FedRAG: A Framework for Fine-Tuning Retrieval-Augmented Generation Systems

Val Andrei Fajardo [1] **David B. Emerson** [1] **Amandeep Singh** [1] **Veronica Chatrath** [1] **Marcelo Lotif** [1] **Ravi Theja** [2] **Alex Cheung** [2] **Izuki Matsuba** [2]

Abstract

Retrieval-augmented generation (RAG) systems have been shown to be effective in addressing many of the drawbacks of relying solely on the parametric memory of large language models. Recent work has demonstrated that RAG systems can be improved via fine-tuning of their retriever and generator models. In this work, we introduce FedRAG, a framework for fine-tuning RAG systems across centralized and federated architectures. FedRAG supports state-of-the-art fine-tuning methods, offering a simple and intuitive interface and a seamless conversion from centralized to federated training tasks. FedRAG is also deeply integrated with the modern RAG ecosystem, filling a critical gap in available tools.

1. Introduction

Large Language Models (LLMs) have demonstrated a remarkable capacity to perform a diverse set of tasks, despite standard pre-training only incorporating a next-token prediction objective (Brown et al., 2020; Grattafiori et al., 2024; Chhun et al., 2022; Ren et al., 2023; Paul et al., 2024). Prompt engineering techniques, such as in-context learning, chain-of-thought (CoT), and self-consistency have been shown to further improve the performance of pre-trained LLMs in many settings (Brown et al., 2020; Wei et al., 2022; Wang et al., 2023a). However, there are tasks, particularly knowledge-intensive ones, where relying solely on information encoded in an LLM's parameters, often referred to as parametric memory, may lead to factual inaccuracies in model responses, colloquially named hallucinations (Huang et al., 2025). Hallucinations issues can become more prevalent when models attempt to address queries beyond their knowledge cutoffs (Dige et al., 2024; Ovadia et al., 2024).

Retrieval-augmented generation (RAG) is a popular methodology that aims to address this specific drawback (Lewis et al., 2020; Ram et al., 2023). Specifically, RAG systems present relevant non-parametric knowledge drawn from external systems alongside queries in the form of additional input to an LLM. Perhaps the most widely used form of this involves computing an embedding of right-sized chunks of the non-parametric knowledge and storing these in a vector store like Qdrant, Chroma or Pinecone for future search and retrieval (Fan et al., 2024). More elaborate designs of RAG systems have also been developed, such as those utilizing knowledge graphs (Peng et al., 2024).

Recent studies have shown that fine-tuning RAG systems can lead to even greater performance improvements (Lin et al., 2023b; Zhang et al., 2024; Chen et al., 2025). That is, through fine-tuning, the overall RAG system, comprised of a generator, retriever and knowledge store, may be adapted to work more cohesively as a single unit in performing tasks.

The RAG ecosystem of today is quite vibrant and includes a wide range of options for LLMs, retrieval models, and rerankers, among other components (Karpukhin et al., 2020; Ma et al., 2023; Blagojevic, 2023; Wang et al., 2023b; Zhuang et al., 2023) being offered by organizations operating under both open- as well as closed-source business models. There are also more than a few storage, observability, and evaluation solutions that developers can choose from in order to build an end-to-end RAG production. Finally, popular RAG frameworks such as LlamaIndex (Liu, 2022) and LangChain (Chase, 2022) offer users the ability to rapidly assemble and experiment with diverse RAG system configurations, ultimately aiding in the discovery of optimal designs. Yet, to the best of our knowledge, there are few, if any, frameworks that help simplify RAG fine-tuning, while remaining well integrated with other available tools and resources in the ecosystem.

The work presented here aims to directly fill this gap. Specifically, we introduce FedRAG, a framework for fine-tuning RAG systems across both centralized and federated architec-

[*]Equal contribution [1]Vector Institute, Toronto ON M5G 0C6, Canada [2]Independent Researcher, Toronto, Canada. Correspondence to: Val Andrei Fajardo <andrei.fajardo@vectorinstitute.ai>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

tures.[1] Decentralized designs for LLM training and deployment are becoming increasingly important, as evidenced by popular initiatives like Anthropic's Model Context Protocol (MCP) (Anthropic, 2024) and Google's Agent2Agent Protocol (Google, 2025). Moreover, in settings where data privacy prevents centralizing datasets, decentralized training techniques like federated learning (FL) become an indispensable tool for improving RAG systems.

