MergePRAG: Orthogonal Merging of Passage-experts for Multi-hop Parametric RAG

Xuebing Liu[1] , Shanbao Qiao[1] , Roseline Nyange[1] , Dongwook Min[2] , Hyun Kim[3] , Seung-Hoon Na[2*]

1. *Department of Computer Science and Artificial Intelligence, Jeonbuk National University*
2. *Graduate School of Artificial Intelligence, Ulsan National Institute of Science and Technology*
3. *Electronics and Telecommunications Research Institute*

ICLR

Background and Motivation

- Retrieval-augmented generation (RAG): retrieved passages as prompts
- parametric retrieval-augmented generation (PRAG): retrieved passages as parameter updates

Limitation:

- RAG cannot fully utilize retrieved content.
- Existing PRAG is limited to single-hop retrieval.
- Existing PRAG methods require complex data preparation.

Method Overview

- Generate sub-question
- Retrieve passages
- Parameterize passages
- Inner merge
- Continual merge
- Answer & continue

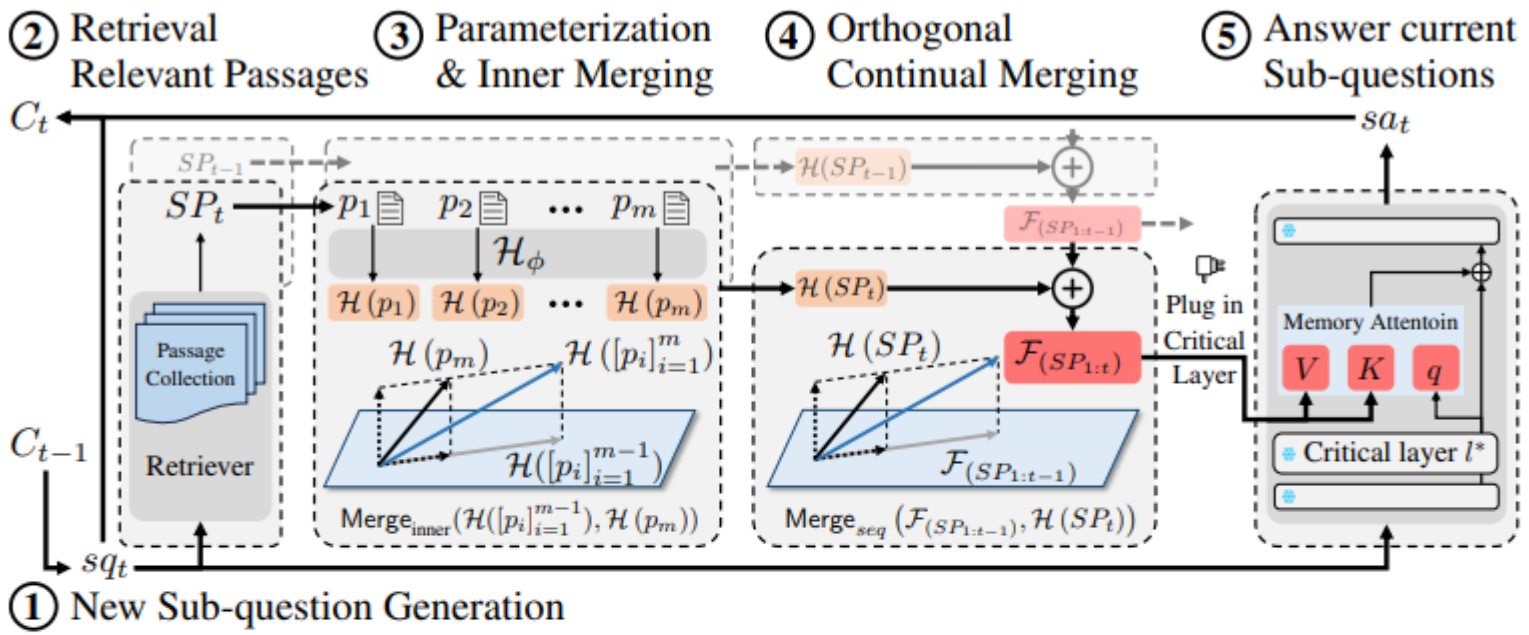


Figure 1: Overview of MergePRAG for multi-hop QA. A complex query is decomposed into sub-questions, and retrieved passages are sequentially incorporated through parameterization and continual merging. At each timestep t: (1) a sub-question sq_t is generated from the reasoning chain C_{t-1} (Eq. 1, Section 3.1); (2) the retriever returns top-ranked passages $SP_t \subseteq \mathcal{R}$; (3) given $SP_t = [p_i]_{i=1}^m$, each passage is parameterized by the hypernetwork to produce $\{\mathcal{H}_\phi(p_i)\}_{i=1}^m$, which are combined into $\mathcal{H}_\phi(SP_t)$ via the inner-merging mechanism (Eq. 6, Section 3.2); (4) orthogonal continual merging updates the accumulated parameters $\mathcal{F}(SP_{1:t-1})$ with $\mathcal{H}_\phi(SP_t)$ to obtain $\mathcal{F}(SP_{1:t})$ (Eq. 11, Section 3.2.2); and (5) the merged expert $\mathcal{F}(SP_{1:t})$ is injected into the base LLM $\mathcal{M}_{\theta_0}$ at the critical layer l^* to generate the sub-answer (Eqs. 4–5). This process repeats until no further sub-questions are produced, after which the final answer is generated.

