RAG:** convert retrieved evidence into structured subgraphs (entities + relations) and use graph context to provide structured grounding, improving global consistency for complex queries.

- **Teacher→Student distillation (e.g., DRAG):** use a large RAG teacher to generate ranked evidence and knowledge graphs, then distill that structured, ranked evidence into an SLM. DRAG shows that evidence- and graph-based distillation can transfer strong factual knowledge to small models and reduce hallucination, but it requires costly teacher-stage computation and careful ranking/aggregation.
- **Mini / lightweight RAG (MiniRAG, Self-RAG, SimRAG):** various efforts adapt RAG for small models via efficient retrievers, LoRA-style fine-tuning, or self-improvement loops; they trade off some fidelity for lower compute. DRAG often outperforms these on benchmarks by using richer evidence and graph distillation.

2.6 Why DRAG's limitations matter

DRAG achieves strong improvements by leveraging teacher LLMs to curate evidence and graphs, but the distillation pipeline introduces two practical pain points:

- *Compute and time overhead* during teacher evidence + graph generation and ranking.
- *Operational complexity* in building and ranking multi-evidence sets and graphs, plus the potential for leakage or noisy ranking to affect the distilled model.

The distillation process, while efficient at inference-time for the student, still requires non-negligible compute during distillation and evidence generation.

2.7 How our SLM+retrieval+Gemini hybrid addresses DRAG's limitations

Instead of performing heavy teacher-driven distillation, our pipeline keeps the small-model inference-time and retrieval responsibilities local, and uses the stronger LLM only at the decision point. This reduces up-front training/distillation cost while retaining many practical benefits of RAG:

- **No full distillation step required:** our method uses an off-the-shelf instruction-tuned SLM to produce WHY/WHAT/HOW sub-queries and to condense retrieved snippets into compact hints; we do not fine-tune or distill the SLM, avoiding the cost of running a large teacher across a large training set.
- **Runtime decomposition and retrieval:** the SLM runs locally as a lightweight planner (decomposer + hint extractor). Retrieval is done per-query, so we only pay retrieval and final LLM cost for questions that need it; training cost is minimal.
- **Adaptive and incremental:** because decomposition and hint extraction happen at runtime, the pipeline can be made adaptive (invoke hints only on low-confidence or high-stakes queries), dramatically reducing average cost compared to an expensive offline distillation.
- **Simpler engineering:** we avoid building and maintaining a knowledge-graph index and graph-ranking infrastructure; instead we rely on web/snippet ranking plus small-model summarization. This reduces implementation complexity and storage/maintenance costs compared to DRAG's graph-based pipeline.

The main trade-off is that per-question token usage with the larger model can increase (e.g., ∼1,259 tokens/question on MUSR), so adaptive invocation and confidence gating are important to keep monetary costs manageable. Retrieval noise remains a risk; adopting snippet ranking, domain filters, and query paraphrase loops can help mitigate it.

2.8 Key Distinctions from DRAG

In summary, our method differs from DRAG along several axes:

- **Role split:** DRAG uses a teacher LLM to generate evidence/graphs followed by offline distillation into an SLM; our method uses an SLM (local) to decompose and control web retrieval at runtime and a large LLM (cloud) to finalize the answer.

- **Training vs. runtime cost:** DRAG spends compute up-front on evidence generation and distillation; we spend mostly runtime tokens and local SLM compute but avoid a heavy offline distillation pass.
- **Structured graphs vs. concise hints:** DRAG explicitly builds knowledge graphs for compactness/-efficiency; we extract short, verifiable bullet hints designed to fit in the LLM context window. Graphs reduce token cost but require graph construction code and ranking heuristics; hints are simpler to produce but cost more tokens per query.
- **Privacy model:** DRAG demonstrates privacy-preserving variants with de-identified teacher queries; our pipeline can similarly rephrase or strip PII before retrieval but more of the processing stays local, reducing exposure risk.

Figure1 empirically compares MiniRAG, DRAG, and our method on a shared LLAMA-3.2-3B backbone across three datasets.

