Doc2Weight: In-Weights Retrieval-Augmented Generation via Parameter Injection

Anonymous ACL submission

Abstract

Retrieval-augmented generation (RAG) typically incorporates external knowledge by appending retrieved documents to the input prompt. While effective for knowledge-intensive tasks, this in-context formulation increases memory use and serving cost as the retrieved context grows. Motivated by this limitation, we propose Doc2Weight, an in-weights RAG framework that injects retrieved information into model parameters rather than the textual context. Doc2Weight converts retrieved documents into parameter-efficient low-rank updates and fuses them in weight space according to their relevance to the query. This enables retrieval augmentation with a query-only prompt, decoupling retrieval depth from prompt length while allowing multiple retrieved passages to influence generation within a single forward pass. We evaluate Doc2Weight on instruction-tuned language models of different scales, including LLaMA-3.2-1B, Qwen-2.5-1.5B, and LLaMA-3-8B, across major QA benchmarks. Experiments show that Doc2Weight consistently outperforms standard in-context RAG and prior in-weights RAG baselines, while ablations validate the importance of query-conditioned fusion.

1 Introduction

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) is a widely adopted paradigm for enhancing large language models (LLMs) with external knowledge. In standard RAG, documents relevant to a given query are retrieved from an external source and appended to the input prompt, allowing the model to generate responses conditioned on both the query and the retrieved evidence. By grounding generation in external documents, RAG substantially improves performance on knowledge-intensive tasks such as open-domain question answering and multi-hop reasoning, where models must access information beyond what is stored in their parametric knowledge.

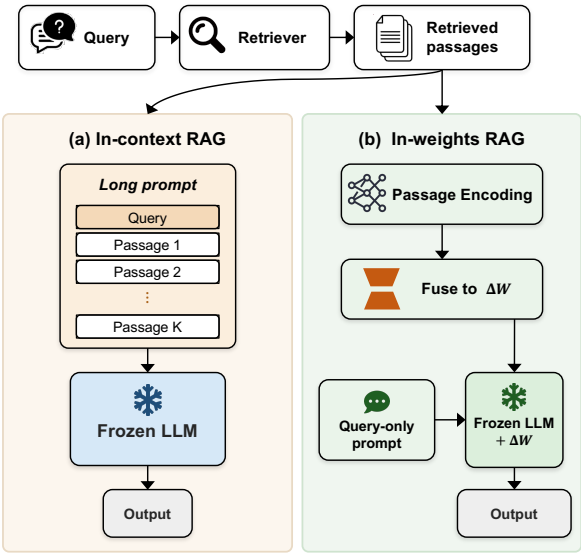

Figure 1: **(a)** Standard in-context RAG concatenates all retrieved passages with the query, increasing prompt length and inference cost. **(b)** In-weights RAG maps the retrieved passages into the parameter space as a weight update (ΔW) and conditions generation on the query alone, avoiding prompt expansion.

As LLMs improve in scale and reasoning (Grattafiori et al., 2024; Liu et al., 2024; Rastogi et al., 2025; Yang et al., 2025), they have become increasingly effective at using retrieved information. This has made RAG a core component of many real-world systems, including conversational agents (Achiam et al., 2023; Team et al., 2023) that answer user queries by referencing web pages, private documents, or external knowledge bases. Despite its effectiveness, standard RAG has a fundamental efficiency limitation: retrieved documents must be processed as part of the textual context. As the number or length of retrieved passages grows, the input context becomes longer, increasing memory usage, context-processing cost, and serving overhead. This makes retrieval aug-

mentation expensive in settings with limited memory budgets, fixed context windows, or throughput-sensitive serving requirements.

To mitigate this limitation, recent work has explored alternatives that avoid directly appending retrieved documents to the prompt. Instead, retrieved information is transformed into parameter-space representations, adapters, or passage-conditioned modules that influence a frozen language model without requiring the documents to be processed as input tokens (Liu et al., 2025; Chen et al., 2025; Tan et al., 2025). This line of work has often been referred to as parametric RAG (Tan et al., 2025); in this paper, we use the term **in-weights RAG** to emphasize its contrast with standard in-context RAG. The central promise of in-weights RAG is to decouple retrieval augmentation from textual prompt expansion: retrieved documents can influence generation through model weights rather than through additional input tokens.

However, existing in-weights RAG methods still face important challenges in how retrieved documents are represented and composed. DyPRAG (Tan et al., 2025) learns a lightweight translator that maps document representations into LoRA-style parametric updates, reducing the need to store separately trained document-specific parameters. While effective, each document is represented by a compressed final-token hidden state and converted into a parametric update independently, which may limit both token-level document expressiveness and explicit cross-document composition. Generative Adapter (Chen et al., 2025) provides a more expressive alternative by generating adapters from second-order statistics over context hidden states, allowing information from all context tokens to contribute to the generated update. However, such context-to-parameter mappings still do not explicitly address a key issue in multi-document retrieval: for a given query, different retrieved passages should not contribute equally to the final update.

Intuitively, when multiple passages are retrieved, their usefulness depends on the input query. A passage that is central evidence for one query may be irrelevant, redundant, or even distracting for another. Thus, effective in-weights RAG requires not only expressive document parameterization, but also query-aware composition of multiple retrieved documents in weight space. Rather than compressing retrieved documents into a query-agnostic update, the model should selectively emphasize passages that are more relevant to the current query.

Motivated by this observation, we propose **Doc2Weight**, an in-weights RAG framework that integrates retrieved documents into LLMs through query-conditioned, parameter-efficient weight updates. Doc2Weight first encodes each retrieved passage into a compact second-order signature derived from its hidden states, and then fuses these signatures in weight space according to their relevance to the input query. The fused representation is mapped into a single low-rank update and injected into the frozen backbone model during inference. As a result, Doc2Weight enables retrieval augmentation without expanding the textual prompt, while allowing multiple retrieved passages to influence generation through a single fused parameter update.

We evaluate Doc2Weight on a diverse set of question answering benchmarks using instruction-tuned language models of different scales, including LLaMA-3.2-1B, Qwen-2.5-1.5B, and LLaMA-3-8B. Across backbones and QA benchmarks, Doc2Weight provides consistent improvements over standard in-context RAG and prior in-weights RAG baselines, demonstrating that retrieved evidence can be effectively integrated through query-aware weight-space adaptation without appending retrieved passages to the prompt.