Passage Parameterization

Passage → Hypernetwork → (K_p, V_p) memory

Lightweight passage expert

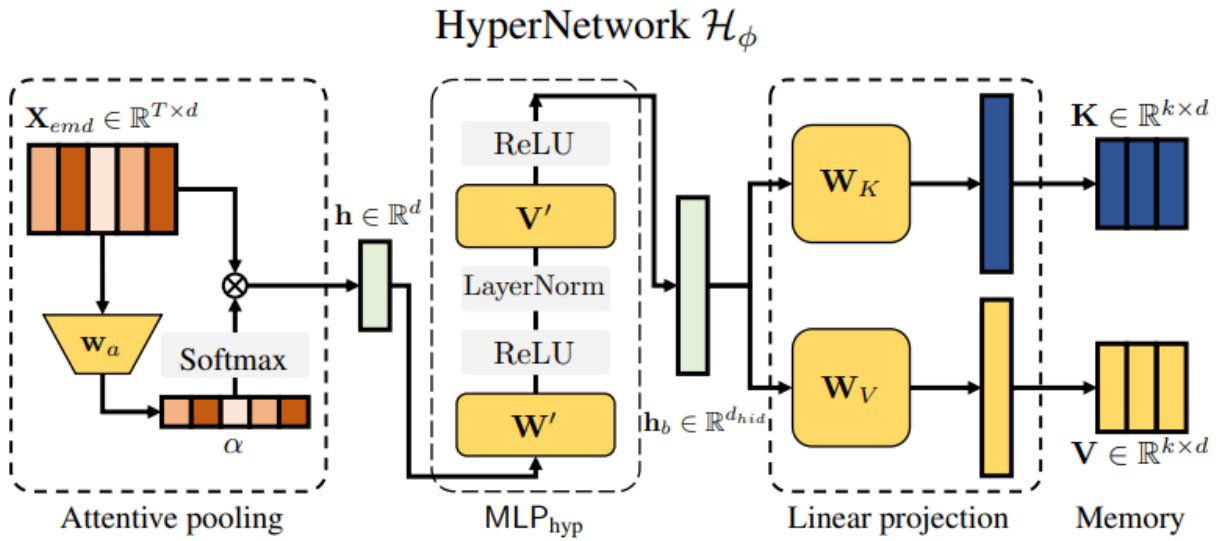


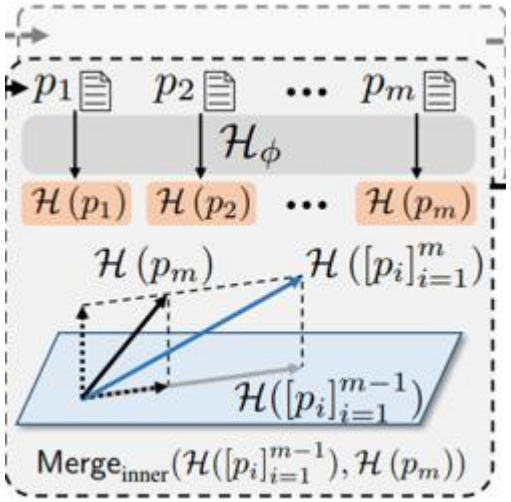
Figure 5: The hypernetwork $\mathcal{H}_\phi(p)$ generates passage-specific key–value vectors $\mathbf{K}_p, \mathbf{V}_p \in \mathbb{R}^{k\times d_{\text{model}}}$, referred to as passage-specific memory, which serve as lightweight, plug-in passage-level experts for downstream reasoning. The process consists of three stages: (1) **attentive pooling**, (2) **MLP**, and (3) **linear projection**. *1) Attentive pooling.* Given a one-hot token matrix $\mathbf{X} \in \mathbb{R}^{T\times|\mathcal{V}|}$ for passage p, the model first converts it into a sequence of embeddings $\mathbf{X}_{\text{emb}} \in \mathbb{R}^{T\times d}$ via the word embedding layer: $\mathbf{X}_{\text{emb}} = \text{Embedding}(\mathbf{X})$ (Eq. 19). Attention is then applied over the token embeddings, where a learnable vector $\mathbf{w}_a \in \mathbb{R}^d$ serves as the query: $\text{Emd}(p) = \mathbf{h} \in \mathbb{R}^d$ (Eq. 20). *2) MLP.