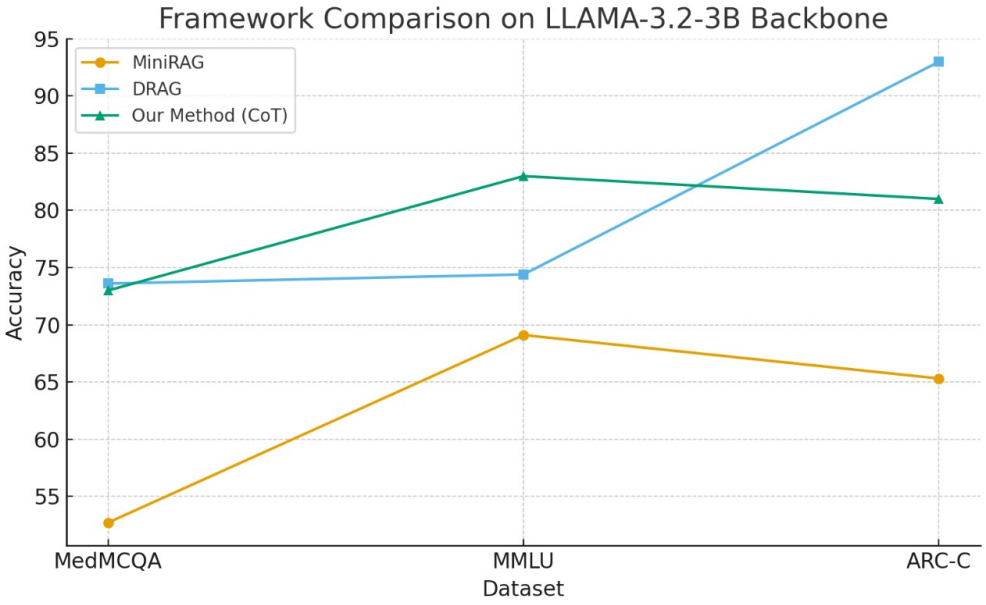

Figure 1: Framework comparison on the LLAMA-3.2-3B backbone: MiniRAG, DRAG, and our SLM+retrieval framework on MedMCQA, MMLU, and ARC-Challenge. Our method matches or outperforms MiniRAG and is competitive with DRAG while avoiding heavy offline distillation.

3 Methodology

We compare two approaches for multiple-choice QA:

- **Baseline (Gemini-only):** A single LLM (Gemini) receives the question and four answer options, and is asked to output the correct option letter. We experiment with several prompting variants; in this report, we focus on the *chain-of-thought* (CoT) variant, where the prompt includes "think step by step" and the model is allowed to generate short reasoning before the final choice.
- **Our Method (SLM + decomposition + retrieval + Gemini):** A small local model orchestrates decomposition and retrieval, and Gemini only appears in the final step. The pipeline consists of four stages, described below.

3.1 Small-model decomposition into sub-queries

Given the original multiple-choice question, we first use a 3B-parameter instruction-tuned SLM (e.g., a LLaMA-style or Phi-style model) to rewrite the question into three focused sub-queries labelled `WHY`, `WHAT`, and `HOW`. The prompt instructs the SLM to output exactly:

```
WHY: <why-question>
WHAT: <what-question>
HOW: <how-question>
```

The idea is to systematically cover causal, definitional, and procedural aspects of the original problem. For example, for a photosynthesis question, the sub-queries might be "Why do plants need carbon dioxide?", "What gas is consumed in photosynthesis?", and "How does photosynthesis convert light into chemical energy?"
This stage is deliberately handled by the SLM because:

1. It is structurally constrained (the model only needs to fill in three short lines), making it a good fit for smaller models.
2. It is invoked once per question, so running it locally avoids API costs.
3. We found that small models are sufficiently capable at query rewriting when given clear instructions.

3.2 Web retrieval for each sub-query

For each of the three sub-queries, we perform a web search using the Serper API and retrieve a small number of snippets (up to two per query). These snippets are short fragments of web pages or search summaries that contain candidate facts. We store the snippets together with their source URLs for potential inspection.

