A Comparison of Independent and Joint Fine-tuning Strategies for Retrieval-Augmented Generation

Neal Lawton, Alfy Samuel, Anoop Kumar, Daben Liu
{neal.lawton, alfy.samuel, anoop.kumar, daben.liu}@capitalone.com

Abstract

Retrieval augmented generation (RAG) is a popular framework for question answering that is powered by two large language models (LLMs): an embedding model that retrieves context documents from a database that are relevant to a given question, and a generator model that uses the retrieved context to generate an answer to the question. Both the embedding and generator models can be fine-tuned to increase performance of a RAG pipeline on a new task, but multiple fine-tuning strategies exist with different costs and benefits. In this paper, we evaluate and compare several RAG fine-tuning strategies, including independent, joint, and two-phase fine-tuning. In our experiments, we observe that all of these strategies achieve about equal improvement in EM and F1 generation quality metrics, although they have significantly different computational costs. We conclude the optimal fine-tuning strategy to use depends on whether the training dataset includes context labels and whether a grid search over the learning rates for the embedding and generator models is required.

1 Introduction

Retrieval augmented generation (RAG) is a popular framework for NLP tasks like question answering. RAG is powered by two LLMs: an embedding model that retrieves context documents from a database that are relevant to a given question, and a generator model that uses the retrieved context documents to generate an answer to the question.

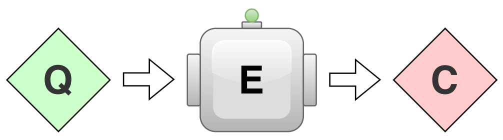
Both the embedding model and generator model can be fine-tuned to improve the end-to-end performance of a RAG pipeline. Given a dataset of (*question*, *context*) pairs, the embedding model can be fine-tuned to retrieve more relevant context documents for a given question. This requires a training dataset with context labels, i.e., where each question is paired with one or more relevant context documents from the database. Given a dataset of (*question*, *context*, *answer*) triplets, where the context is either provided as part of the training dataset as context labels or retrieved from the database using a baseline embedding model, the generator model can be fine-tuned to increase the likelihood of generating the correct answer given the question and relevant context documents.

Although the embedding and generator models can be fine-tuned independently, fine-tuning both models jointly with an end-to-end fine-tuning method such as RAG-Token or RAG-Sequence (Lewis et al., 2020) may yield equal or better end-to-end performance without the need for context labels. Additionally, we consider a two-phase fine-tuning strategy that uses RAG-Token to first fine-tune the generator model while holding the embedding model frozen, then fine-tunes the embedding model while holding the generator model frozen.

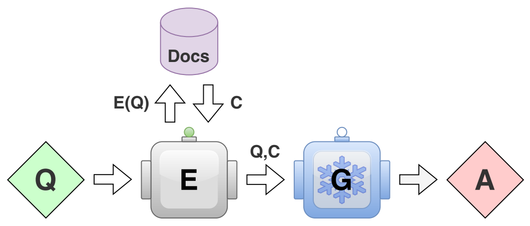
The choice of learning rate used for fine-tuning may significantly affect the end-to-end performance of the RAG pipeline, and the optimal choice of learning rate for the embedding and generator models may be different. We use a grid search to find a suitable choice of learning rates.

In this paper, we compare independent, joint, and two-phase fine-tuning and find they all achieve similar end-to-end performance when using a suitable choice of learning rates. Based on our experimental results, we make the following conclusions:

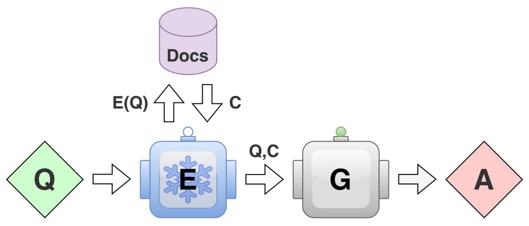
- Independent fine-tuning is the least computationally expensive strategy, and so should be used when possible. However, this strategy can only be used if the training dataset includes context labels.
- If context labels are not available, but a suitable choice of learning rate for the embedding and generator models is already known, then joint fine-tuning should be used since it is less computationally expensive than two-phase fine-tuning.



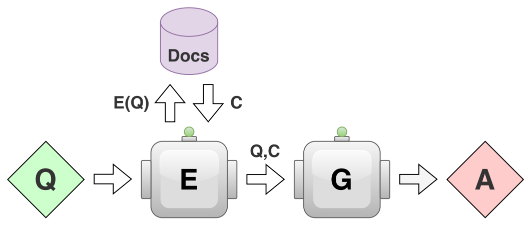
(a) Fine-tune the embedding model using context labels.



