Think Straight, Stop Smart: Structured Reasoning for Efficient Multi-Hop RAG

Jihwan Bang **Juntae Lee** **Seunghan Yang** **Sungha Choi***
Qualcomm AI Research† Qualcomm Korea YH, Seoul, Republic of Korea
{jihwbang, juntlee, seunghan, sunghac}@qti.qualcomm.com

Abstract

Multi-hop retrieval-augmented generation (RAG) is a promising strategy for complex reasoning, yet existing iterative prompting approaches remain inefficient. They often regenerate predictable token sequences at every step and rely on stochastic stopping, leading to excessive token usage and unstable termination. We propose TSSS (**T**hink **S**traight, **S**top **S**mart), a structured multi-hop RAG framework designed for efficiency. TSSS introduces (i) a template-based reasoning that caches recurring prefixes and anchors sub-queries to the main question, reducing token generation cost while promoting stable reasoning, and (ii) a retriever-based terminator, which deterministically halts reasoning once additional sub-queries collapse into repetition. This separation of structured reasoning and termination control enables both faster inference and more reliable answers. On HotpotQA, 2WikiMultiHop, and MuSiQue, TSSS achieves state-of-the-art accuracy and competitive efficiency among RAG-CoT approaches, highlighting its effectiveness in efficiency-constrained scenarios such as on-device inference.

1 Introduction

On-device large language models (LLMs) [1–3] are increasingly appealing for privacy-preserving and latency-sensitive applications such as personal assistants, mobile knowledge agents, and offline reasoning systems [4]. Unlike server-scale LLMs, however, on-device models operate with orders of magnitude fewer parameters, which severely limits their reasoning capacity [5]. At the same time, computational efficiency is critical: every generated token incurs latency and energy cost, making it essential to reduce token generation during inference [6].

Multi-hop question answering (QA) [7–9] exemplifies these challenges. It requires combining information across multiple documents, often through iterative reasoning. Multi-hop retrieval-augmented generation (RAG) has therefore been studied extensively as a powerful strategy for improving reasoning accuracy [10–14]. However, most existing methods are designed for larger server-based models and prioritize accuracy gains over efficiency. Iterative reasoning loops frequently regenerate predictable prefixes or boilerplate sub-questions, and weak termination control often leads to duplicated queries. These inefficiencies, while tolerable in server settings, make current multi-hop RAG methods impractical for efficiency-constrained on-device inference.

Recent efforts such as EfficientRAG [15] begin to address this gap by modeling when to stop reasoning. Yet, termination alone is insufficient: as shown in Figure 1a, token usage also balloons from repetitive and predictable generation patterns, which not only increase inference cost but may also obscure the final answer. In addition, EfficientRAG requires a dedicated termination module trained on

*indicates the corresponding author.
†Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Think Straight, Stop Smart: Structured Reasoning for Efficient Multi-Hop RAG.

Figure 1: **Comparison of multi-hop reasoning processes.** (a) The baseline method (*e.g.,* Self-Ask) suffers from inefficiencies such as repetitive prefix generation and duplicated queries, which increase token generation usage and may lead to incorrect answers. (b) Our TSSS framework addresses these issues through a *structured reasoning template* that reuses recurring token sequences and anchors sub-queries to the main question, together with a *retriever-based terminator* that halts reasoning once further queries collapse into repetition. This combination reduces token usage while maintaining accuracy, resulting in faster and more reliable inference.

external data. While such training can be performed on servers, it introduces additional cost, model complexity, and potential domain dependence. By contrast, Our approach requires no additional parameter training; only heuristic template design and hyperparameter selection are performed, making it effectively training-free. This plug-and-play design makes it lightweight and directly deployable to diverse on-device models without fine-tuning, while still tackling the two fundamental obstacles in efficient multi-hop RAG: *predictable repetition* and *generator-based termination*.

To this end, we propose TSSS (**T**hink **S**traight, **S**top **S**mart), a structured multi-hop RAG framework explicitly designed for efficient on-device reasoning. TSSS introduces two key innovations: (i) *Template-based reasoning*, which encodes recurring reasoning traces as structured templates and prefills them at each hop. This systematically eliminates redundant token generation (e.g., repeated prefixes) while keeping sub-queries anchored to the main question, thereby improving both efficiency and stability. (ii) *Retriever-based terminator*, which shifts stopping control from the generator to the retriever. By detecting when newly generated queries collapse into repetition, the retriever deterministically halts the loop, preventing unnecessary duplication.