If the search API is unavailable (for example, due to missing credentials), we fall back to a generic placeholder snippet. In all reported experiments, however, we used real search results.

3.3 Hint extraction with the SLM

Next, we prompt the SLM again to transform the raw snippets into a small set of bullet-point hints. The SLM receives the original question, the three labeled sub-queries, and all retrieved snippets. It is instructed to output 4–8 concise, verifiable facts, each starting with a dash ("-"), and *not* to answer the question yet. For instance, for the photosynthesis example the hints might look like:

- Carbon dioxide is a key reactant in photosynthesis.
- Plants use sunlight to convert carbon dioxide and water into glucose.
- Oxygen is released as a byproduct of photosynthesis.

This distillation step reduces noise from retrieval and produces a compact hint set that can fit comfortably into the LLM's context window. Again, a small model suffices here because the task is essentially extractive summarization under strong instruction.

3.4 Final answer with Gemini

Finally, we construct a prompt for Gemini containing:

1. The original question and its four answer choices (labelled A–D).
2. An optional "Hints" section listing the SLM-generated bullet points.
3. An instruction to think step-by-step and then output only the final letter (A/B/C/D).

For example, the prompt might include:

```
Hints (you may use or ignore these):
- <hint 1>
- <hint 2> ...
```

We parse Gemini's response and record whether the predicted letter matches the gold label. In all our CoT experiments, Gemini is allowed a short reasoning segment (a few sentences) before the final letter, but only the letter is scored.

4 Experimental Setup

4.1 Datasets

We evaluate on eight multiple-choice QA datasets, using 100-question subsets from each to stay within practical API limits:

- **ARC-Challenge (ARC):** Grade-school science questions from the AI2 Reasoning Challenge. We sample 100 questions from the Challenge set.
- **HellaSwag:** Commonsense story completion where the model chooses the most plausible continuation. We sample 100 validation examples.
- **GPQA:** Graduate-level science questions in physics, chemistry, and related areas. We use a 100-question subset.
- **MMLU:** A 100-question mixed subset of MMLU multiple-choice tasks spanning several disciplines.
- **MedMCQA:** 100 medical multiple-choice questions focusing on clinical and basic-science knowledge.
- **MUSR:** A 100-question subset from the MUSR benchmark, which requires reasoning about long narrative contexts (e.g., murder mysteries and object placements) with four answer options.
- **TruthfulQA:** Questions designed to test whether the model avoids common misconceptions, false assumptions, and imitative falsehoods. We sample 100 multiple-choice questions covering health, law, and general knowledge.
- **AveriTeC:** A hallucination-focused benchmark evaluating whether the model gives factually grounded answers across science, commonsense, and world-knowledge categories. We sample 100 questions from the validated subset.

Each example has four options (A–D) and a single correct answer. For ARC, HellaSwag, and MUSR we run both the *baseline* and *our method*. For GPQA, MMLU, and MedMCQA we currently report only *our method* due to API quota constraints.

4.2 Prompting and evaluation

All runs reported here use the chain-of-thought (CoT) prompting variant for Gemini. That is, Gemini is encouraged to think step-by-step and then choose among A, B, C, and D. For the SLM, we use deterministic decoding with strong formatting instructions so that the outputs are easy to parse and feed into retrieval.

We measure accuracy as the fraction of questions where the predicted option matches the gold answer. For datasets with both variants, we report:

- **Baseline (Gemini CoT):** Gemini answers directly with CoT, no SLM, no retrieval.
- **Our Method (SLM + retrieval + Gemini CoT):** Full pipeline described above.

For MUSR we also log the Gemini token usage for our method. On the 100-question subset, the total prompt tokens used by Gemini were approximately 125,802, with 100 output tokens, for a total of 125,902 tokens. This corresponds to roughly 1,259 tokens per question when using hints.