(b) Freeze the generator model while fine-tuning the embedding model with either RAG-Token or RAG-Sequence.



(c) Freeze the embedding model while fine-tuning the generator model with RAG-Token or RAG-Sequence.



(d) Fine-tune the embedding and generator models jointly with RAG-Token or RAG-Sequence.

Figure 1: RAG fine-tuning strategy subprocesses. Each of the RAG fine-tuning strategies discussed in this paper uses a combination of these subprocesses. Key: **Q**uestion, **C**ontext, **A**nswer, **E**mbedding model, **G**enerator model.

- If context labels are not available and a suitable choice of learning rates for the embedding and generator models is unknown, then two-phase fine-tuning should be used while performing independent grid searches over the learning rates for the embedding and generator models.

2 Fine-tuning Strategies

2.1 Embedding Model Fine-tuning

The embedding model of a RAG pipeline can be fine-tuned to retrieve more relevant context documents given a dataset of (*question*, *context*) pairs by minimizing the distance (or maximizing the similarity) between the embedding vectors of each (*question*, *context*) pair. This method is illustrated in Figure 1a. Note that the embedding vectors of the context documents are held frozen in the pre-computed vector database, so that only the embedding vectors of the questions are updated. There are many different options for the choice of loss function to minimize, including contrastive loss (Hadsell et al., 2006), multiple negatives ranking loss (Henderson et al., 2017), and the GISTEmbed loss (Solatorio, 2024) using either cosine similarity or L_2 distance as the distance metric. Cached variants (Gao et al., 2021) of these methods exist that allow for effectively much larger batch sizes without increased GPU memory usage. In our experiments, we use cosine similarity as the distance metric and multiple negatives ranking loss without caching with batch size 8 as the loss function.

2.2 Generator Model Fine-tuning

The generator model can be fine-tuned by minimizing the negative log-likelihood of the answer given the question and relevant context documents. In our experiments, we always fine-tune the generator model using context retrieved by a baseline embedding model rather than context labels. This is equivalent to the "frozen embedding" fine-tuning process illustrated in Figure 1c. In our experiments, we fine-tune the generator model with QLoRA (Dettmers et al., 2023; Hu et al., 2022) using LoRA rank 16 and 4-bit quantization.

2.3 Joint Fine-tuning

The embedding and generator models can be fine-tuned jointly by fine-tuning the RAG pipeline end-to-end with either RAG-Token or RAG-Sequence (Lewis et al., 2020), illustrated in Figure 1d. Both these methods optimize an objective that is fully differentiable with respect to both the embedding model and generator model's parameters by approximating the RAG pipeline with a simplified probability model; the two methods differ only in the approximation they make. Instead of using context labels, these methods use context retrieved by the embedding model to fine-tune the generator model, and reward the embedding model for retrieving context documents that actually improve the generator model's prediction for the answer. In our experiments, we use full fine-tuning for the embedding model and QLoRA for the generator model. We fine-tune using two learning rates: one for the embedding model's parameters, and the other for the generator model's parameters.

2.4 Two-Phase Fine-tuning

We also consider a two-phase fine-tuning strategy that uses RAG-Token to first fine-tune the generator model while holding the embedding model frozen as in Figure 1c, then fine-tunes the embedding model while holding the generator model frozen as in Figure 1b. As in joint fine-tuning, we fine-tune using two learning rates.

2.5 Learning Rate Grid Search

Using a suitable choice of learning rate is important for maximizing end-to-end performance for each fine-tuning strategy. In order to find a near-optimal choice of learning rate, we perform a grid search over the learning rate for each experiment. Performing this grid search is computationally inexpensive for strategies that fine-tune only either the embedding model or generator model: we simply repeat the experiment for each grid value, then keep only the result that achieves the best end-to-end validation performance. The grid search is also computationally inexpensive when fine-tuning both models independently or with the two-phase strategy, since the grid search can be performed independently for the embedding and generator models. However, jointly optimizing over the learning rates for the embedding and generator models is much more computationally expensive. Instead, in our joint fine-tuning experiments, we use the same learning rates as those discovered by the grid search for the two-phase fine-tuning strategy.