Together, these innovations enable TSSS to maintain alignment with the main question while reducing token usage. As illustrated in Figure 1b, unlike baseline methods such as Self-Ask, which waste tokens on repetitive patterns and duplicated queries, TSSS generates concise and well-structured reasoning steps, leading to faster and more reliable inference.

Our contributions are: (1) We formally identify predictable repetition and generator-based termination as fundamental efficiency bottlenecks in multi-hop RAG; (2) We introduce TSSS, combining a structured reasoning template with a retriever-based terminator to reduce token usage while maintaining reasoning accuracy; (3) We demonstrate that TSSS achieves substantial token savings alongside accuracy improvements, enabling practical multi-hop reasoning in on-device settings.

2 Methodology

2.1 Template-Based Reasoning with KV-Cache

In standard multi-hop prompting, the LLM generates entire reasoning traces in free-form natural language. This approach is token-inefficient, since recurring prefixes such as restating the main question, retrieved evidence, or scaffolding phrases are regenerated at every iteration. Without structural constraints, the model expends computation on predictable token sequences rather than focusing on the essential reasoning content.

Method	HotpotQA			2WikiMultiHop			MuSiQue		
	EM(↑)	ACC_L(↑)	T(↓)	EM(↑)	ACC_L(↑)	T(↓)	EM(↑)	ACC_L(↑)	T(↓)
No-RAG	17.7	29.1	0.1	16.6	16.6	0.2	2.6	7.2	0.2
Standard-RAG	27.7	40.1	3.3	11.3	16.2	3.7	4.1	7.6	3.3
Self-Ask [12]	25.6	37.4	15.2	21.7	31.4	13.6	8.2	14.6	18.0
Iter-RetGen [10]	30.4	43.5	27.9	12.7	17.8	29.9	5.8	10.6	27.2
IRCoT [11]	28.0	47.7	19.2	23.8	35.8	24.9	6.5	13.8	21.2
TSSS (Ours)	**34.1**	**50.9**	**8.1**	**33.6**	**42.3**	**9.0**	**14.5**	**22.8**	**9.5**

Table 1: **Overall performance comparison of TSSS with baselines.** The evaluation metrics EM and ACC_L are defined in Appendix C, and T indicates the average inference time (seconds) per sample. **Bold** marks the best score; for T, it indicates the lowest among RAG-CoT methods.

To address this inefficiency, we introduce a *template-based reasoning framework* that leverages the KV-Cache (See Appendix A for details). Each reasoning step is embedded into a fixed scaffold (gray regions in Figure 1b), where the scaffold tokens are treated as prefilled and their Key-Value states can be pre-computed and cached. As a result, the LLM only generates the variable components (green regions), such as the specific sub-query or extracted evidence. This design significantly reduces the number of newly generated tokens at each hop. By reusing the cached Key-Value states for the prefilled parts, it also lowers the computational cost for each iteration, resulting in a substantial decrease in overall latency.

Additionally, the template explicitly incorporates the main question and accumulated evidence at every iteration. By consistently anchoring sub-queries to these elements, the framework not only improves efficiency but also promotes stability in reasoning traces, mitigating the tendency of sub-queries to drift away from the original task (See Appendix B for prompt template).

2.2 Retriever-Based Terminator

A common limitation of prior iterative prompting methods (*e.g.,* Self-Ask) is the generation of near-duplicate or semantically overlapping sub-queries. Such repetition increases token usage and retrieval cost without contributing new information. To prevent this inefficiency, we introduce a *retriever-based termination rule* that deterministically decides when to halt the reasoning process.

Formally, let the current sub-query at iteration i be denoted as q_i, the main question as q_m, and the set of previously generated sub-queries as $\{q_1, \cdots, q_{i-1}\}$. We compute embedding representations $\phi(\cdot)$ for each query from the retriever, and define a similarity score:

$$\text{score}(q_i) = \max_{q \in \mathcal{Q}_i} \cos(\phi(q_i), \phi(q)), \tag{1}$$

where $\cos(\cdot, \cdot)$ denotes cosine similarity, and $\mathcal{Q}_i = \{q_m\} \cup \{q_1, \cdots, q_{i-1}\}$.