Dataset	Prompting	Baseline (Gemini)	Our Method	Δ (Our–Base)
ARC-Challenge	CoT	89%	81%	−8%
HellaSwag	CoT	71%	87%	+16%
MUSR	CoT	44%	52%	+8%
GPQA	CoT	39%	46%	+7%
MMLU	CoT	80%	83%	+3%
MedMCQA	CoT	74%	73%	−1%
AveriTeC	CoT	52%	57%	+5%
TruthfulQA	CoT	73%	64%	−9%

Table 1: Chain-of-thought accuracy on 100-question subsets. Baseline uses Gemini alone; our method uses SLM decomposition, web retrieval, hint extraction, and Gemini.

5 Results and Analysis

5.1 CoT accuracy across datasets

Table 1 summarizes the CoT accuracies on our 100-question subsets. Results are reported as fractions between 0 and 1.

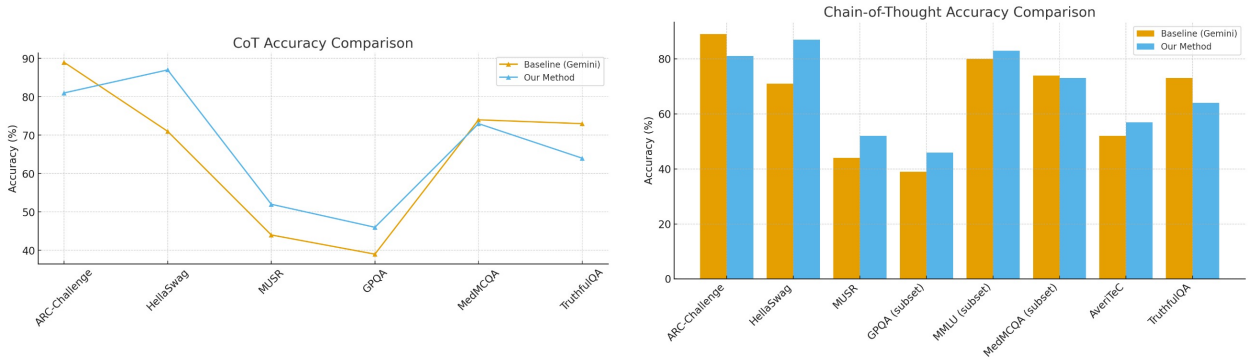

Figure 2: CoT accuracy comparison between the baseline (Gemini-only) and our SLM+retrieval method across datasets. Left: line plot; right: bar chart. The two panels are aligned horizontally for easier visual comparison.

On **HellaSwag**, our method improves accuracy from 0.71 to 0.87, a gain of 16 percentage points. This dataset requires subtle commonsense reasoning about plausible story continuations, and we observe that web-based hints often supply key facts (e.g., typical sequences of events, physical plausibility) that Gemini alone sometimes misses. The hints effectively anchor Gemini's chain of thought around realistic scenarios.

On **MUSR**, our method improves accuracy from 0.44 to 0.52 (+8 points). Here the challenge is reasoning about relatively long and descriptive narratives. The decomposition into WHY/WHAT/HOW questions helps retrieve specific details (e.g., character relationships, object locations) that the SLM then distills into hints. These hints seem to steer Gemini away from plausible but incorrect suspects or locations.

For **ARC-Challenge**, the baseline Gemini CoT accuracy is already relatively high at 0.89, and our method slightly *reduces* accuracy to 0.81. Inspection of failure cases suggests that when the base model already knows the relevant science fact, noisy or partially relevant web snippets can sometimes distract it. In other words, retrieval helps more on genuinely knowledge- or context-starved tasks than on straightforward textbook questions that the LLM already handles well.

For **AveriTeC**, our method improves accuracy from 0.52 to 0.57 (+5 points). Since AveriTeC is designed

Dataset	Prompting	Baseline (Gemini)	Our Method	Δ (Our–Base)
ARC-Challenge	Instruction	94%	83%	−11%
HellaSwag	Instruction	74%	66%	−8%
MUSR	Instruction	35%	59%	+24%
GPQA	Instruction	31%	33%	+2%
AveriTeC	Instruction	45%	46%	+1%
TruthfulQA	Instruction	74%	74%	0%
MedMCQA	Instruction	70%	77%	+7%

Table 2: Instruction-based prompting accuracy on 100-question subsets. Baseline uses Gemini alone; our method uses SLM decomposition, web retrieval, hint extraction, and Gemini.

to probe hallucination resistance, many questions require grounding in small but crucial factual details. We find that the SLM-driven decomposition frequently retrieves concise, well-verified snippets (e.g., definitions, factual statements, domain-specific rules) that help anchor Gemini's chain of thought in concrete evidence. In several cases, the hints prevented Gemini from producing confident but subtly incorrect answers, indicating that retrieval provides useful factual guardrails on this benchmark.