3 Experiments

Here we evaluate and compare the performance of the RAG fine-tuning strategies described in the previous section for four RAG pipelines, each consisting of either an MPNet (Reimers and Gurevych, 2019) or MiniLM (Reimers and Sanseviero, 2021) embedding model and either a LLaMA-3-8b-Instruct (AI@Meta, 2024) or Mistral-7b-Instruct-v0.1 (Jiang et al., 2023) generator model. We fine-tune and evaluate on two datasets: HotPotQA (Yang et al., 2018) and PopQA (Mallen et al., 2022). Our retrieval system uses the embedding model to retrieve the top $k = 5$ most relevant documents from Wikipedia[1]. We use the same chunking of Wikipedia as Xiong et al. (2024), which contains 29.9M chunks. We construct a vector database from the corpus using a FAISS index (Johnson et al., 2019). Each experiment was conducted on a node with 8 NVIDIA A10 GPUs.

To minimize the computational expense of our experiments, in each experiment we fine-tune for only 1 epoch (for the two-phase strategy, each model is fine-tuned for 1 epoch) (Komatsuzaki, 2019; Egele et al., 2023). In all experiments, we use a linear learning rate schedule. To find near-optimal choices of learning rates, we perform a grid search over values between 10^{-8} and 10^{-4}, with grid values separated roughly by factors of 3: specifically, 10^{-8}, 3×10^{-8}, 10^{-7}, 3×10^{-7}, 10^{-6}, 3×10^{-6}, 10^{-5}, 3×10^{-5}, and 10^{-4}.

3.1 Results

The results of our experiments are in Table 3 and illustrated in Figure 2. Each cell shows the validation exact match (EM), F1 metric, and Recall@5 for each experiment, averaged over the four RAG pipelines described at the beginning of this section. "No Ft." is the baseline RAG pipeline with no fine-tuning. "Ft. Embed." fine-tunes only the embedding model using context labels and the multiple negatives ranking loss. "Ft. Gen." fine-tunes only the generator model. "Indp." combines the independently fine-tuned embedding and generator models from "Ft. Embed." and "Ft. Gen." "2-Phase" is the two-phase fine-tuning strategy. "RAG-Seq." and "RAG-Tok." fine-tune the embedding and generator models jointly with RAG-Sequence and RAG-Token, respectively.

Comparing the "Baseline", "Ft. Embed.", and "Ft. Gen." experiments, we observe that fine-tuning the generator model alone significantly improves EM and F1 scores and that fine-tuning the embedding alone significantly improves Recall@5, with downstream benefits for EM and F1. We also observe that fine-tuning the generator model is much more computationally expensive than fine-tuning the embedding model using context labels. This is because the generator model is much larger than the embedding model, and so the latency of a single forward pass is much higher for the generator model than for the embedding model.