If $\text{score}(q_i) \geq \tau$, the process terminates and the model produces the final answer; otherwise, reasoning continues with the next sub-query. This retriever-based termination prunes repetitive queries early, eliminating unnecessary retrievals and token generations. Consequently, it improves efficiency (fewer iterations, fewer tokens). In our experiments, we set $\tau = 0.85$, which we found to balance between avoiding duplication and allowing sufficient reasoning depth (See Appendix D.1 for details).

3 Experiment

3.1 Implementation Details

We adopt three benchmark datasets: HotpotQA [7], 2WikiMultiHopQA [9], and MuSiQue [8]. Wikipedia passages serve as the 21M retrieval corpus for all datasets. To verify the effectiveness of TSSS, we select several baselines ranging from No-RAG to RAG-CoT (e.g., Self-Ask [12], Iter-RetGen [10], and IRCoT [11]), focusing on methods that do not require additional training cost. For fair comparison, all methods use Llama3.1-8B [1] as the generator. To reduce retrieval cost, we use the FAISS library [16] with the e5-base-v2 retriever [17], retrieving 3 documents per query. Baseline implementations are based on the open-source RAG framework FlashRAG [18]. For iterative RAG-CoT methods (e.g., Iter-RetGen, IRCoT), we set the maximum number of retrieval iterations to 10 to allow self-termination. All experiments are conducted on a single NVIDIA H100 GPU.

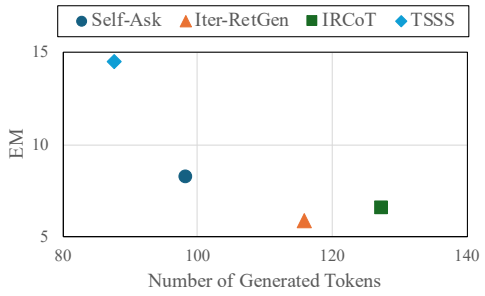

Figure 2: **Performance vs. efficiency on the MuSiQue dataset.** TSSS achieves a superior performance-efficiency trade-off, with the highest EM score and the fewest tokens.

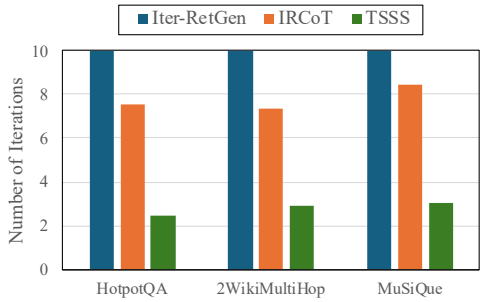

Figure 3: **Effect of retriever-based terminator.** TSSS proves efficient and stable termination mechanism by stopping fewer retrieval iterations than the other baselines.

3.2 Results

Overall performance. Table 1 summarizes the results on HotpotQA, 2WikiMultiHop, and MuSiQue. TSSS achieves the best EM and ACC_L across all three benchmarks while also maintaining competitive inference efficiency. Compared to Standard-RAG, TSSS is about 2–3 times slower (e.g., 8.1s vs. 3.3s on HotpotQA, 9.0s vs. 3.7s on 2WikiMultiHop, and 9.5s vs. 3.3s on MuSiQue), but this overhead stems from structured iterative reasoning that substantially boosts accuracy. Importantly, TSSS is significantly faster than other RAG-CoT methods. For example, it reduces inference time by more than half compared to Iter-RetGen (e.g., 9.0s vs. 29.9s on 2WikiMultiHop, 9.5s vs. 27.2s on MuSiQue) while keeping higher performance. These results highlight its key strength: *effective multi-hop reasoning with both superior accuracy and practical efficiency*.

Performance vs. efficiency. Figure 2 shows the trade-off between EM performance and the number of generated tokens on the MuSiQue dataset (See Appendix D.2 for other datasets). In this setting, methods that appear closer to the upper-left corner achieve better efficiency while maintaining strong accuracy. Compared to existing baselines, TSSS achieves the highest EM score with the fewest generated tokens. This indicates that our method is not only more accurate but also significantly more efficient in terms of token usage.