Further analyses validate the role of query-conditioned fusion and measure the prompt-length and throughput trade-offs of weight-space retrieval augmentation.

Our contributions are summarized as follows:

- We introduce **Doc2Weight**, an in-weights RAG framework that injects retrieved evidence through parameter-efficient low-rank updates, avoiding prompt expansion during inference.

- We propose query-conditioned weight-space fusion, which selectively composes multiple retrieved passages into a single parameter update based on their relevance to the input query.

- We show that Doc2Weight consistently outperforms standard in-context RAG and prior in-weights RAG baselines across multiple backbones and QA benchmarks, and validate the importance of query-conditioned fusion through controlled ablations.

2 Related Work

2.1 Retrieval Augmentation: In Context vs. In Weights

Retrieval-augmented generation (RAG) enhances language models by retrieving external evidence and conditioning generation on the retrieved content (Lewis et al., 2020; Guu et al., 2020). Most RAG methods follow an *in-context* formulation, where the retrieved passages $d_{1:k}$ are appended to the query q and processed as part of the input:

$$p_\theta(y \mid q, d_{1:k}). \tag{1}$$

This paradigm is simple and broadly effective for knowledge-intensive tasks, and prior work has refined how retrieved passages are selected, read, and combined (Borgeaud et al., 2022; Izacard and Grave, 2021; Lin et al., 2024; Asai et al., 2023; Jiang et al., 2023; Su et al., 2024). However, because retrieved documents remain part of the textual context, in-context RAG couples retrieval depth to prompt length, memory usage, and context-processing cost.

An alternative direction is *in-weights RAG*, also referred to as parametric RAG, where retrieved documents influence generation through model parameters rather than additional input tokens:

$$p_{\theta+\Delta_\phi(d_{1:k})}(y \mid q). \tag{2}$$

Here, $\Delta_\phi(d_{1:k})$ denotes an instance-specific parameter update generated from the retrieved passages. This formulation decouples retrieval augmentation from textual prompt expansion, while introducing new challenges in how retrieved content should be represented, converted into parameter updates, and composed across multiple passages. Doc2Weight further extends this view by making the update explicitly query-conditioned, i.e., $\Delta_\phi(q, d_{1:k})$, so that different passages can contribute differently depending on the input query.

2.2 Parameter-Space Adaptation and Composition

Recent work has explored several ways to inject retrieved or contextual information into model parameters. Parametric RAG (PRAG) learns document-specific LoRA parameters offline, replacing retrieved text with loadable parameter updates at inference time (Su et al., 2025). This reduces reliance on long prompts, but requires training and storing document-specific parameters. Dynamic Parametric RAG (DyPRAG) reduces this cost by learning a lightweight parameter translator that maps document representations into LoRA-style updates at test time (Tan et al., 2025). Relatedly, GenPoE constructs passage-conditioned experts and combines them for knowledge enhancement, highlighting the importance of multi-passage composition in parameter space (Liu et al., 2025). Generative Adapter, inspired by hypernetworks and fast weights, provides a more general context-to-parameter framework by generating low-rank adapters from second-order statistics over context hidden states (Ha et al., 2016; Ba et al., 2016; Schlag et al., 2021; Chen et al., 2025).

These methods demonstrate that external text can be converted into parameter-space updates, but they leave a retrieval-specific composition problem largely open: when multiple passages are retrieved for a query, the final update should depend not only on the documents themselves, but also on which passages are relevant, redundant, or distracting for the current query. Doc2Weight directly targets this gap by constructing passage-level signatures and composing them through query-conditioned weight-space fusion.

Doc2Weight is also related to parameter-efficient adaptation (Houlsby et al., 2019; Hu et al., 2022) and modular composition methods that combine learned updates in weight space (Pfeiffer et al., 2021; Ilharco et al., 2023). Unlike task- or domain-level composition, Doc2Weight performs instance-level composition over retrieved passages, producing a single query-conditioned update per input.

3 Method

3.1 Overview

Given a query q and a set of retrieved passages $\mathcal{D}(q) = \{d_i\}_{i=1}^K$, our goal is to generate an answer token sequence $y = (y_1, \ldots, y_{T_y})$. Standard in-context RAG conditions the model on retrieved passages by appending them to the textual prompt. Doc2Weight instead keeps the textual prompt query-only and uses the retrieved passages to generate a temporary parameter update for the current input. That is, the update is *instance-specific*: it is constructed on the fly from the current query and retrieved passages, and does not modify the frozen backbone permanently.

Formally, given a frozen language model p_θ, Doc2Weight constructs a query-conditioned update $\Delta_\Phi(q, \mathcal{D})$ and generates from

$$y \sim p_{\theta+\Delta_\Phi(q,\mathcal{D})}(\cdot \mid q), \tag{3}$$

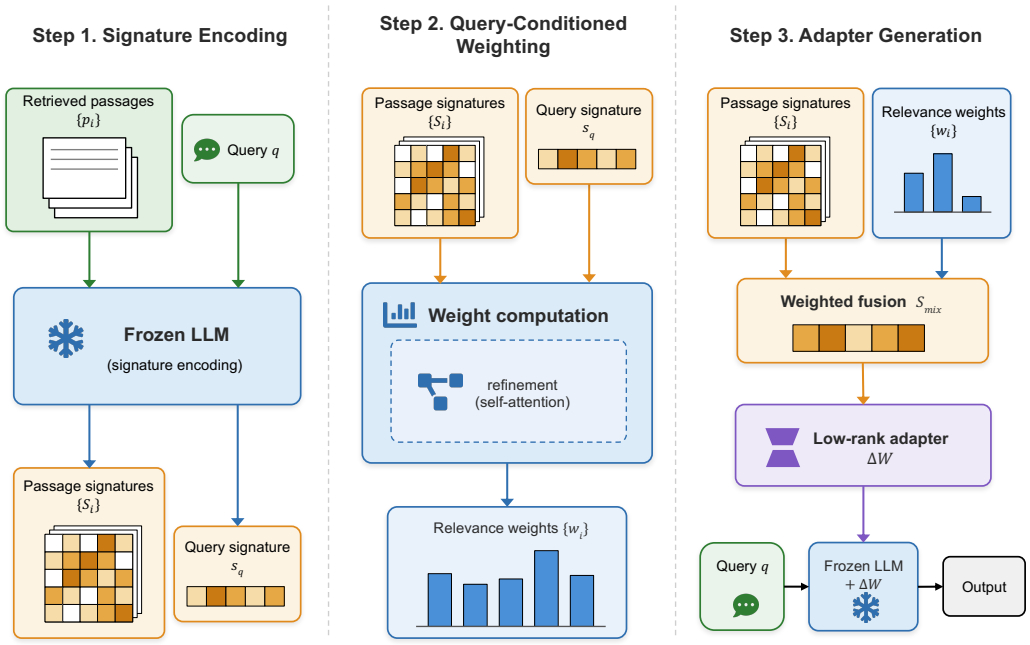

Figure 2: Overview of our method. At the first step, a shared frozen LLM encodes both the retrieved passages and the query into compact *signatures*. Next, query-conditioned relevance weights are computed over the passage signatures and used to fuse them into a single signature. Finally, the fused signature is mapped to a low-rank adapter injected into the frozen LLM, which generates from the query alone; retrieval influences generation only through the parameter-space adapter.