[1]https://huggingface.co/datasets/legacy-datasets/wikipedia

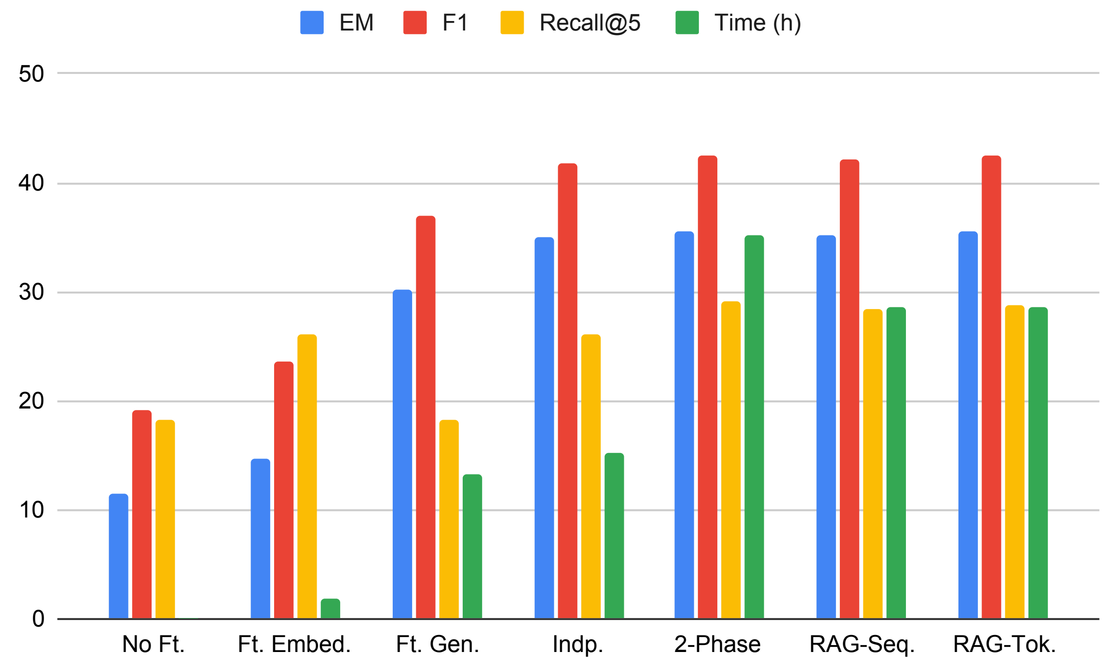


Figure 2: Validation performance metrics and time to fine-tune for different fine-tuning strategies, averaged across all four RAG pipelines and both HotPotQA and PopQA datasets.

Method	HotPotQA				PopQA			
	EM	F1	Recall@5	Time (h)	EM	F1	Recall@5	Time (h)
No Ft.	10.3	19.8	19.1	0.0	12.6	18.6	17.4	0
Ft. Embed.	11.1	20.8	21.4	3.5	18.2	26.6	30.8	0.4
Ft. Gen.	28.4	39.4	19.1	23.8	32.1	34.7	17.4	2.9
Indp.	29.3	40.2	21.4	27.4	40.6	43.2	30.8	3.2
2-Phase	30.0	41.3	25.1	61.0	41.0	43.7	33.3	9.4
RAG-Seq.	29.1	40.2	24.0	49.2	41.4	44.1	32.8	7.9
RAG-Tok.	29.5	40.8	24.3	49.3	41.6	44.4	33.1	8.0

Figure 3: HotPotQA and PopQA validation performance metrics after fine-tuning and time to fine-tune for different fine-tuning strategies, averaged across all four RAG pipelines.

Comparing "Ft. Embed." to "2-Phase", "RAG-Seq.", and "RAG-Tok.", we observe that fine-tuning the embedding model using context labels may achieve worse Recall@5 compared to the end-to-end methods that do not use context labels. However, it may be possible to improve the results for our "Ft. Embed." experiment by using the cached variant of the multiple negatives ranking loss and increasing the batch size.

We observe that "Indp.", "2-Phase", "RAG-Sequence", and "RAG-Token" all achieve about the same EM and F1 scores. This suggests these strategies are about equally effective for fine-tuning a RAG pipeline. However, the strategies have significantly different computational cost: independent fine-tuning is the least expensive, followed by joint fine-tuning with RAG-Sequence or RAG-Token, followed by the two-phase fine-tuning strategy.

4 Conclusion

In this paper, we compared various strategies for fine-tuning the embedding and generator models of a RAG pipeline. From our experiments with four different RAG pipelines on HotPotQA and PopQA, we observed that independent, joint, and two-phase fine-tuning are all about equally effective for fine-tuning a RAG pipeline. While independent fine-tuning is computationally less expensive, joint fine-tuning and two-phase fine-tuning have the benefit of not requiring context labels to perform fine-tuning. In addition, two-phase fine-tuning allows for a more efficient hyperparameter search for the embedding and generator model learning rates compared to joint fine-tuning.

Limitations

In order to maximize the end-to-end performance of each fine-tuning strategy, we used a grid search to find near-optimal choices of the learning rates for the embedding and generator models. However, it may be possible to further increase end-to-end performance by additionally performing hyperparameter optimizations over the number of training epochs and the training batch size. In particular, it may be possible to improve the end-to-end performance achieved in the "Ft. Embed." experiments, which fine-tune the embedding model by optimizing the multiple negatives ranking loss, by increasing the training batch size to a number much larger than 8.

We perform our fine-tuning experiments using a basic RAG pipeline setup. However, more complex RAG pipelines are common in practice, e.g., pipelines that perform context document re-ranking after the document retrieval step, or pipelines that perform multiple document retrieval steps to answer multi-hop questions. It remains unclear how introducing these complexities to the RAG pipeline might impact the effectiveness of each of the fine-tuning strategies discussed in this paper.

References

AI@Meta. 2024. Llama 3 model card.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Romain Egele, Isabelle Guyon, Yixuan Sun, and Prasanna Balaprakash. 2023. Is one epoch all you need for multi-fidelity hyperparameter optimization? *arXiv preprint arXiv:2307.15422*.

Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling deep contrastive learning batch size under memory limited setup. *arXiv preprint arXiv:2101.06983*.