Effectiveness of retriever-based terminator. Appropriate iteration is crucial for both performance and efficiency. As shown in Figure 3, Iter-RetGen and IRCoT often require too many iterations, which leads to unnecessary overhead. In contrast, our method derives the answer with only 2–3 iterations, which can be regarded as a reasonable number. This effectiveness comes from the retriever-based terminator, which halts once sufficient context is gathered, enabling efficient reasoning without sacrificing accuracy.

4 Related Works

Retrieval-augmented generation (RAG) is a key approach for knowledge-intensive tasks but struggles with multi-hop questions. SuRe [19] proposes a framework integrating reasoning-aware retrieval and answer synthesis, while RECOMP [20] leverages query decomposition and retrieval fusion. RePlug [21] enhances single-turn RAG by incorporating structured intermediate reasoning signals during retrieval. However, these single-turn methods often fail when intermediate reasoning is required, motivating iterative approaches (*i.e.,* RAG-CoT). Self-Ask [12] decomposes queries into follow-up questions, IRCoT [11] interleaves retrieval and reasoning, and Iter-RetGen [10] introduces an iterative retrieval–generation loop for more relevant evidence.

5 Conclusion

We presented TSSS, a multi-hop RAG framework that enhances efficiency with template-based reasoning and a retriever-based terminator. It improves accuracy on HotpotQA and MuSiQue, and yields faster inference on 2WikiMultiHop with minor accuracy loss, showing its value in efficiency-constrained settings (See Appendix E for limitations).

References

[1] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *ArXiv Preprint*, 2024.

[2] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *ArXiv Preprint*, 2025.

[3] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *ArXiv Preprint*, 2024.

[4] Jihwan Bang, Juntae Lee, Kyuhong Shim, Seunghan Yang, and Simyung Chang. Crayon: Customized on-device llm via instant adapter blending and edge-server hybrid inference. In *Proceedings of the Association for Computational Linguistics*, pages 3720–3731, 2024.

[5] Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't. *ArXiv Preprint*, 2025.

[6] Juntae Lee, Jihwan Bang, Kyuhong Shim, Seunghan Yang, and Simyung Chang. Chain-of-rank: Enhancing large language models for domain-specific rag in edge device. In *Proceedings of the Nations of the Americas Chapter of the Association for Computational Linguistics*, pages 5601–5608, 2025.

[7] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018.

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

[9] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of International Conference on Computational Linguistics*, pages 6609–6625, 2020.

[10] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Proceedings of Empirical Methods in Natural Language Processing (Findings)*, pages 9248–9274, 2023.

[11] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the Association for Computational Linguistics*, pages 10014–10037, 2023.

[12] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 5687–5711, 2023.

[13] Ruofan Wu, Youngwon Lee, Fan Shu, Danmei Xu, Seung-won Hwang, Zhewei Yao, Yuxiong He, and Feng Yan. Composerag: A modular and composable rag for corpus-grounded multi-hop question answering. *ArXiv Preprint*, 2025.

[14] Zhouyu Jiang, Mengshu Sun, Lei Liang, and Zhiqiang Zhang. Retrieve, summarize, plan: Advancing multi-hop question answering with an iterative approach. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1677–1686, 2025.

[15] Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Efficientrag: Efficient retriever for multi-hop question answering. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 3392–3411, 2024.

[16] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[17] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv Preprint*, 2022.

[18] Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In *Companion Proceedings of the ACM on Web Conference*, pages 737–740, 2025.

[19] Jaehyung Kim, Jaehyun Nam, Sangwoo Mo, Jongjin Park, Sang-Woo Lee, Minjoon Seo, Jung-Woo Ha, and Jinwoo Shin. Sure: Summarizing retrievals using answer candidates for open-domain qa of llms. In *Proceedings of International Conference on Learning Representations*, 2024.

[20] Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *Proceedings of International Conference on Learning Representations*, 2024.

[21] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8364–8377, 2024.

[22] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623, 2023.

[23] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *ArXiv Preprint*, 2024.

-Supplementary Material-

Think Straight, Stop Smart: Structured Reasoning for Efficient Multi-Hop RAG

A Preliminary: KV-Cache for Efficient Decoding

In Transformer-based LLMs, auto-regressive decoding at step t requires computing the interaction between the current token and all t previously generated tokens. Let T be the total sequence length and d the model (hidden) dimension. For clarity, the following costs are per layer and omit constant factors from the number of heads.