where Φ denotes the trainable parameters used to generate the update.

As shown in Figure 2, Doc2Weight constructs $\Delta_\Phi(q, \mathcal{D})$ in three stages. First, each retrieved passage is encoded by the frozen backbone, and its token-level hidden states are summarized into a compact second-order matrix representation, which we call a passage signature. Second, the query and passage signatures are jointly refined to compute query-conditioned fusion weights over the retrieved passages, determining which passages should contribute more strongly for the current query. Third, the fused signature is mapped into LoRA-style low-rank updates and injected into selected linear projections of the frozen model.

At generation time, the model receives only the query tokens as textual input, while retrieved passages influence generation through the constructed update $\Delta_\Phi(q, \mathcal{D})$.

3.2 Passage-to-Signature Encoding

The first stage converts each retrieved passage into a compact descriptor used for fusion and weight generation. Motivated by Generative Adapter (Chen et al., 2025), Doc2Weight represents a passage through token-level hidden-state statistics rather than a single document embedding (Tan et al., 2025). This allows the representation to aggregate information across all passage tokens while remaining compact enough to be mapped into low-rank parameter updates. We refer to the resulting normalized Gram-based descriptor as a *passage signature*.

Hidden-state features. For each retrieved passage d_i, we encode the passage with the frozen backbone language model and obtain token-level hidden states

$$\mathbf{H}_i \in \mathbb{R}^{N_i \times d}, \tag{4}$$

where N_i is the number of tokens in passage d_i and d is the hidden size.

Gram matrix summary. We project the hidden states into a low-dimensional rank space using a trainable projection matrix $\mathbf{R} \in \mathbb{R}^{d \times r}$, while keeping the backbone encoder frozen:

$$\mathbf{U}_i = \mathbf{H}_i \mathbf{R} \in \mathbb{R}^{N_i \times r}. \tag{5}$$

We then compute a length-normalized Gram matrix:

$$\mathbf{G}_i = \frac{1}{N_i} \mathbf{U}_i^\top \mathbf{U}_i \in \mathbb{R}^{r \times r}. \tag{6}$$

This Gram matrix is the second-order statistic used by Doc2Weight: it summarizes average feature co-activation patterns across passage tokens, rather

than only first-order information such as a pooled hidden vector. The normalization by N_i makes the summary less sensitive to passage length.

Normalization and passage signature. Because Gram matrices can vary substantially in scale, we normalize $\mathbf{G}_i$ before using it for fusion and weight generation. We first apply a regularized whitening transform and then use SVD-based normalization for scale stabilization:

$$\begin{aligned} \bar{\mathbf{G}}_i &= (\mathbf{G}_i + \varepsilon \mathbf{I})^{-1/2} \mathbf{G}_i \\ &\quad (\mathbf{G}_i + \varepsilon \mathbf{I})^{-1/2}, \\ \mathbf{S}_i &= \mathrm{svdnorm}(\bar{\mathbf{G}}_i). \end{aligned} \tag{7}$$

Here, $\mathrm{svdnorm}(\cdot)$ denotes SVD-based normalization that removes singular-value magnitudes while preserving the singular-vector orientation. We use $\mathbf{S}_i \in \mathbb{R}^{r \times r}$ as the matrix-form passage signature.

We also define its vectorized form by flattening all entries of $\mathbf{S}_i$ in a fixed order:

$$\mathbf{s}_i = \mathrm{norm}(\mathrm{vec}(\mathbf{S}_i)) \,. \tag{8}$$

Here, $\mathrm{vec}(\cdot)$ denotes full-matrix vectorization and $\mathrm{norm}(\cdot)$ denotes ℓ_2 normalization. Thus, each passage is represented by a matrix-form signature $\mathbf{S}_i \in \mathbb{R}^{r \times r}$ and a vector-form signature $\mathbf{s}_i \in \mathbb{R}^{r^2}$.

3.3 Query-Conditioned Signature Fusion

The second stage determines how retrieved passages contribute to the final update. Since the usefulness of a passage depends on the input query, retrieved passages should not contribute uniformly. Doc2Weight therefore computes query-conditioned fusion weights over passage signatures, allowing the model to emphasize relevant passages while reducing the influence of redundant or distracting ones.

We encode the query using the same signature construction procedure as in Section 3.2, obtaining a query signature $\mathbf{s}_q$. We then initialize a sequence containing the query and passage signatures, where $D = r^2$:

$$\mathbf{Z} = [\mathbf{s}_q; \mathbf{s}_1; \ldots; \mathbf{s}_K] \in \mathbb{R}^{(K+1) \times D}. \tag{9}$$

To model query–passage relevance together with interactions among retrieved passages, we apply a single parameter-free self-attention-style refinement over the signatures:

$$\begin{aligned} \mathbf{A} &= \mathrm{softmax}_{\mathrm{row}}\left(\frac{\mathbf{Z}\mathbf{Z}^\top}{\sqrt{D}\,\tau_a} \right), \\ \widetilde{\mathbf{Z}} &= \mathrm{LN}\big(\tfrac{1}{2}\mathbf{Z} + \tfrac{1}{2}\mathbf{A}\mathbf{Z}\big), \end{aligned} \tag{10}$$

where τ_a is a temperature parameter and LN denotes layer normalization. Because refinement is performed over the joint set of query and passage signatures, it captures both query–passage relevance and passage–passage interactions. After refinement, let $\widetilde{\mathbf{z}}_q$ denote the refined query representation and $\widetilde{\mathbf{z}}_i$ denote the refined representation of passage d_i. We compute passage scores and normalize them into fusion weights:

$$a_i = \frac{\widetilde{\mathbf{z}}_q \cdot \widetilde{\mathbf{z}}_i}{\sqrt{D}\tau}, \quad w_i = \frac{\exp(a_i)}{\sum_{j=1}^{K} \exp(a_j)}. \tag{11}$$

The final weights are therefore query-conditioned, but they are computed from passage representations that have already interacted with the other retrieved passages, allowing the weights to reflect redundancy and complementarity among evidence.