2. Related Work

2.1. Fine-tuning RAG Systems

As discussed above, RAG systems are comprised of a number of components, some of which are driven by trainable models. This work specifically focuses on two main components: the generator, which is responsible for text generation; and the retrieval model, which maps queries or prompts into a form, commonly a high-dimensional embedding vector, used to retrieve related context from a knowledge store.

Several studies have focused on generator training via instruction fine-tuning, for which the instruction examples include context retrieved by the retriever from the knowledge store. Lin et al. (2023b) refer to this approach as Retrieval-Augmented Language Model Training (RALT). A similar generator fine-tuning approach called Retrieval-Augmented Fine-Tuning (RAFT) was utilized in (Zhang et al., 2024). RAFT differs from RALT in that the instruction examples also include LLM generated CoT passages with reasoning traces linking the response to the query. Finally, in line with recent trends in reasoning LLMs (Kumar et al., 2025), Chen et al. (2025) introduce ReSearch, which follows a reinforcement learning approach similar to that used in DeepSeek-R1 (DeepSeek-AI et al., 2025). With ReSearch, the LLM learns to generate long CoT passages that incorporate search and retrieval from a knowledge store. In so doing, the generator is adapted to cycle between reasoning and retrieval, potentially multiple times.

Fewer studies exist considering retriever fine-tuning. In the same work that produced RALT, the authors also introduce Language Model Supervised Retriever Training (LSR). In LSR, the retrieval scores of retrieved text chunks as well as corresponding target sequence probabilities produced by the generator model, conditioned on the context of each retrieved chunk, form two distributions whose distance, measured by the Kullback-Leibler divergence, is minimized.

2.2. Federated Learning for LLM Applications

Recently, Flower Labs developed the first federally pretrained LLM called FlowerLLM. Further, efforts supporting federated LLM tuning have been undertaken (Sani et al., 2024). In the context of RAG systems, however, work has primarily focused on decentralized inference rather than federated fine-tuning or training. A notable example published by Flower Labs demonstrates RAG inference in a federated setting.[2] Similarly, LlamaIndex has also developed a library extension to their framework called `llama-index-networks` (Fajardo et al., 2024) for decentralized RAG inference. To the best of our knowledge, no existing tools provide a simple interface for converting centralized RAG fine-tuning to federated tasks.

3. Philosophy and Design Principles

In this section, we describe the core philosophy and design principles that guide the development of FedRAG. These principles address the challenges identified in the previous section and inform implementation decisions.

3.1. Philosophy

We endeavour to build FedRAG for researchers and practitioners alike in a manner that makes applying state-of-the-art fine-tuning techniques to RAG systems simple, yet highly effective, irrespective of whether the system is centralized or federated. Moreover, we seek to make researching new methods for RAG fine-tuning more efficient and scientifically rigorous by promoting reproducibility. This is achieved through designing flexible components and appropriate tools that allow users to easily replicate and disseminate their RAG systems, as well as extend the library with custom trainers, losses, benchmarks, and more.

3.2. Design Principles

Advanced RAG Fine-Tuning: *Comprehensive support for state-of-the-art RAG fine-tuning methods that can be federated with ease.*

This principle is central to advancing the state of knowledge in RAG methods. By implementing and supporting frontier techniques in RAG fine-tuning, while simultaneously making federation straightforward and accessible, researchers are enabled to develop and evaluate novel approaches in a systematic and reproducible fashion. At the same time, such methods transfer smoothly to decentralized systems.

Work With Your Tools: *Seamless integration with popular frameworks including HuggingFace, Unsloth, and LlamaIndex, to become deeply embedded in the RAG ecosystem and beyond.*

By designing FedRAG as a deeply embedded framework

[1] Library Code: https://github.com/VectorInstitute/fed-rag

[2] https://flower.ai/docs/examples/fedrag.html

within the existing RAG ecosystem, barriers to adoption are significantly reduced for both practitioners and researchers. Integrations into popular frameworks and libraries, such as those mentioned above, allow users to leverage familiar tools and workflows while gaining access to advanced fine-tuning capabilities. Finally, extensive ecosystem compatibility facilitates discoverability and further adoption of new methods and results, thus maximizing the impact and reach of research advancements.

Lightweight Abstractions: *Clean, intuitive abstractions that simplify RAG fine-tuning while maintaining full flexibility and control.*

We seek to provide developers with an intuitive interface and abstractions that are easy to work with, customize, and extend. Lowering the learning curve to use FedRAG, while simultaneously increasing its utility and effectiveness, is essential to providing a pleasant development experience. This approach enables practitioners and researchers to focus their efforts entirely on the challenges of designing and experimenting with methods for improving RAG systems rather than wrestling with complex implementation details.