* The pooled representation $\mathbf{h}$ is passed through a two-layer feedforward network with ReLU activations and LayerNorm, producing a latent representation $\mathbf{h}_b$ (Eq. 21). *3) Linear projection.* Two independent linear projection heads map $\mathbf{h}_b$ into the key and value parameter spaces: $\mathbf{K}_p, \mathbf{V}_p \in \mathbb{R}^{k\times d_{\text{model}}}$, yielding flattened key–value memory vectors of length $k \cdot d_{\text{model}}$ for passage p (Eq. 22). The resulting passage-specific memory is subsequently injected into the target model as additional knowledge signals.

$$\mathcal{H}_\phi(p) = (K_p, V_p)$$

Merging Mechanism

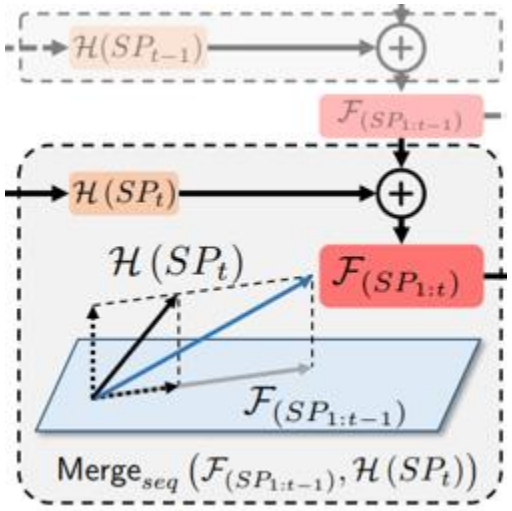
- Within-hop: $\text{Merge}_{\text{inner}}$
- Across-hop: $\text{Merge}_{\text{seq}}$
- Goal: accumulate complementary knowledge

$\text{Merge}_{\text{inner}}$



$$\begin{aligned}\mathcal{H}([p_i]_{i=1}^{m}) &= \text{Merge}_{\text{inner}}(\mathcal{H}_\phi(p_1), \ldots, \mathcal{H}_\phi(p_m)) \\ &= \text{Merge}_{\text{inner}}\left(\mathcal{H}([p_i]_{i=1}^{m-1}), \mathcal{H}(p_m)\right)\end{aligned}$$

$\text{Merge}_{\text{seq}}$



$$\mathcal{F}_{(SP_{1:t})} = \text{Merge}_{seq}\left(\mathcal{F}_{(SP_{1:t-1})}, \mathcal{H}_\phi(SP_t)\right).$$