Raia Hadsell, Sumit Chopra, and Yann Lecun. 2006. Dimensionality reduction by learning an invariant mapping. pages 1735 – 1742.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

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.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Aran Komatsuzaki. 2019. One epoch is all you need. *arXiv preprint arXiv:1906.06669*.

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.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Nils Reimers and Omar Sanseviero. 2021. Sentence transformers in the hugging face hub. https://huggingface.co/blog/sentence-transformers-in-the-hub.

Aivin V Solatorio. 2024. Gistembed: Guided in-sample selection of training negatives for text embedding fine-tuning. *arXiv preprint arXiv:2402.16829*.

Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. 2024. Benchmarking retrieval-augmented generation for medicine. *arXiv preprint arXiv:2402.13178*.

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.

A Prompt

In all experiments, we use the following prompt for the generative model to generate an answer given a question and concatenated context documents.

```
prompt = """You are a helpful general \
knowledge expert. Answer the following \
question using the relevant context. Use \
as few words as possible.

### Context:
{context}

### Question:
{question}

### Answer:
"""
```

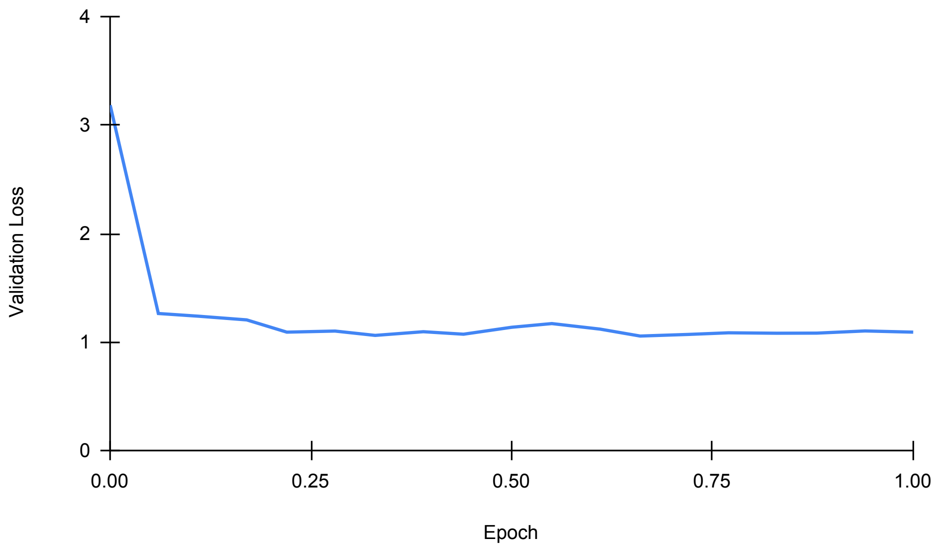


Figure 4: Validation loss convergence plot for fine-tuning a RAG pipeline consisting of a MiniLM embedding model and LLaMA-3-8b generator model on HotPotQA with joint fine-tuning. The validation loss converges quickly during fine-tuning, well within the 1 epoch fine-tuning period.

Model Name	# Params
MiniLM	22.7M
MPNet	109M
Mistral-7b	7.24B
LLaMA3-8b	8.03B

Figure 5: Number of parameters in each model used in this paper.