Without KV-cache. At decoding step t, a naive implementation recomputes the entire prefix of length t. The cost is $\mathcal{O}(t\,d^2)$ for the Q, K, V projections and $\mathcal{O}(t^2 d)$ for self-attention (scores and the weighted sum). Summed over all steps, this is formulated as below.

$$\sum_{t=1}^{T} \big(\mathcal{O}(t\,d^2) + \mathcal{O}(t^2 d)\big) = \mathcal{O}(T^2 d^2) + \mathcal{O}(T^3 d),$$

i.e., cubic scaling in T due to recomputing past states at every step. (By contrast, a single full-sequence forward pass of length T costs $\mathcal{O}(Td^2 + T^2 d)$.)

With KV-cache. At step $t{+}1$, we keep the cached keys/values $K_{1:t}, V_{1:t}$ and only compute the incremental components K_{t+1}, V_{t+1}. The attention becomes

$$\mathrm{Attn}(Q_{t+1}, K_{1:t+1}, V_{1:t+1}) = \mathrm{Attn}(Q_{t+1}, [K_{1:t} \| K_{t+1}], [V_{1:t} \| V_{t+1}]),$$

where $\|$ denotes concatenation along the sequence dimension. The per-step cost is then $\mathcal{O}(d^2)$ for the new token's projections plus $\mathcal{O}(t\,d)$ for attending to t cached tokens, i.e., $\mathcal{O}(d^2 + t\,d)$. Summed over T steps, the total cost is calculated as follow.

$$\mathcal{O}(Td^2 + T^2 d).$$

Thus, the KV-cache removes the cubic term caused by recomputation, but total decoding time remains quadratic in T because each new token attends to all prior tokens.[1] The memory footprint of the KV-cache is $\mathcal{O}(T\,d)$ per layer.

B Prompt Template for `TSSS`

The following template illustrates the structured reasoning format used in `TSSS` for multi-hop question answering. In this template, {main question} represents the original user query that the reasoning process aims to answer. Each {LLM-generated Question i} is a sub-question created by the model at iteration i-th to gather additional evidence. The placeholder {Retrieved contexts with LLM-generated Question i} refers to the top-k passages retrieved from the external knowledge source (*e.g.,* Wikipedia) based on that sub-question. {LLM-generated Response i} is the model's answer to the sub-question using the retrieved contexts.

[1] If attention is restricted to a fixed window W (e.g., sliding-window attention), the attention cost becomes $\mathcal{O}(T \cdot W \cdot d)$, which is linear in T for fixed W.

Prompt template for TSSS

```
To answer the Main Question ({main question}), I propose the following additional question:
Question: {LLM-generated Question1}
{Retrieved contexts with LLM-generated Question1}
Based on the contexts, {LLM-generated Response1}

To answer the Main Question ({main question}), using the following facts:
- {LLM-generated Response1}
I propose the following additional question:
Question: {LLM-generated Question2}
{Retrieved contexts with LLM-generated Question2}
Based on the contexts, {LLM-generated Response2}

To answer the Main Question ({main question}), using the following facts:
- {LLM-generated Response1}
- {LLM-generated Response2}
I propose the following additional question:
Question: {LLM-generated Question3}
{Retrieved contexts with LLM-generated Question3}
Based on the contexts, {LLM-generated Response3}
...

Finally, based on the fact that {LLM-generated ResponseN}, the complete answer to the
main question ({main question}) is:
```

C Evaluation Metrics

During the evaluation phase, we adopt exact-match (EM) as our primary metric, which determines whether the predicted answer exactly matches the golden answer. To further refine our evaluation, we employ an LLM-as-Judge approach [22], using GPT-4o [23] as the evaluation model to assess whether the predicted answer is correct. This accuracy metric is referred to as ACC_L. The evaluation prompt is as follows.