The final query-conditioned signature is obtained by fusing the matrix signatures in rank space:

$$\mathbf{S}_{\mathrm{mix}} = \sum_{i=1}^{K} w_i \mathbf{S}_i \in \mathbb{R}^{r \times r}. \tag{12}$$

The resulting $\mathbf{S}_{\mathrm{mix}}$ serves as a compact summary of the retrieved evidence, weighted according to its relevance to the current query.

3.4 Signature-to-Weight Generation

The fused signature is not itself an adapter; it is a retrieval-dependent descriptor from which Doc2Weight generates LoRA-style weight updates. This stage converts the query-conditioned evidence summary into parameter updates compatible with the frozen backbone.

Let $\mathcal{M}$ denote the set of target linear modules. For each module $m \in \mathcal{M}$ with frozen weight matrix $\mathbf{W}_m$, let $\ell(m)$ denote its layer. Doc2Weight generates a low-rank factor from the fused signature:

$$\mathbf{B}_{\ell(m)} = \mathbf{S}_{\mathrm{mix}} \mathbf{C}_{\ell(m)}, \tag{13}$$

where $\mathbf{C}_{\ell(m)}$ is a learned matrix shared across instances and chosen to match the input dimension of module m. The module update is then formed as

$$\Delta \mathbf{W}_m = \mathbf{A}_m \mathbf{B}_{\ell(m)}, \tag{14}$$

where $\mathbf{A}_m$ is a learned module-specific matrix. This yields a LoRA-style low-rank update for each selected module while keeping the backbone parameters fixed.

Overall, the update injected into the backbone is

$$\Delta_\Phi(q, \mathcal{D}) = \{\Delta \mathbf{W}_m\}_{m \in \mathcal{M}}. \tag{15}$$

3.5 Training Objective

Doc2Weight keeps the pretrained backbone frozen and trains only the lightweight parameters used to construct instance-specific updates. The trainable parameters are

$$\Phi = \{\mathbf{R}\} \cup \{\mathbf{A}_m\}_{m\in\mathcal{M}} \cup \{\mathbf{C}_\ell\}_{\ell=1}^{L}, \quad (16)$$

where $\mathbf{R}$ is the trainable projection parameter introduced in Section 3.2, and $\mathbf{A}_m$ and $\mathbf{C}_\ell$ are the low-rank update-generation parameters introduced in Section 3.4.

Query-only training. During training, the textual input contains only the query. Retrieved passages are used to construct the parameter update $\Delta_\Phi(q, \mathcal{D})$, but are not appended to the prompt. We optimize the negative log-likelihood over answer tokens:

$$\mathcal{L}_{\text{CE}} = -\sum_{t\in\mathcal{T}_a} \log p_{\theta+\Delta_\Phi(q,\mathcal{D})}(y_t \mid y_{<t}, q), \quad (17)$$

where $\mathcal{T}_a$ denotes the answer-token positions.

Regularization. To stabilize update generation, we add two KL-based regularizers. $\mathcal{L}_{\text{KD}}$ distills from an in-context teacher that observes both the query and retrieved passages, while $\mathcal{L}_{\text{prior}}$ regularizes the adapted model toward the frozen backbone under the same query-only prompt. The full objective is

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_D \mathcal{L}_{\text{KD}} + \lambda_P \mathcal{L}_{\text{prior}} + \lambda_2 \|\Phi\|_2^2. \quad (18)$$

4 Experiments

4.1 Experimental Setup

Benchmarks and baselines. To ensure a fair comparison with prior in-weights RAG methods, we follow the experimental setup of DyPRAG (Tan et al., 2025), including its evaluation benchmarks, retrieved passages, and decoding protocol. We evaluate on four open-domain and multi-hop QA benchmarks: 2WikiMultihopQA (2WQA), HotpotQA (HQA), PopQA (PQA), and ComplexWebQuestions (CWQ) (Ho et al., 2020; Yang et al., 2018; Mallen et al., 2023; Talmor and Berant, 2018; Tan et al., 2025), using three instruction-tuned backbones of different scales: LLaMA-3.2-1B, Qwen-2.5-1.5B, and LLaMA-3-8B. We compare Doc2Weight against Vanilla (no retrieval), Standard RAG (Lewis et al., 2020), and two recent in-weights RAG baselines, PRAG (Su et al., 2025) and DyPRAG (Tan et al., 2025). Under the same prompt format, all retrieval-augmented methods receive identical retrieved passages, so performance differences mainly reflect how retrieved evidence is incorporated. Implementation details are provided in Appendix §A.

Evaluation. We evaluate Doc2Weight under two prompting settings: (1) *query-only*, our primary setting, where the textual prompt contains only the question and retrieved passages are used solely to construct the instance-specific update, directly reflecting the goal of augmenting generation without expanding the prompt; and (2) *In-context*, where the same passages are additionally concatenated into the prompt, which follows prior in-weights RAG work and tests whether weight-space adaptation complements explicit in-context evidence.

4.2 Main Results

Table 1 reports F1 for the *query-only* and *in-context* settings, with the macro-averaged F1 in the **Avg** column. Among the compared methods, Vanilla and Standard RAG use the off-the-shelf instruction-tuned backbone, whereas PRAG, DyPRAG, and Doc2Weight learn parameters that convert retrieved passages into weight updates. Within this setup, Doc2Weight obtains the best average F1 for every backbone in both settings.

In the query-only setting, our primary setting, Doc2Weight consistently outperforms the prior in-weights RAG methods (PRAG and DyPRAG), which are trained under the same protocol but do not condition the generated update on the query. This isolates the contribution of query-conditioned weighting: emphasizing query-relevant passages, rather than composing evidence in a query-agnostic way, yields better use of the retrieved set. Doc2Weight also improves over Vanilla across all backbones, showing that retrieved evidence can be integrated through weight-space adaptation alone, without placing any passages in the prompt.