For **TruthfulQA**, accuracy drops from 0.73 to 0.64 (9 points). TruthfulQA intentionally contains questions where misleading heuristics, common myths, or imitative falsehoods can trick large models. We observe that web retrieval sometimes amplifies these issues: surface-level search snippets may echo the very misconceptions that the dataset is designed to expose, pulling the model toward familiar but incorrect statements. In contrast, Gemini's baseline CoT—without external retrieval—tends to rely more on its internal fact-checked representations, which appear more reliable for this benchmark. This suggests that retrieval must be carefully filtered or fact-verified when dealing with misinformation-sensitive tasks like TruthfulQA.

On **GPQA**, **MMLU**, and **MedMCQA**, we currently only have results for our method (0.46, 0.83, and 0.73, respectively). These values are reasonable given the difficulty of the tasks and the small sample size (100 questions each), and they demonstrate that our pipeline can handle graduate-level science, broad multi-domain questions, and specialized medical content within a unified framework. In future work, we plan to add direct Gemini baselines for these datasets to quantify the gains more precisely.

5.2 Instruction-based accuracy

Table 2 reports the corresponding accuracies under instruction-based prompting, where Gemini is asked to answer concisely without explicit CoT.

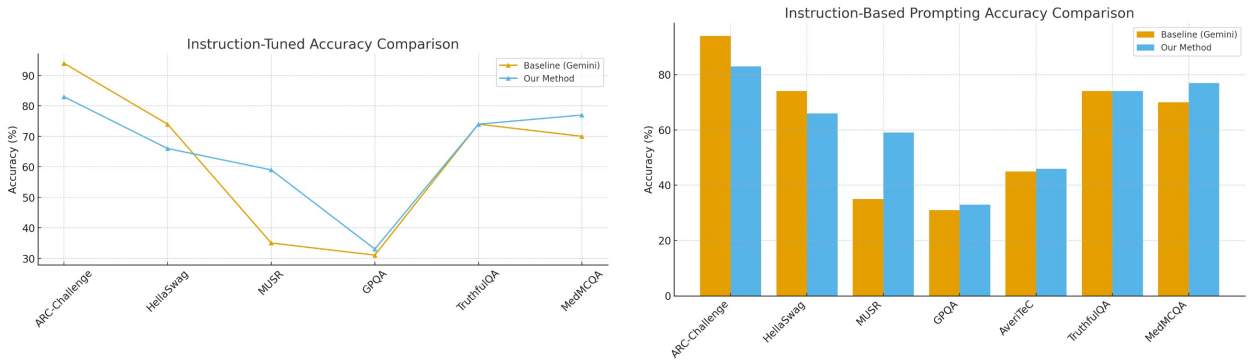

Figure 3: Instruction-based accuracy comparison between the baseline (Gemini-only) and our SLM+retrieval method across datasets. Left: line plot; right: bar chart. The two panels are aligned horizontally for consistency with Figure 2.

5.3 Token usage and cost considerations

For the MUSR benchmark, our method uses roughly 1,259 Gemini tokens per question when hints are included. The small-model stages (decomposition and hint extraction) run locally and do not incur API cost. While this per-question token count is higher than a pure baseline prompt, our results suggest that the accuracy gains on challenging datasets like HellaSwag and MUSR can justify the additional tokens in settings where quality is more important than minimal cost.
In practical deployments, one could make the pipeline *adaptive*: first query Gemini directly, then invoke the SLM+retrieval steps only when the model appears uncertain (e.g., low confidence in options) or when a question is flagged as high-stakes. This would retain most of the gains while greatly reducing average token usage.