4. FedRAG Framework Overview

This section provides a more detailed overview of the FedRAG library.

4.1. Library Organization

FedRAG incorporates a modular design, consisting of several modules with clear and intuitive separation of concerns. Table 1 presents non-exhaustive overview of the key modules and their responsibilities.

Module	Description
core	Core types i.e., `RAGSystem`
evals	Evaluation metrics and benchmarks
fl_tasks	Federated learning task definitions
generators	Generator types
knowledge_stores	Data storage
retrievers	Retriever types
trainers	Trainer types

Table 1. Key modules in FedRAG and their responsibilities.

4.2. Standard Usage Patterns

Building a RAG System: We first introduce the main classes that FedRAG offers and with which users will often work. We begin with the **core**.`RAGSystem` class, which is comprised of three parts, namely: **knowledge_stores**.`KnowledgeStore`, **retrievers**.`Retriever`, and **generators**.`Generator`. Figure 1 provides a code snippet on how to assemble a `RAGSystem` with FedRAG.

```python
# Flat imports are supported
from fed_rag import (
  RAGSystem,
  RAGConfig,
  HFSentenceTransformerRetriever,
  UnslothFastModelGenerator,
  QdrantKnowledgeStore
)

knowledge_store = QdrantKnowledgeStore()
generator = UnslothFastModelGenerator(
  "unsloth/gemma-3-4b-it",
)
retriever = HFSentenceTransformerRetriever(
  "nthakur/dragon-plus-query-encoder",
  "nthakur/dragon-plus-context-encoder",
)

# Assemble rag_system
rag_system = RAGSystem(
  knowledge_store=knowledge_store,
  generator=generator,
  retriever=retriever,
  rag_config=RAGConfig(top_k=2)
)

# Executes the typical RAG pipeline
response = rag_system.query("What are tulips?")
```

Figure 1. Creating a `RAGSystem` with FedRAG. Not depicted here, but the `retriever` is also used to populate embedded reference chunks into the `knowledge_store`, prior to querying.

RAG Fine-Tuning: After the essential RAG components are constructed, the system can be trained using the fine-tuning abstractions offered by the library. FedRAG provides various **trainers**.`Trainer` classes distinguished by their methodology and which model, generator or retriever, they target. A typical pattern for performing retriever or generator training is provided in Figure 2. There, the `manager` is an orchestrator object that bears the responsibility of preparing the target model for training and ensuring the other model is frozen. Note that both generator and retriever trainer objects also expose a `train()` method that can be called without using `manager` providing an interface similar to that of HuggingFace.

Federated Fine-Tuning: With the centralized fine-tuning pattern established, we show the simple process for converting the previous task to a federated one. This is done by extracting an **fl_tasks**.`FL_task` object from the `manager`. This is demonstrated in Figure 3. Each client has its own fine-tuning dataset and contribute to the tuning process in a decentralized manner and updates are combined with federated averaging (**?**).

Evaluation and Benchmarking: In this final pattern demonstration, we show how benchmarking a RAG system can be achieved. Figure 4 depicts an intuitive benchmarking pattern where an **evals**.`Benchmarker`

```python
... # Keep code from Figure 1
from fed_rag.trainers import (
  HuggingFaceTrainerForRALT,
  HuggingFaceTrainerForLSR,
)
from fed_rag.trainer_managers import (
  HuggingFaceRAGTrainerManager
)
from datasets import Dataset

# Train datasets are examples of (query,
# response) pairs
train_dataset = Dataset.from_dict(
  {
    "query": [...],
    "response": [...]
  }
)
generator_trainer = HuggingFaceTrainerForRALT(
    rag_system=rag_system,
    train_dataset=train_dataset,
)
retriever_trainer = HuggingFaceTrainerForLSR(
    rag_system=rag_system,
    train_dataset=train_dataset,
)
manager = HuggingFaceRAGTrainerManager(
    mode="retriever",  # can be generator
    retriever_trainer=retriever_trainer,
    generator_trainer=generator_trainer,
)

# Train
train_result = manager.train()
```

Figure 2. Fine-tuning a `RAGSystem` with FedRAG.

```python
... # Keep code from Figures 1 and 2
import flwr as fl  # The FL backend
# fl_task
fl_task = manager.get_federated_task()
# Build fl server and client
server = fl_task.server(
  model=retriever_trainer.model
)
client = fl_task.client(
  model=retriever_trainer.model,
  train_dataset=train_dataset,
)
# Spin up client and server using flwr
fl.start_server(server)
fl.start_client(client)
```

Figure 3. Federated fine-tuning of a `RAGSystem` with FedRAG.

runs the desired **`evals`**.`Benchmark` using the chosen **`evals`**.`EvaluationMetric`.