A second observation concerns the in-context setting, where every method additionally reads the retrieved passages from the prompt: applying an in-weights update on top improves all in-weights methods, and Doc2Weight again achieves the best result. Since Doc2Weight and Standard RAG receive identical passages here, this gain shows that the retrieval-conditioned update is not redundant with the in-context evidence but provides a complementary signal, indicating that in-weights adapta-

Table 1: Main results across three backbones (F1). Scores marked † are taken from Tan et al. (2025), and **Avg** is recomputed as the macro-average of the four dataset **Total** scores for all methods. Per base model and column, the best result is in **bold** and the second-best is underlined.

Base LLM	Prompt Mode	Method	2WQA					HQA			PQA	CWQ	**Avg**
			Compare	Bridge	Infer.	Compos.	**Total**	Bridge	Compare	**Total**			
LLaMA-3.2-1B	Query-only	Vanilla†	42.89	24.17	16.91	7.87	22.52	13.25	40.26	18.79	2.26	34.94	19.63
		PRAG†	50.20	24.34	19.11	8.24	27.73	13.65	40.90	21.50	**23.58**	35.86	27.17
		DyPRAG†	51.25	48.15	17.35	7.54	25.31	**14.05**	**43.90**	19.97	11.33	36.86	23.37
		Doc2Weight	**57.74**	**51.67**	**26.24**	**13.39**	**37.26**	12.02	39.12	**25.57**	14.15	**42.76**	**29.93**
	In-context	Standard RAG†	41.23	26.78	22.51	10.21	23.12	21.38	42.46	27.14	17.65	37.39	26.33
		PRAG†	40.50	31.30	22.85	9.77	30.30	22.56	41.55	28.31	32.59	39.63	32.71
		DyPRAG†	**52.13**	46.19	22.54	12.60	29.18	22.05	43.78	26.58	29.93	38.96	31.16
		Doc2Weight	50.52	**48.69**	**30.53**	**15.16**	**36.23**	**29.42**	**47.94**	**38.68**	**35.06**	**42.76**	**38.18**
Qwen2.5-1.5B	Query-only	Vanilla†	45.74	39.06	17.04	7.27	26.87	12.18	39.46	17.76	2.87	26.47	18.99
		PRAG†	44.96	43.96	19.29	**11.14**	27.55	13.27	40.42	18.42	**21.55**	30.82	24.58
		DyPRAG†	43.03	**47.20**	17.04	8.55	26.46	13.72	41.39	19.67	6.64	31.94	21.18
		Doc2Weight	**52.97**	44.67	**23.67**	10.93	**33.06**	**16.04**	**44.68**	**30.36**	10.70	**39.48**	**28.40**
	In-context	Standard RAG†	38.75	38.84	11.87	5.68	24.31	16.19	37.13	20.73	9.97	28.23	20.81
		PRAG†	40.50	44.00	16.30	8.17	27.49	18.86	36.49	23.10	23.43	32.13	26.54
		DyPRAG†	35.83	**44.89**	14.81	8.64	25.18	21.56	**41.25**	27.57	22.69	33.57	27.25
		Doc2Weight	**55.66**	41.15	**30.62**	**14.68**	**35.53**	**31.33**	39.61	**35.47**	**36.99**	**41.52**	**37.38**
LLaMA-3-8B	Query-only	Vanilla†	54.90	55.20	24.59	14.43	33.02	19.00	45.63	21.29	7.96	42.44	26.18
		PRAG†	57.78	**58.93**	27.61	**19.17**	39.19	**33.68**	**65.88**	38.08	26.13	43.54	36.74
		DyPRAG†	57.39	56.43	25.33	18.88	37.80	24.85	58.59	28.56	13.60	41.87	30.46
		Doc2Weight	**58.61**	53.85	**29.15**	16.57	**39.55**	25.17	53.61	**39.39**	**27.38**	**51.66**	**39.50**
	In-context	Standard RAG†	58.43	47.77	19.20	11.07	34.55	19.68	42.10	24.23	16.13	35.45	27.59
		PRAG†	60.13	56.69	32.71	20.91	40.55	**39.41**	**68.22**	44.84	26.23	36.41	37.01
		DyPRAG†	**66.00**	**59.46**	35.78	**26.90**	**50.24**	33.37	57.93	38.35	**32.86**	39.07	40.13
		Doc2Weight	64.02	54.72	**39.87**	22.13	45.19	**39.41**	61.73	**50.57**	28.07	**46.29**	**42.53**

tion remains beneficial even when retrieval is also used in context. These improvements hold from the 1–1.5B models to LLAMA3-8B, suggesting that the benefit of weight-space retrieval augmentation is not tied to a particular model scale.

Beyond this setting, we further probe long-form (ELI5) and out-of-distribution QA (PubMedQA, SciQ) in Appendix §B.

5 Analysis

We analyze Doc2Weight along three axes: query-conditioned passage weighting, retrieval-depth robustness, and inference-time efficiency. Additional component ablations on the passage-signature form, LoRA rank, and SVD normalization are reported in Appendix §C.

5.1 Effect of Query-Conditioned Passage Weighting

Doc2Weight computes passage weights from both the query and the retrieved passage signatures. To isolate the role of query conditioning, we evaluate two query-agnostic variants that remove the query from the fusion module of Section 3.3. The *uniform* variant ignores the fused weights entirely and averages all passage signatures, $S_{\text{mix}} = \frac{1}{K}\sum_{i=1}^{K} S_i$. The *doc-centrality* variant instead drops the query signature s_q from the fusion sequence in Eq. (9), so the weights are computed only from interactions among the retrieved passages; this still yields non-uniform weights, but they reflect passage-to-passage similarity rather than relevance to the query.

Table 2: Query-conditioning ablation (F1). **Avg** is the macro-average over the four datasets.