Orthogonal Continual Merging

- Keep only the orthogonal residual
- Preserve previous knowledge

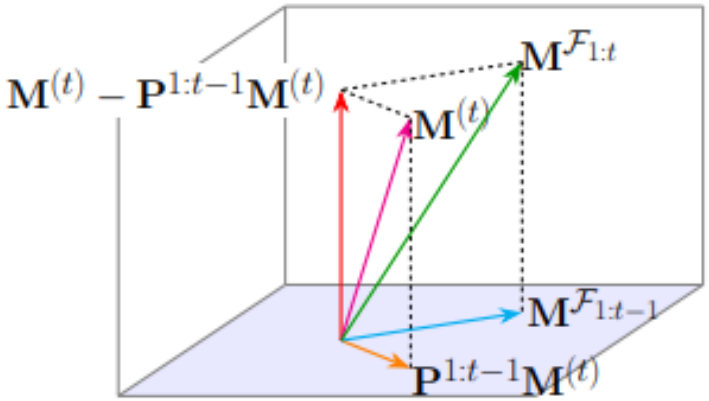
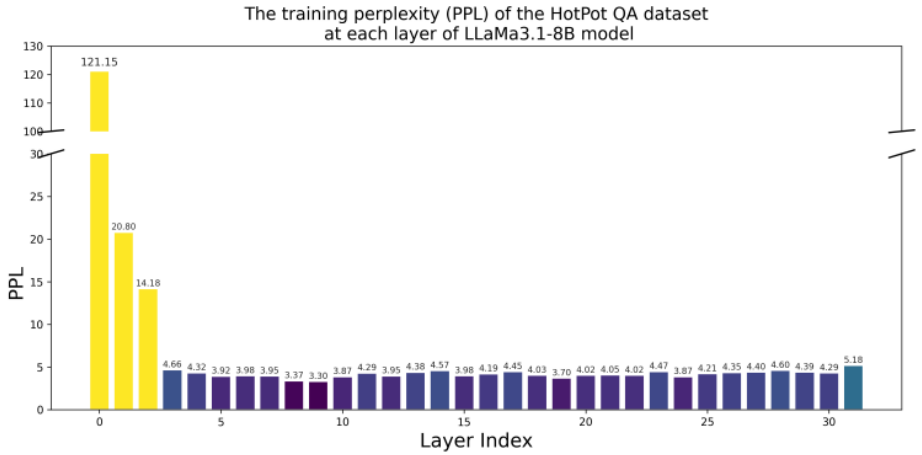


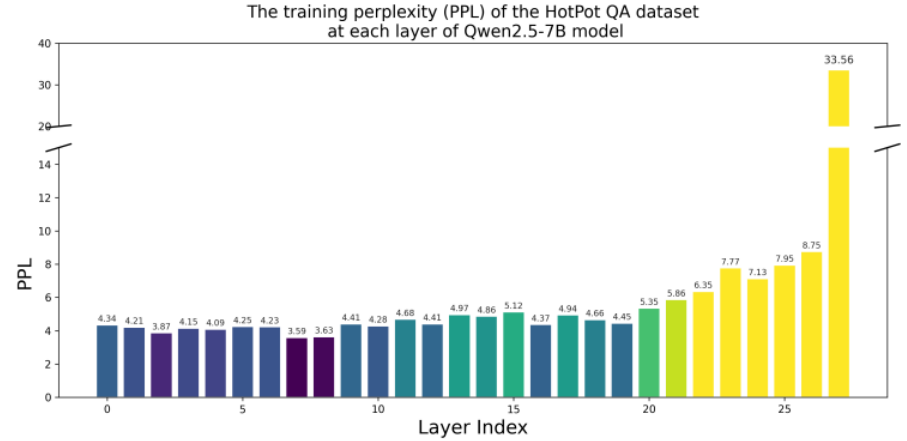
Figure 4: Illustration of orthogonal continual merging based on Gram–Schmidt procedure.

$$\mathbf{M}^{\mathcal{F}_{1:t}} = \mathbf{M}^{\mathcal{F}_{1:t-1}} + \left(\mathbf{I} - \mathbf{P}^{1:t-1}\right)\mathbf{M}^{(t)},$$

project new expert onto previous subspace, then keep the residual

Critical-layer Parameterization





- Inject at one critical layer
- Selected by layer-wise scanning

Model	HotpotQA	2WikiMultihopQA	MuSiQue
LLaMA3.1-8B	$l^* = 9$	$l^* = 7$	$l^* = 8$
Qwen2.5-7B	$l^* = 7$	$l^* = 8$	$l^* = 9$

Table 8: Selected critical layers l^* for passage-vector insertion based on layer-wise perplexity analysis across datasets.

Experimental Setup

Category	Content
Models	LLaMA3.1-8B, Qwen2.5-7B
Retrievers	E5, BM25, Contriever
Datasets	HotpotQA, 2WikiMultihopQA, MuSiQue, MQuAKE
Metrics	EM, F1
Baselines	RAG, PRAG, IRCoT, DeepRAG, R3-RAG, etc.