5.4 Why SLMs help: empirical and conceptual evidence

Conceptually, small models are attractive in our pipeline because they are:

- **Cheap and fast:** Decomposition and summarization require quick iteration over many questions; using an SLM avoids paying full LLM prices at this stage.
- **Sufficiently capable:** Our experiments show that a 3B model can reliably produce structured sub-queries and concise hints for these datasets, which is all we need.
- **Less hallucination-prone in restricted formats:** When constrained to short bullet points or query rewrites, small models can behave quite conservatively, often copying or lightly rephrasing retrieved facts rather than inventing new ones.

Empirically, the gains on HellaSwag and MUSR provide evidence that an SLM-guided retrieval layer can make a strong LLM meaningfully better on difficult reasoning tasks. Taken together with external reports that SLM+RAG systems can match or rival much larger models on domain-specific problems, our results support the thesis that *small models are particularly well-suited to planning, retrieval, and hint-generation roles in hybrid QA systems.*

5.5 Interpretation of Results and Problem Analysis

Across all datasets, our hybrid SLM-guided retrieval pipeline shows non-uniform gains, revealing important insights about when retrieval helps, when it hurts, and what underlying problems our current pipeline still faces.
First, we observe large improvements on tasks that require background knowledge or subtle commonsense priors, such as HellaSwag and MUSR. These tasks typically contain scenarios where the model must understand everyday plausibility, social conventions, or narrative consistency—patterns that are not always stored reliably in the LLM parameters. In such cases, the WHY/WHAT/HOW decomposition tends to fetch external information that anchors Gemini's chain of thought. The pipeline essentially injects "missing priors" into the LLM, preventing it from relying on shallow heuristics.
However, the pipeline performs worse than the baseline on datasets like ARC-Challenge or TruthfulQA, even though retrieval was expected to help. Upon inspection, the problem becomes clearer:

- **The LLM already knows the fact.** ARC-Challenge covers grade-school science, which Gemini handles well. The injected hints sometimes introduce tangential or irrelevant facts, causing the model to overthink or follow misleading context.
- **Web search is noisy.** The decomposition occasionally produces sub-queries that do not align perfectly with the ground-truth question. This leads to retrieved snippets that are partially correct but not scoped to the question's constraints, confusing the LLM.
- **Hint extraction is too extractive.** The SLM often paraphrases raw snippets without filtering domain-specific ambiguity. This can amplify imprecise web statements, especially for medical or scientific datasets.

- **Truthfulness trade-offs.** Retrieval helps with knowledge gaps, but it also introduces noise that reduces consistency on truthfulness-oriented datasets. If the snippets contain trivia-like content or ambiguous facts, the LLM may incorrectly trust the wrong hint.

Overall, the results reveal a core problem: the pipeline does not adapt dynamically. When retrieval helps, it helps a lot; when the LLM already knows the answer or the search returns irrelevant noise, performance degrades. This highlights the need for an adaptive retrieval controller or confidence-based routing so that the SLM pipeline is invoked only when necessary, not universally.

6 Discussion

6.1 When does SLM-guided retrieval help?

Our results suggest the following pattern:

- On **knowledge-intensive** or **commonsense-heavy** tasks where the base LLM is often missing crucial context (HellaSwag, MUSR), SLM-guided retrieval and hinting yields clear gains.
- On **moderately difficult but well-covered** science questions (ARC-Challenge), the base LLM already has strong internal knowledge. In such cases, extra snippets can occasionally introduce misleading details and hurt performance.

This indicates that hybrid SLM+LLM pipelines should be used selectively, especially when the base model is already very strong on a domain. An interesting direction for future work is to train a meta-model that predicts when retrieval is likely to help and only invokes the SLM pipeline in those cases.

6.2 Future extensions

We plan several extensions:

- **Instruction and non-CoT evaluation:** In addition to CoT, we will evaluate both methods under normal and instruction prompting. The LaTeX skeleton already includes placeholders for inserting those results once they are available.
- **Additional benchmarks:** We aim to add AveriTeC (truthfulness calibration) and TruthfulQA to probe hallucination behavior more directly, again comparing Gemini-only and SLM+retrieval variants.
- **Learned decomposition:** Currently, our SLM decomposition template is fixed (WHY/WHAT/HOW). A more advanced approach could learn the best set of sub-queries per domain or dataset, potentially improving retrieval coverage and reducing noise.