These patterns demonstrate the consistent API design of FedRAG, enabling users to seamlessly transition between RAG system development, central and decentralized fine-tuning, and evaluation with minimal code changes.

4.3. Integrations

In this section, we briefly outline the existing integrations to popular tools and frameworks within the RAG ecosystem. Of the integrations listed in Table 2, only the LlamaIndex

```python
... # Keep code from Figures 1 and 2
import fed_rag.evals.benchmarks as benchmarks
from fed_rag.evals import (
  Benchmarker,
  ExactMatchEvaluationMetric,
)

benchmarker = Benchmarker(rag_system=rag_system)
mmlu = benchmarks.HuggingFaceMMLU(streaming=True)
metric = ExactMatchEvaluationMetric()

# Run benchmark with first 3 examples only
result = benchmarker.run(
  benchmark=mmlu,
  metric=metric,
  is_streaming=True,
  num_examples=3,
  agg="avg",
)
```

Figure 4. Benchmarking with FedRAG.

integration had not been represented in the preceding patterns of Figures 1–4. FedRAG supports a bridge to convert a `RAGSystem` object to a LlamaIndex RAG system equivalent, thus enabling users to leverage their powerful inference features and ecosystem. These integrations allow FedRAG users to leverage existing tools while gaining RAG fine-tuning capabilities, aligning with the *Work With Your Tools* design principle in Section 3.2.

Library	Integration
`HuggingFace`	Generators, retrievers, datasets
`Unsloth`	Fast fine-tuning of generators
`Qdrant`	Storage solution for knowledge
`LlamaIndex`	Bridge to inference object

Table 2. Currently supported integrations in FedRAG.

5. Future Work and Conclusions

In this paper, we introduced FedRAG, a framework for fine-tuning RAG systems across both centralized and federated architectures that offers state-of-the-art fine-tuning methods and fills a critical gap within the RAG ecosystem. In Appendix A, a lightweight experiment is presented. The results confirm that the framework can be used to successfully and flexibly execute RAG fine-tuning tasks. The experimental code and a containerized image of the knowledge store is released with this paper to facilitate reproducibility.

In terms of future development, we have several exciting and impactful additions on the development roadmap. For example, an MCP RAG system and companion MCP knowledge store will soon be integrated into the framework, which will pave the way for studying the effects of adapting RAG systems to knowledge provided by third-party MCP providers. Additional high-priority development items are presented in Table 4 of Appendix B. We are eager to continue the

development of FedRAG and believe that it will enable researchers and practitioners to more easily explore advanced RAG fine-tuning techniques in both centralized and federated settings.

Acknowledgements

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute www.vectorinstitute.ai/partnerships/.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anthropic. Model context protocol, 2024. URL https://github.com/modelcontextprotocol/modelcontextprotocol.

Berant, J., Chou, A., Frostig, R., and Liang, P. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544, 2013.

Blagojevic, V. Enhancing RAG pipelines in Haystack: Introducing diversityranker and lostinthemiddleranker, 2023. https://towardsdatascience.com/enhancing-rag-pipelines-in-haystack-45f14e2bc9f5.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., and et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Chase, H. LangChain, October 2022. URL https://github.com/langchain-ai/langchain.

Chen, M., Li, T., Sun, H., Zhou, Y., Zhu, C., Wang, H., Pan, J. Z., Zhang, W., Chen, H., Yang, F., Zhou, Z., and Chen, W. Research: Learning to reason with search for llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2503.19470.

Chhun, C., Colombo, P., Suchanek, F. M., and Clavel, C. Of human criteria and automatic metrics: A benchmark of the evaluation of story generation. In Calzolari, N., Huang, C.-R., Kim, H., Pustejovsky, J., Wanner, L., Choi, K.-S., Ryu, P.-M., Chen, H.-H., Donatelli, L., Ji, H., Kurohashi, S., Paggio, P., Xue, N., Kim, S., Hahm, Y., He, Z., Lee, T. K., Santus, E., Bond, F., and Na, S.-H. (eds.), *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 5794–5836, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.509/.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., and et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Dige, O., Willes, J., and Emerson, D. B. Evaluating RAG system performance: The impact of knowledge cut-off and fine-tuning. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=2K6I3317QV.