Main Results

Model	Retriever	Method	*HotpotQA* EM	*HotpotQA* F1	*2WikiMhQA* EM	*2WikiMhQA* F1	*MuSiQue* EM	*MuSiQue* F1
LLaMA3.1-8B	E5	RAG$_{\|SP\|=1}$	21.60	36.67	4.90	17.36	2.00	11.49
	E5	RAG$_{\|SP\|=4}$	27.80	40.51	5.10	15.80	2.70	11.27
	E5	RAG-CoT$_{\|SP\|=1}$	37.60	45.15	30.90	35.00	5.60	13.38
	E5	RAG-CoT$_{\|SP\|=4}$	43.70	50.41	36.20	40.00	5.90	12.49
	E5	IRCoT†	39.30	46.00	35.10	37.50	12.00	13.60
	E5	FLARE†	17.80	20.90	10.90	11.40	2.30	2.80
	E5	R3-RAG†	45.60	58.80	52.90	60.90	**21.20**	**32.80**
	BM25	R3-RAG†	44.40	57.60	50.60	58.60	17.20	27.70
	BM25	Search-o1†	14.80	24.08	22.20	27.10	5.40	11.98
	BM25	Auto-RAG†	25.80	36.09	23.00	30.09	-	-
	BM25	DeepRAG†	40.70	51.54	48.10	53.25	-	-
	BM25	PRAG†	-	44.84	-	40.55	-	-
	BM25	DyPRAG†	-	38.35	-	50.24	-	-
	E5	**MergePRAG +**$_{\|SP\|=1}$	48.80	55.53	66.30	71.05	14.40	25.04
	E5	**MergePRAG +**$_{\|SP\|=4}$	**52.40**	**60.67**	**73.20**	**79.34**	16.70	27.69
	BM25	**MergePRAG +**$_{\|SP\|=1}$	46.80	53.40	61.10	67.31	17.80	29.39
	BM25	**MergePRAG +**$_{\|SP\|=4}$	52.40	60.58	70.20	76.65	20.30	31.20
Qwen2.5-7B	E5	RAG$_{\|SP\|=1}$	36.60	43.37	34.90	37.36	3.20	8.71
	E5	RAG$_{\|SP\|=4}$	45.30	52.08	42.00	44.49	5.80	12.73
	E5	RAG-CoT$_{\|SP\|=1}$	30.20	36.20	19.10	23.05	4.30	8.30
	E5	RAG-CoT$_{\|SP\|=4}$	44.60	51.28	35.40	37.79	5.20	9.55
	E5	IRCoT†	35.70	41.10	31.10	33.50	9.40	11.20
	E5	FLARE†	23.40	32.06	21.80	26.51	3.60	4.80
	E5	R3-RAG†	46.40	**59.70**	54.20	62.70	**21.40**	**34.00**
	BM25	R3-RAG†	44.90	58.20	52.80	61.10	17.60	30.00
	BM25	Search-o1†	11.60	16.95	22.00	25.02	2.10	7.48
	BM25	DeepRAG†	32.10	41.14	40.40	44.87	-	-
	E5	**MergePRAG +**$_{\|SP\|=1}$	43.40	50.64	65.80	69.72	9.70	19.61
	E5	**MergePRAG +**$_{\|SP\|=4}$	50.80	58.37	**77.40**	**81.49**	12.30	21.57
	BM25	**MergePRAG +**$_{\|SP\|=1}$	42.00	49.09	59.70	63.05	13.00	23.35
	BM25	**MergePRAG +**$_{\|SP\|=4}$	**51.40**	59.33	71.80	76.06	16.70	27.33

MergePRAG consistently outperforms strong RAG / PRAG baselines

Efficiency and Takeaways

- Orthogonal merging performs best
- Better knowledge accumulation across hops
- MergePRAG is both effective and practical

	Generate Passage Memory	Response sa	Average Time
RAG$_{\|SP\|=1}$	-	-	0.712s
RAG-CoT$_{\|SP\|=1}$	-	-	6.389s
MergePRAG+$_{\|SP\|=1}$	0.001s	0.259s	2.517s

Table 11: Efficiency analysis of MergePRAG using the LLaMA3.1-8B model on the HotpotQA dataset.

MergePRAG+ / *HotpotQA*		
Merge$_{seq}$	EM	F1
▲	36.20	43.47
●	48.20	55.04
♦	46.40	54.07
▼	48.20	54.95
■	48.80	55.53

Table 5: Performance comparison between different merging methods for Merge$_{seq}$ under the setting of $|SP| = 1$: ▲: Additive merging, ●: Arithmetic mean merging, ♦: TIES merging, ▼: Concat merging, ■: Gram–Schmidt orthogonalization merging.

Thanks!

Paper

Code