Variant	2WQA	HQA	PQA	CWQ	**Avg**
LLaMA-3.2-1B					
Doc2Weight (full)	37.26	25.57	14.15	42.76	**29.93**
– query (uniform)	34.93	15.81	13.21	40.87	26.21
– query (doc-centrality)	33.51	16.58	13.49	40.93	26.13
Qwen2.5-1.5B					
Doc2Weight (full)	33.06	30.36	10.70	39.48	**28.40**
– query (uniform)	31.59	20.44	11.76	41.33	26.28
– query (doc-centrality)	31.54	20.71	11.26	41.61	26.28
LLaMA-3-8B					
Doc2Weight (full)	39.55	39.39	27.38	51.66	**39.50**
– query (uniform)	37.00	30.43	26.69	52.55	36.67
– query (doc-centrality)	37.76	30.54	26.60	51.96	36.72

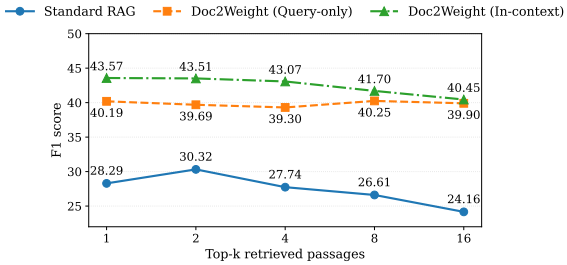

Figure 3: Effect of retrieval depth (top-k) on CWQ.

Table 2 shows that both variants lower the average F1 across all backbones. The comparison separates two effects: uniform averaging tests whether simple weight-space aggregation suffices, and document-centrality tests whether query-free non-uniform weighting can substitute for query-aware selection. Both underperform Doc2Weight, indicating that the gains stem neither from merely compressing passages into a single adapter nor from non-uniform weighting alone—the query is what determines which evidence should shape the injected update. The effect is most pronounced on HQA, where the useful evidence often depends on question-specific relations between passages.

5.2 Robustness to Retrieval Depth

We sweep top-$k \in \{1, 2, 4, 8, 16\}$ on CWQ using a BM25 retrieval pipeline implemented with Elasticsearch, following the DyPRAG retrieval setup. Figure 3 compares Standard RAG, query-only Doc2Weight, and Doc2Weight with in-context evidence, isolating how each form of evidence integration responds to increasing retrieval depth.

Query-only Doc2Weight stays nearly flat across depths (around 40 F1), indicating that its retrieval-conditioned update absorbs evidence from multiple passages without exposing the generator to a longer, noisier prompt. Standard RAG instead degrades as more passages are appended—improving from top-1 to top-2 but declining steadily thereafter—showing that simply adding textual evidence can hurt generation once redundant or distracting passages enter the prompt. In-context Doc2Weight is the strongest configuration at every depth and remains above both alternatives even at top-16. Since it accesses the same passages as Standard RAG but adds the fused weight-space update, this confirms that parameter-space adaptation is complementary to in-context retrieval and helps the model tolerate larger retrieval depths.

Table 3: Prompt length (tokens) and end-to-end throughput (tok/s) on the test set; relative change vs. Standard RAG in parentheses.

Method	Prompt len. ↓	Throughput ↑
LLaMA-3.2-1B		
Standard RAG	1,349.3	**0.401**
Doc2Weight	**68.9** (−94.9%)	0.381 (−5.0%)
Qwen2.5-1.5B		
Standard RAG	1,356.0	**0.684**
Doc2Weight	**63.1** (−95.3%)	0.657 (−3.9%)
LLaMA-3-8B		
Standard RAG	1,338.1	**1.116**
Doc2Weight	**43.9** (−96.7%)	1.037 (−7.1%)

5.3 Inference-Time Efficiency

Because Doc2Weight generates from a query-only prompt, it removes the retrieved passages from the autoregressive generation context, whereas Standard RAG generates from a prompt whose length scales with the retrieved set, $|q| + \sum_i |p_i|$. As shown in Table 3, this shortens the generation prompt from roughly 1.3K tokens to fewer than 70 (about a 20× reduction) across all backbones, lowering the memory and KV-cache footprint of generation. The trade-off is modest: since Doc2Weight encodes the retrieved passages in an additional forward pass, its end-to-end throughput is slightly lower than Standard RAG (by about 4–7%). We thus view Doc2Weight as trading a small throughput cost for a large reduction in prompt-side memory, and report these as preliminary prompt-side efficiency evidence rather than a full serving-level latency analysis.

6 Conclusion

We presented Doc2Weight, an in-weights retrieval-augmented generation framework that composes multiple retrieved passages into a single query-conditioned low-rank update, so that each passage contributes according to its relevance to the current query and retrieval augmentation requires no expansion of the inference-time prompt. Across three instruction-tuned backbones and four QA benchmarks, Doc2Weight consistently improves over standard in-context RAG and prior in-weights RAG baselines, in both the query-only and in-context settings. Our analysis attributes these gains to query-conditioned fusion, and further shows that Doc2Weight stays robust as retrieval depth increases while sharply reducing prompt length at a small throughput cost—a practical approach under prompt-length and memory constraints.

Limitations

Doc2Weight summarizes each retrieved passage into a compact second-order signature and fuses these into a single low-rank update. This compression is what removes the passages from the prompt, but it can discard fine-grained token- or sentence-level information, so Doc2Weight may be less suitable than in-context RAG for tasks requiring precise evidence attribution, citation, or span-level reasoning, where the full evidence text must stay visible. Accordingly, our evaluation centers on multi-passage short-form QA with EM/F1 metrics. Pilot studies on long-form QA (ELI5) and out-of-distribution QA (Appendices §B.1 and §B.2) suggest the approach extends further, but they are small-scale; behavior on broader long-form or citation-heavy generation—and on out-of-distribution retrieval, where irrelevant passages cannot be filtered as in the prompt—remains insufficiently characterized.

Unlike in-context RAG, which applies to any off-the-shelf model, Doc2Weight trains the lightweight parameters that map passages to weight updates. Following the PEFT-style setting, these are learned per backbone while the backbone stays frozen, so the converter must be retrained for a new architecture, and its scalability to larger same-family models remains uncharacterized; a fully training-free variant is left to future work. Applying an instance-specific update per query is also not natively supported by serving engines (e.g., vLLM) optimized for static adapters, so efficient on-the-fly injection is an additional engineering requirement.

Finally, our measurements, though collected within our own pipeline, already show that weight-space injection is not free at inference: constructing and applying the per-query update lowers throughput relative to in-context RAG (Section 5.3), so the prompt-length savings come with a compute cost rather than a uniform speedup. A precise accounting would require evaluation under realistic serving conditions, where latency and memory further depend on retrieval, batching, KV-cache behavior, hardware, and the serving engine.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. 2016. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, volume 29.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, and 9 others. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Ben Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. 2025. Generative adapter: Contextualizing language models in parameters with a single forward pass. In *International Conference on Learning Representations*, volume 2025, pages 27450–27470.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

David Ha, Andrew Dai, and Quoc V. Le. 2016. Hypernetworks. *Preprint*, arXiv:1609.09106.