6.3 Limitations: Dataset Size, Sampling Bias, and Resource Constraints

A key limitation of our experimental evaluation is that for each dataset, we used only 100 randomly sampled questions. This decision was driven primarily by:

- API cost constraints,
- the time required to run the full pipeline, and
- local GPU limitations for the SLM stages.

While 100 samples provide a rough signal, they are vulnerable to sampling bias. A random subset may inadvertently contain examples that overrepresent certain skills or underrepresent others. Therefore, although our results reveal meaningful trends, we openly acknowledge that the selective sample may not fully reflect the true benchmark performance.

Another limitation is the computational cost of retrieval. Our pipeline greatly increases prompt size in

the final Gemini call, which restricts scalability when evaluating thousands of examples.

We also recognize that our retrieval module relies on web search, where snippet quality varies widely. Without domain filtering or per-query ranking, even a correct decomposition can still return low-value text.

We also wanted to train the SLM and apply similar approach but we couldnt do that due to lack of compute.

These constraints shaped our experimental design, and while our findings are directionally valid, a more comprehensive evaluation with larger sample sizes and better-fine-tuned SLM components would increase robustness.

7 Conclusion

We presented a modular framework for multiple-choice QA that combines a small local model for decomposition and retrieval with a strong LLM for final answering. The SLM generates structured sub-queries, controls web search, and distills retrieved snippets into concise hints, while Gemini performs chain-of-thought reasoning over the original question plus hints.

On our QA benchmarks, this hybrid approach improves CoT accuracy substantially on challenging commonsense and narrative reasoning tasks (HellaSwag, MUSR), remains competitive on science and specialized domains (ARC, GPQA, MMLU, MedMCQA, AveriTeC), and highlights interesting failure modes on TruthfulQA where naive retrieval can hurt truthfulness. Our findings support the broader view that small language models, when combined with retrieval, are not merely cheaper versions of LLMs but powerful components for orchestrating reasoning pipelines.

A Prompt Templates

A.1 SLM WHY–WHAT–HOW Decomposition Prompt

```
You are a small language model that decomposes multiple-choice questions
into focused web-search queries.

--- INPUT QUESTION ---
Question:
{question}

Options:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}

--- TASK ---
Rewrite the question as three short queries that a search engine could
answer. Cover complementary aspects:

- WHY: causes, reasons, or motivations.
- WHAT: key definitions, facts, or entities.
- HOW: mechanisms, processes, or procedures.

Follow this EXACT format:

WHY: <one-sentence why-style query>
WHAT: <one-sentence what-style query>
HOW: <one-sentence how-style query>

Do NOT answer the question. Do NOT add extra text.
```

A.2 SLM Hint-Extraction Prompt

```
You are a small language model that summarizes web snippets into short,
verifiable hints for a larger QA model.

--- ORIGINAL QUESTION ---
{question}

Options:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}

--- SUB-QUERIES (WHY / WHAT / HOW) ---
{why_what_how_block}

--- RETRIEVED SNIPPETS ---
{snippets_block}

--- TASK ---
Using ONLY the information in the snippets, write 48 concise,
fact-style bullet points that would help another model answer the
question. Each bullet must:
```

```
- start with "- "
- contain exactly one atomic fact
- avoid speculation or unstated assumptions
- not mention "snippet", "search", "web", or "Google"

Do NOT answer the question directly.
Return ONLY the bullet points.
```

A.3 Gemini CoT Answer Prompt (with Hints)

```
You are an expert QA model. You will receive a multiple-choice question,
its options, and a set of hints distilled from web search.

--- QUESTION ---
{question}

Options:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}

--- HINTS (you may use or ignore these) ---
{hints_block}

--- TASK ---
1. Think step by step about the question, using the hints when useful.
2. Decide which option (A, B, C, or D) is most likely to be correct.
3. At the end, output ONLY the final letter (A, B, C, or D) on its own
   line with no explanation.

Let's reason step by step, then give the final letter.
```