Embed. Model	Gen. Model	Method	HotPotQA EM	F1	Recall@5	Time(h)	Embed. LR	Gen. LR
MiniLM	LLaMA3-8b	No Ft.	15.3	24.6	19.5	0.0	N/A	N/A
		Ft. Embed	16.5	26.0	21.3	1.5	1E-06	N/A
		Ft. Gen	29.9	41.2	19.5	21.8	N/A	1E-05
		Indp.	30.5	41.7	21.3	23.3	1E-06	1E-05
		2-Phase	30.8	42.4	23.7	35.2	3E-08	1E-05
		RAG-Seq.	27.8	38.5	22.9	45.9	3E-08	1E-05
		RAG-Tok.	30.0	41.4	23.2	46.0	3E-08	1E-05
MiniLM	Mistral-7b	No Ft.	5.5	15.2	19.5	0.0	N/A	N/A
		Ft. Embed	6.2	15.7	21.3	1.5	1E-06	N/A
		Ft. Gen	26.8	37.5	19.5	24.6	N/A	1E-05
		Indp.	27.9	38.5	21.3	26.1	1E-06	1E-05
		2-Phase	27.7	38.7	23.0	36.6	3E-08	1E-05
		RAG-Seq.	27.5	38.4	22.6	49.9	3E-08	1E-05
		RAG-Tok.	26.8	37.3	22.3	49.8	3E-08	1E-05
MPNet	LLaMA3-8b	No Ft.	15.1	24.5	18.6	0.0	N/A	N/A
		Ft. Embed	16.0	25.9	21.5	5.5	1E-06	N/A
		Ft. Gen	29.8	41.0	18.6	22.9	N/A	3E-06
		Indp.	30.7	41.8	21.5	28.4	1E-06	3E-06
		2-Phase	32.1	43.8	27.3	37.9	3E-08	3E-06
		RAG-Seq.	31.8	43.7	25.7	48.7	3E-08	3E-06
		RAG-Tok.	31.9	44.0	26.4	49.1	3E-08	3E-06
MPNet	Mistral-7b	No Ft.	5.4	15.0	18.6	0.0	N/A	N/A
		Ft. Embed	5.7	15.6	21.5	5.5	1E-06	N/A
		Ft. Gen	27.2	37.8	18.6	26.0	N/A	1E-05
		Indp.	28.1	38.8	21.5	31.6	1E-06	1E-05
		2-Phase	29.4	40.6	26.4	39.1	3E-08	1E-05
		RAG-Seq.	29.1	40.4	24.7	52.4	3E-08	1E-05
		RAG-Tok.	29.2	40.3	25.3	52.5	3E-08	1E-05

Figure 6: HotPotQA validation performance metrics after fine-tuning, time to fine-tune, and learning rates used for different fine-tuning strategies and RAG pipelines.

Embed. Model	Gen. Model	Method	PopQA					
			EM	F1	Recall@5	Time(h)	Embed. LR	Gen. LR
MiniLM	LLaMA3-8b	No Ft.	17.3	23.4	17.9	0.0	N/A	N/A
		Ft. Embed	23.6	31.1	28.5	0.1	1E-05	N/A
		Ft. Gen	34.6	37.4	17.9	2.5	N/A	1E-05
		Indp.	40.8	43.7	28.5	2.6	1E-05	1E-05
		2-Phase	41.1	44.0	30.7	6.3	3E-07	1E-05
		RAG-Seq.	40.6	43.6	30.1	7.2	3E-07	1E-05
		RAG-Tok.	41.8	44.3	30.9	7.3	3E-07	1E-05
MiniLM	Mistral-7b	No Ft.	8.9	15.3	17.9	0.0	N/A	N/A
		Ft. Embed	12.1	20.4	28.5	0.1	1E-05	N/A
		Ft. Gen	30.9	33.4	17.9	2.7	N/A	3E-05
		Indp.	37.5	40.5	28.5	2.8	1E-05	3E-05
		2-Phase	38.6	41.5	31.3	6.5	3E-08	3E-05
		RAG-Seq.	39.5	42.3	30.6	7.7	3E-08	3E-05
		RAG-Tok.	39.9	42.4	31.4	7.8	3E-08	3E-05
MPNet	LLaMA3-8b	No Ft.	16.0	21.6	16.9	0.0	N/A	N/A
		Ft. Embed	25.1	33.4	33.1	0.6	3E-05	N/A
		Ft. Gen	33.6	36.1	16.9	3.0	N/A	1E-04
		Indp.	43.0	45.5	33.1	3.5	3E-05	1E-04
		2-Phase	43.2	45.9	35.8	6.4	3E-07	1E-04
		RAG-Seq.	44.0	46.5	35.4	8.1	3E-07	1E-04
		RAG-Tok.	42.4	46.1	35.2	8.1	3E-07	1E-04
MPNet	Mistral-7b	No Ft.	8.2	14.2	16.9	0.0	N/A	N/A
		Ft. Embed	12.0	21.2	33.1	0.6	3E-05	N/A
		Ft. Gen	29.2	31.9	16.9	3.3	N/A	3E-05
		Indp.	40.8	43.2	33.1	3.9	3E-05	3E-05
		2-Phase	40.9	43.5	35.5	6.9	3E-07	3E-05
		RAG-Seq.	41.5	44.1	35.2	8.7	3E-07	3E-05
		RAG-Tok.	42.2	44.9	35.0	8.6	3E-07	3E-05

Figure 7: PopQA validation performance metrics after fine-tuning, time to fine-tune, and learning rates used for different fine-tuning strategies and RAG pipelines.