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.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. *Preprint*, arXiv:2212.04089.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. 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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Richard James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. RA-DIT: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*. OpenReview.net.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Xuebing Liu, Shanbao Qiao, and Seung-Hoon Na. 2025. Genpoe: Generative passage-level mixture of experts for knowledge enhancement of llms. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5082–5097.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Abhinav Rastogi, Albert Q Jiang, Andy Lo, Gabrielle Berrada, Guillaume Lample, Jason Rute, Joep Barmentlo, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, and 1 others. 2025. Magistral. *arXiv preprint arXiv:2506.10910*.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9355–9366. PMLR.

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.

Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. 2025. Parametric retrieval augmented generation. *Preprint*, arXiv:2501.15915.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Yuqiao Tan, Shizhu He, Huanxuan Liao, Jun Zhao, and Kang Liu. 2025. Dynamic parametric retrieval augmented generation for test-time knowledge enhancement. *arXiv preprint arXiv:2503.23895*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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.

Table 4: Default training hyperparameters; sweepable values are marked.

Hyperparameter	Value
Optimization	
Optimizer	AdamW
Learning rate	2×10^{-4}
LR schedule	Cosine
Batching	
Effective batch size	16
Max sequence length	2048
Mixed precision	BF16
Adapter	
LoRA rank r	16 (sweepable)
Target modules	FFN projections
Retrieval	
Max passages k	4 (sweepable)
Max passage length	1024 tokens

A Implementation Details

Unless otherwise noted, all experiments follow the DyPRAG setup, reusing its retrieved passages, evaluation splits, and decoding settings.

Hardware and training data. All experiments run on a single machine with 8 NVIDIA RTX A6000 GPUs. Models are trained on the DyPRAG-augmented QA data; the original QA pairs are not used for training. Each training instance consists of an augmented question, its answer, and one or more associated retrieved passages, including DyPRAG-provided rewritten passages where available.

Evaluation setup. Evaluation uses the original benchmark test examples, which are disjoint from the augmented training data. We report Exact Match (EM) and F1 with SQuAD-style normalization, taking the best score over all acceptable gold answers.

Base models and optimization. We use instruction-tuned backbones from the LLaMA-3 / LLaMA-3.2 and Qwen-2.5 families, keeping all backbone parameters frozen and optimizing only the lightweight Doc2Weight parameters (Eq. 16). We train with AdamW, a cosine learning-rate schedule, and mixed precision. Since the backbone is already instruction-tuned and we train only a small converter on a modestly sized augmented set, a single epoch is sufficient; longer training tends to overfit in this setting. Default hyperparameters and the values we sweep are listed in Table 4.

Evaluation prompt

You should answer the question by referring to the knowledge provided below and integrating your own knowledge.
`{passages}`
Question: `{question}`

Figure 4: Evaluation prompt template, identical across both settings. In the in-context setting, `{passages}` is filled with the retrieved passages; in the query-only setting, it is the empty string, so no passages appear in the input.

Prompt templates. We use the DyPRAG-style single-turn chat format for both training and evaluation, with the standard causal-LM loss computed over assistant tokens only. The two prompt modes differ only in whether retrieved passages appear in the textual input: the query-only mode keeps the prompt to the instruction and question, while the in-context mode additionally concatenates the retrieved passages. In both modes, Doc2Weight uses the same retrieved passages to construct the weight update; query-only thus tests whether retrieval can be injected through parameter space without exposing the passages as input tokens. The templates are shown in Figure 4.

B Extended Evaluation

We include two pilot studies that probe how Doc2Weight behaves beyond its main setting of short-form, in-domain QA. Both target the two assumptions built into the method—that compact second-order signatures retain enough information, and that a converter trained on a fixed distribution still transfers: long-form generation (ELI5) stresses the first, and out-of-distribution QA (PubMedQA, SciQ) stress the second. Both use small evaluation splits and are meant as pilot probes rather than full benchmarks.

B.1 Long-form QA

Compressing passages into compact parameter-space signatures may discard fine-grained details that matter for long-form generation, so we run a pilot evaluation on ELI5. Rather than training from scratch, we take the Doc2Weight model from the main experiments and lightly adapt it to ELI5 (450 examples, one epoch), evaluating on 50 held-out examples with F1 and ROUGE-L under both prompting settings.

As shown in Table 5, Doc2Weight improves ROUGE-L over the corresponding baseline in all

Table 5: Long-form QA results on ELI5.

Method	F1	ROUGE-L
LLaMA-3.2-1B		
Vanilla	15.68	15.38
Standard RAG	15.28	13.95
Doc2Weight (Query-only)	15.21	16.23
Doc2Weight (In-context)	15.46	16.12
Qwen2.5-1.5B		
Vanilla	15.50	13.22
Standard RAG	15.36	13.60
Doc2Weight (Query-only)	19.14	16.07
Doc2Weight (In-context)	15.44	14.63
LLaMA-3-8B		
Vanilla	16.30	15.15
Standard RAG	15.40	14.38
Doc2Weight (Query-only)	19.11	17.48
Doc2Weight (In-context)	18.25	16.40

Table 6: Out-of-distribution QA results (F1). Avg is the macro-average over PubMedQA and SciQ.

Method	PubMedQA	SciQ	Avg
LLaMA-3.2-1B			
Vanilla	37.00	24.40	30.70
Standard RAG	58.40	54.81	56.61
Doc2Weight (Query-only)	56.20	36.16	46.18
Doc2Weight (In-context)	57.00	77.87	67.44
Qwen2.5-1.5B			
Vanilla	53.60	32.60	43.10
Standard RAG	57.60	68.57	63.08
Doc2Weight (Query-only)	56.00	47.22	51.61
Doc2Weight (In-context)	56.00	80.70	68.35
LLaMA-3-8B			
Vanilla	58.80	56.88	57.84
Standard RAG	73.20	82.70	77.95
Doc2Weight (Query-only)	61.00	54.56	57.78
Doc2Weight (In-context)	72.40	82.81	77.61

settings, and F1 in all but one (the only drop is a small F1 decrease for LLaMA-3.2-1B in the query-only setting, where ROUGE-L still improves). The consistent ROUGE-L gains suggest that retrieval-conditioned updates can improve long-form lexical coverage even when passages are compressed into parameter space, while the isolated F1 drop hints at a trade-off: compression may favor broader coverage over exact short-answer overlap. With only light adaptation to a small ELI5 split, we read these as preliminary evidence that Doc2Weight can extend to long-form generation, pending a larger-scale evaluation.