Fajardo, V. A., Liu, J., Markewich, L., Suo, S., Zhang, H., and Desai, S. LlamaIndex Networks, 11 2024. URL https://github.com/run-llama/llama_index/llama-index-networks. Extension for LlamaIndex.

Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.-S., and Li, Q. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6491–6501, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671470. URL https://doi.org/10.1145/3637528.3671470.

Google. Agent2agent (a2a) protocol, 2025. URL https://github.com/google/A2A.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., and et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu,

T. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2), January 2025. ISSN 1046-8188. doi: 10.1145/3703155. URL https://doi.org/10.1145/3703155.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 1(2): 4, 2022.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In Webber, B., Cohn, T., He, Y., and Liu, Y. (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550.

Kumar, K., Ashraf, T., Thawakar, O., Anwer, R. M., Cholakkal, H., Shah, M., Yang, M.-H., Torr, P. H. S., Khan, F. S., and Khan, S. Llm post-training: A deep dive into reasoning large language models, 2025. URL https://arxiv.org/abs/2502.21321.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Lin, S.-C., Asai, A., Li, M., Oguz, B., Lin, J., Mehdad, Y., Yih, W.-t., and Chen, X. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. *arXiv preprint arXiv:2302.07452*, 2023a.

Lin, X. V., Chen, X., Chen, M., Shi, W., Lomeli, M., James, R., Rodriguez, P., Kahn, J., Szilvasy, G., Lewis, M., et al. Ra-dit: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2023b.

Liu, J. LlamaIndex, November 2022. URL https://github.com/jerryjliu/llama_index.

Ma, X., Gong, Y., He, P., Zhao, H., and Duan, N. Query rewriting in retrieval-augmented large language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5303–5315, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.322. URL https://aclanthology.org/2023.emnlp-main.322.

Ovadia, O., Brief, M., Mishaeli, M., and Elisha, O. Fine-tuning or retrieval? comparing knowledge injection in llms, 2024. URL https://arxiv.org/abs/2312.05934.

Paul, D., Ismayilzada, M., Peyrard, M., Borges, B., Bosselut, A., West, R., and Faltings, B. REFINER: Reasoning feedback on intermediate representations. In Graham, Y. and Purver, M. (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1100–1126, St. Julian's, Malta, March 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.eacl-long.67/.

Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., and Tang, S. Graph retrieval-augmented generation: A survey, 2024. URL https://arxiv.org/abs/2408.08921.

Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., and Shoham, Y. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023. doi: 10.1162/tacl_a_00605. URL https://aclanthology.org/2023.tacl-1.75.

Ren, J., Zhao, Y., Vu, T., Liu, P. J., and Lakshminarayanan, B. Self-evaluation improves selective generation in large language models. In Antorán, J., Blaas, A., Buchanan, K., Feng, F., Fortuin, V., Ghalebikesabi, S., Kriegler, A., Mason, I., Rohde, D., Ruiz, F. J. R., Uelwer, T., Xie, Y., and Yang, R. (eds.), *Proceedings on "I Can't Believe It's Not Better: Failure Modes in the Age of Foundation Models" at NeurIPS 2023 Workshops*, volume 239 of *Proceedings of Machine Learning Research*, pp. 49–64. PMLR, 16 Dec 2023. URL https://proceedings.mlr.press/v239/ren23a.html.

Sani, L., Iacob, A., Cao, Z., Lee, R., Marino, B., Gao, Y., Cai, D., Li, Z., Zhao, W., Qiu, X., and Lane, N. D. Photon: Federated llm pre-training, 2024. URL https://arxiv.org/abs/2411.02908.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023a. URL https://openreview.net/forum?id=1PL1NIMMrw.

Wang, Y., Lipka, N., Rossi, R. A., Siu, A., Zhang, R., and Derr, T. Knowledge graph prompting for multi-document question answering, 2023b. URL https://arxiv.org/abs/2308.11730.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Zhang, T., Patil, S. G., Jain, N., Shen, S., Zaharia, M., Stoica, I., and Gonzalez, J. E. Raft: Adapting language model to domain specific rag. In *First Conference on Language Modeling*, 2024.

Zhuang, S., Liu, B., Koopman, B., and Zuccon, G. Open-source large language models are strong zero-shot query likelihood models for document ranking. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 8807–8817, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.590. URL https://aclanthology.org/2023.findings-emnlp.590.