A.4 Gemini Instruction-Mode Answer Prompt (with Hints)

```
You are an expert QA model. You will receive a multiple-choice question,
its options, and a set of hints distilled from web search.

--- QUESTION ---
{question}

Options:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}

--- HINTS (you may use or ignore these) ---
{hints_block}

--- TASK ---
Based on the question, options, and hints, choose the SINGLE best answer.
Return ONLY the letter A, B, C, or D with no explanation and no extra
text.
```

B Generalized Pipeline

B.1 Loading SLM

```
slm_model = None
if SLM_API_KEY:
    genai.configure(api_key=SLM_API_KEY)
    slm_model = genai.GenerativeModel(llm_name)
    print("SLM initialised.")
else:
    print("WARNING: SLM_API_KEY not set, SLM disabled.")

print("Loading local SLM (this can take a bit)...")
bnb_config = BitsAndBytesConfig(load_in_4bit=True)
slm_tokenizer = AutoTokenizer.from_pretrained(model_name)
slm_model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map=DEVICE_MAP,
    quantization_config=bnb_config
)
print("Local SLM loaded.")
```

B.2 SLM Token Statistics

```
slm_token_stats = {"prompt": 0, "output": 0, "total": 0, "calls": 0}

def _accumulate_usage(resp):
    usage = getattr(resp, "usage_metadata", None)
    if usage is not None:
        pt = getattr(usage, "prompt_token_count", 0)
        ct = getattr(usage, "candidates_token_count", 0)
        tt = getattr(usage, "total_token_count", pt + ct)
        slm_token_stats["prompt"] += pt
        slm_token_stats["output"] += ct
        slm_token_stats["total"] += tt
    slm_token_stats["calls"] += 1

def call_slm_with_cooldown(prompt, max_retries=3):
    last_exc = None
    for attempt in range(1, max_retries + 1):
        try:
            resp = slm_model.generate_content(prompt)
            _accumulate_usage(resp)
            time.sleep(BASE_SLEEP_BETWEEN_CALLS)
            return resp
        except Exception as e:
            last_exc = e
            msg = str(e)
            wait_s = None
            m1 = re.search(r"retry in ([0-9.]+)s", msg)
            if m1: wait_s = float(m1.group(1))
            else:
                m2 = re.search(r"seconds:\s*([0-9]+)", msg)
                if m2: wait_s = float(m2.group(1))
            if wait_s is None: wait_s = 60.0
            print(f"[Rate limit] Attempt {attempt}/{max_retries}     sleeping {
                wait_s:.1f}s...")
```

```
            time.sleep(wait_s)
    raise RuntimeError("SLM generate_content failed after retries.") from last_exc
```

B.3 SLM Response Generation

```
def slm_generate(prompt, max_new_tokens=128):
    inputs = slm_tokenizer(prompt, return_tensors="pt").to(slm_model.device)
    out_ids = slm_model.generate(**inputs, max_new_tokens=max_new_tokens)
    text = slm_tokenizer.decode(out_ids[0], skip_special_tokens=True)
    return text.strip()

def web_search(query, num_results=2):
    num_results = min(num_results, MAX_WEB_RESULTS)
    results = []
    if SERPER_API_KEY:
        url = "https://google.serper.dev/search"
        headers = {"X-API-KEY": SERPER_API_KEY, "Content-Type": "application/json"}
        payload = {"q": query}
        try:
            r = requests.post(url, headers=headers, json=payload, timeout=WEB_TIMEOUT)
            data = r.json()
            if data.get("organic"):
                for item in data["organic"][:num_results]:
                    snippet = (item.get("snippet") or "").replace("\n", " ")
                    link = item.get("link") or ""
                    if snippet:
                        results.append({"snippet": snippet, "url": link})
        except Exception as e:
            print("Serper error:", e)
    if not results:
        results.append({"snippet": f"General information about: {query}", "url": ""})
    return results[:num_results]
```