B.2 Out-of-Distribution QA

Because the update generator is trained on a fixed, general-domain distribution, a natural question is whether it still injects useful signal when the retrieved evidence comes from a domain unseen during training. We test this on two out-of- distribution QA datasets, PubMedQA and SciQ, evaluating without any domain-specific training (Table 6).

The first observation is that Doc2Weight improves over the closed-book Vanilla model in essentially every case, under both prompting settings. Since the only difference from Vanilla is the injected retrieval-conditioned update, this confirms that the update actually transmits content from the retrieved passages into generation, rather than collapsing under distribution shift. In the query-only comparison, where no passages enter the prompt, Doc2Weight improves over Vanilla (e.g., PubMedQA $37.0 \rightarrow 56.2$ and SciQ $24.4 \rightarrow 36.2$ on LLaMA-3.2-1B) but stays below Standard RAG, since the update is then the only channel through which the domain evidence reaches the model.

Combining both channels is the most reliable configuration. In the in-context setting, Doc2Weight reads the same passages as Standard RAG and additionally injects the weight-space update; on average it surpasses Standard RAG on the two smaller backbones (e.g., $56.6 \rightarrow 67.4$ on LLaMA-3.2-1B) and is comparable on LLaMA-3-8B. The behavior is asymmetric across datasets: the update yields a large gain on SciQ (up to $54.8 \rightarrow 77.9$) and stays close to Standard RAG on PubMedQA, so the prompt evidence provides a stable floor while the update adds a sizable complement when it aligns with the domain.

Notably, this robustness holds even though the update generator is trained only on short-form, general-domain QA, which suggests that training on more diverse domains and larger data is a promising direction for further strengthening out-of-distribution performance.

C Component Ablations

We ablate three design choices of Doc2Weight—the form of the passage signature, the LoRA rank of the injected update, and SVD-based signature normalization—on LLaMA-3.2-1B, varying only the ablated factor and holding all other settings fixed within each study. We evaluate on the same test sets as the main experiments and report macro-averaged scores over the four QA benchmarks, under both the query-only and in-context settings.

Table 7: Signature form ablation.

Prompt Mode	Signature	F1
Query-only		
	Last-hidden (DyPRAG-style)	26.90
	Mean-pool (GenPoE-style)	26.90
	Second-order (ours)	**27.40**
In-context		
	Last-hidden (DyPRAG-style)	35.96
	Mean-pool (GenPoE-style)	35.70
	Second-order (ours)	**36.79**

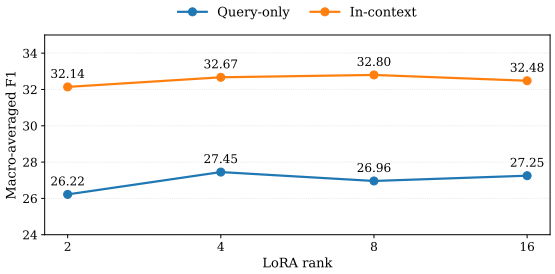

Figure 5: Performance as the LoRA rank varies.

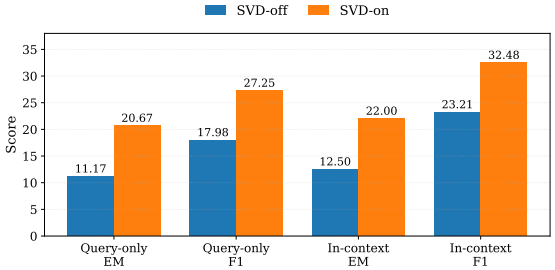

Figure 6: Performance with and without SVD normalization.

C.1 Signature Form

As described in Section 3.2, our passage signature is a second-order summary: starting from the token-level hidden states $\mathbf{H}_i$ (Eq. 4), Doc2Weight forms a Gram matrix (Eq. 6) that retains token-level co-activation structure. While Generative Adapter (Chen et al., 2025) has shown second-order statistics to be effective for context-to-parameter mapping, we verify that this choice also benefits our retrieval-fusion setting rather than assuming it. We compare against two first-order alternatives that replace the Gram-based signature with a single vector summarizing each passage:

$$s_i^{\text{mean}} = \frac{1}{N_i} \sum_{t=1}^{N_i} \mathbf{H}_i[t,:], \qquad s_i^{\text{last}} = \mathbf{H}_i[N_i,:]. \tag{19}$$

Unlike our second-order signature, these variants use the first-order vector directly, without the Gram construction or its subsequent normalization. The mean-pool variant s_i^{mean} mirrors the mean-pooled passage representation used in GenPoE (Liu et al., 2025), while the last-hidden variant s_i^{last} corresponds to the compressed final-token representation used in DyPRAG (Tan et al., 2025).

As shown in Table 7, the second-order signature attains the best macro-averaged F1 in both prompting settings, indicating that retaining token-level co-activation structure, rather than collapsing each passage into a single first-order vector, benefits weight-space fusion.

C.2 LoRA Rank

We vary the rank $r \in \{2, 4, 8, 16\}$ of the retrieval-conditioned update (Figure 5). Performance saturates early: F1 rises from rank 2 to rank 4 and is then essentially flat through rank 16 in both settings (within about 0.5 F1 of the best rank in each case). This indicates that Doc2Weight does not depend on large adapter capacity—a small rank around 4 already captures most of the benefit—so its effectiveness stems more from query-conditioned fusion and a stable signature-to-weight mapping than from adapter size. A small rank thus offers a strong efficiency–performance trade-off.

C.3 SVD Normalization

We compare enabling and disabling the SVD-based normalization applied to each passage signature (Figure 6). Normalization helps substantially in both prompting settings, improving F1 from 18.0 to 27.3 in the query-only setting and from 23.2 to 32.5 in the in-context setting, with comparably large EM gains. The improvement in the query-only setting is particularly informative: since the update is then the only channel carrying retrieved evidence, the large gain shows that normalization improves the quality of the parameter-space update itself, rather than merely interacting with in-context evidence. This is consistent with its role of removing scale variation in the second-order signatures so that they map to more stable low-rank updates.