A. Example: RA-DIT

In their work, Lin et al. (2023b) conducted various experiments studying the effectiveness of RALT and LSR fine-tuning methods. Their experiments revealed that applying RALT or LSR individually leads to performance gains, but the greatest gain comes after applying both RALT and LSR in succession. They termed this combination of fine-tuning techniques Retrieval-Augmented Dual Instruction Tuning (RA-DIT). In order to illustrate the potential of the FedRAG framework, we aim to reproduce a lightweight version of their experiments. The rest of this appendix outlines the RAG system specifications, details on fine-tuning as well as evaluation setup, and finally the results of the experiments.

A.1. RAG System

A.1.1. Knowledge Store & Retriever

We use text chunks from the December 2021 Wikipedia dump released by Izacard et al. (2022). This release includes two files, `infobox.jsonl` and `text-list-100-sec.jsonl`, which can be downloaded from the `facebookresearch/atlas` GitHub repository.[3] For the knowledge store, we use the first 10M text passages provided in `text-list-100-sec.jsonl` with no further preprocessing with the exception of concatenating the `title`, `section`, and `text` fields for each passage.

For the retriever, we use DRAGON+ (Lin et al., 2023a). More specifically, we use the `SentenceTransformer` versions of this dual-encoder model available on HuggingFace.[4,5] The context encoder of DRAGON+ is used to encode the 10M text chunks prior to loading the embeddings into the knowledge store.

A.1.2. Generator

For the generator LLM, we use a quantized (4-bit) Llama2-7B (Touvron et al., 2023). We specifically use the official version of this model available on HuggingFace,[6] with the `load_in_4bit` parameter set to `True`.

A.2. Fine-tuning & Evaluation

For the fine-tuning dataset, we use Web Questions (Berant et al., 2013) available on HuggingFace.[7] We apply QLoRA (Dettmers et al., 2023) fine-tuning with only the RALT objective by making use of a **`trainers`**`.HuggingFaceTrainerForRALT` object and supplying it a **`generators`**`.HFPeftModelGenerator`. The latter is used to load a `PeftModel` available on HuggingFace.[8]

For evaluation, we use the `test` split of the `global_facts` subset of MMLU (Hendrycks et al., 2020) available on HuggingFace, which has exactly 100 data points.[9] Similar to Lin et al. (2023b), we apply 5-shot in-context learning for each evaluation example. The few-shot examples are randomly drawn from the `validation` split and held fixed throughout evaluation. For performance measurement, we use the exact match metric.

A.3. Results

We report the results of two separate fine-tuning runs in Table 3. For comparison, we also report the performance of the same RAG system but without any fine-tuning applied at all.

The results of our lightweight experiment corroborate the findings of Lin et al. (2023b) and align with results from other work (Zhang et al., 2024; Chen et al., 2025). That is, RAG fine-tuning can lead to significant performance gains. Note that the observed variability in runs is due to the sampling parameters used for generation.

[3] https://github.com/facebookresearch/atlas
[4] https://huggingface.co/nthakur/dragon-plus-query-encoder
[5] https://huggingface.co/nthakur/dragon-plus-context-encoder
[6] https://huggingface.co/meta-llama/Llama-2-7b-hf
[7] https://huggingface.co/datasets/stanfordnlp/web_questions
[8] https://huggingface.co/Styxxxx/llama2_7b_lora-quac
[9] https://huggingface.co/datasets/cais/mmlu

Method	Run 1	Run 2	Average
Without fine-tuning	17.0	27.0	22.0
RALT fine-tuning	**27.0**	**34.0**	**30.5**

Table 3. RA-DIT inspired experiment demonstrating the effect of RALT fine-tuning on exact match performance for the MMLU (`global_facts`) benchmark.

B. Development Roadmap

In this section, we provide a portion of our development roadmap that includes the high-priority items, which we deem to be highly impactful for our users.

Item	Description
MCP knowledge store	An MCP client that can be used to receive knowledge context from third-party MCP servers.
MCP RAG system	A specialized RAG system class that is able to retrieve knowledge from the MCP Knowledge Store and subsequently supply it to the generator.
ReSearch generator trainer	An implementation of ReSearch (Chen et al., 2025), i.e., equipping generator LLMs with reasoning infused with search.
Improved retriever trainers	New training objectives for language model supervised retriever fine-tuning.
Advanced federated learning	Support for more advanced federated learning techniques.
LangChain integration	Bridge to LangChain inference objects.
General optimizations	Optimizations for batch RAG system querying and concurrent benchmarking.

Table 4. Development roadmap: high-priority items for FedRAG.