Prompt template for LLM-as-Judge

```
Given a Question and its Golden Answer, verify whether the Predicted Answer is correct. The
prediction is correct if it fully aligns with the meaning and key information of the Golden
Answer. Respond with True if the prediction is corret and False otherwise.

Question: {question}
Golden Answer: {ground truth}
Predicted Answer: {prediction}
```

D Ablation Studies

D.1 Effect of Threshold on Retriever-based Terminator

We conduct an ablation study to examine the effect of the threshold (τ) in the retriever-based terminator. As shown in Table 2, the threshold controls the termination criteria: A larger threshold τ makes the stopping criterion stricter, since only highly similar sub-queries will be considered redundant. This leads to later termination and thus more iterations. With $\tau = 0.8$, the model halts earlier, yielding shorter inference time (*e.g.,* 6.5s on HotpotQA) but lower accuracy. In contrast, $\tau = 0.9$ permits more iterations, improving EM and ACC_L but at the cost of increased latency (*e.g.,* 12.7s on 2WikiMultiHop). The intermediate setting $\tau = 0.85$ provides a favorable balance, delivering competitive accuracy with moderate inference time. Overall, these results confirm a trade-off: stricter termination accelerates inference but limits reasoning, while looser termination enhances accuracy at the expense of efficiency. $\tau = 0.85$ emerges as a robust default across datasets.

D.2 Performance vs. Efficiency across Datasets

Figure 4 further examines the trade-off between accuracy and efficiency on HotpotQA, 2WikiMultiHop, and MuSiQue. On HotpotQA, TSSS achieves the highest EM score, while Iter-RetGen generates

τ	HotpotQA EM(↑)	HotpotQA ACC_L(↑)	HotpotQA T(↓)	2WikiMultiHop EM(↑)	2WikiMultiHop ACC_L(↑)	2WikiMultiHop T(↓)	MuSiQue EM(↑)	MuSiQue ACC_L(↑)	MuSiQue T(↓)
0.8	32.7	50.0	6.5	32.2	40.9	7.3	14.4	22.8	7.9
0.85	34.1	50.9	8.1	33.6	42.3	9.0	14.5	22.8	9.5
0.9	34.5	51.4	9.8	34.3	43.4	12.7	13.4	21.2	10.8

Table 2: **Effect of threshold (τ) on performance and inference time across datasets.** A higher threshold corresponds to looser termination criteria, allowing more iterations before answering the question. This can improve EM and ACC_L, but also increases inference time, reflecting the trade-off between accuracy and efficiency.

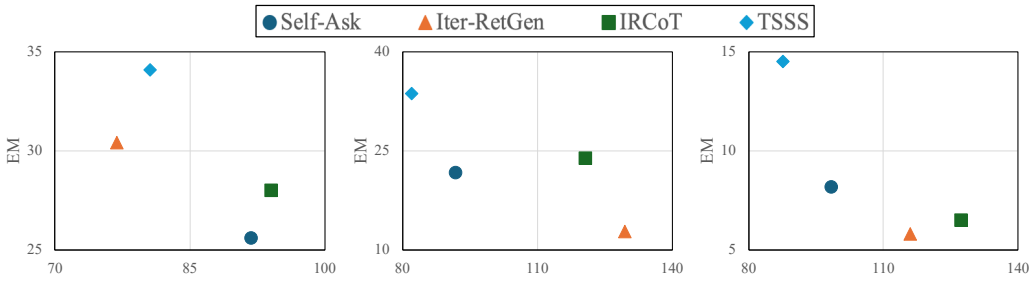

Figure 4: **Performance–efficiency trade-off of different methods on three datasets.** TSSS achieves the highest performance (EM) with fewer tokens than other baselines, demonstrating a superior balance of performance and efficiency.

fewer tokens but suffers a clear performance drop, showing that reducing token usage alone is insufficient without preserving strong reasoning ability. In contrast, on both 2WikiMultiHop and MuSiQue, TSSS simultaneously achieves the fewest generated tokens and the highest EM among all baselines, highlighting its superior balance of performance and efficiency in more complex multi-hop settings. Taken together, these results confirm that TSSS consistently delivers state-of-the-art accuracy while maintaining efficiency, offering the most favorable trade-off under resource-constrained settings.

E Limitations & Future Work

While our framework alleviates key inefficiencies in multi-hop RAG, the termination mechanism still depends on heuristic redundancy detection, which may not capture all cases where additional reasoning is necessary. A promising direction is to train LLMs to self-terminate, enabling more reliable and adaptive stopping. Beyond this, an exciting avenue for future work is the development of *adaptive templates* that extend the benefits of our structured reasoning beyond multi-hop QA. Such templates could dynamically adjust to diverse tasks while maintaining efficiency, offering the potential to preserve high accuracy under on-device constraints by further reducing the computational cost of reasoning. This direction highlights the broader vision of enabling lightweight yet powerful reasoning systems deployable across a wide range of real-world scenarios.