Efficient Federated Search for Retrieval-Augmented Generation using Lightweight Routing

Akash Dhasade[0000−0003−4362−5548], Rachid Guerraoui[0000−0002−4794−8902], Anne-Marie Kermarrec[0000−0001−8187−724X], Diana Petrescu[0009−0006−2229−235X] ⋆, Rafael Pires[0000−0002−7826−1599], Mathis Randl[0009−0003−6844−4695], and Martijn de Vos[0000−0003−4157−4847]

EPFL, Lausanne, Switzerland

Abstract. Large language models (LLMs) achieve remarkable performance across domains but remain prone to hallucinations and inconsistencies. Retrieval-augmented generation (RAG) mitigates these issues by augmenting model inputs with relevant documents retrieved from external sources. In many real-world scenarios, relevant knowledge is fragmented across organizations or institutions, motivating the need for federated search mechanisms that can aggregate results from heterogeneous data sources without centralizing the data. We introduce RAGRoute, a lightweight routing mechanism for federated search in RAG systems that dynamically selects relevant data sources at query time using a neural classifier, avoiding indiscriminate querying. This selective routing reduces communication overhead and end-to-end latency while preserving retrieval quality, achieving up to 80.65% reductions in communication volume and 52.50% reductions in latency across three benchmarks, while matching the accuracy of querying all sources.

Keywords: Retrieval-Augmented Generation · Large Language Models · Federated Search · Resource Selection · Routing.

1 Introduction

Large language models (LLMs) have driven significant advancements across various domains such as natural language processing and healthcare [5,24,18]. Despite their widespread adoption, one major concern is their tendency to *hallucinate*, generating false responses with high confidence [20] and limiting their applicability in critical domains [21]. Retrieval-augmented generation (RAG) mitigates this issue by combining text generation with external retrieval, enhancing factual accuracy and contextual grounding [27,34].

Existing RAG systems typically rely on a single monolithic vector database [26]. In practice, however, real-world knowledge is often distributed across multiple heterogeneous information systems and repositories [6,39]. This setting calls for

⋆ Corresponding author: diana.petrescu@epfl.ch

federated search, where queries are executed across multiple independent data sources and the results are merged into a unified ranking [36]. In RAG systems operating over multiple repositories, federated search constitutes the retrieval layer: queries are dispatched to selected data sources, their results are aggregated and reranked, and the resulting context is passed to the language model for generation. This avoids centralizing data, which might be complicated due to regulatory constraints or privacy considerations [8,23], and enables organizations to reuse existing infrastructure, therefore reducing operational and storage overhead.

A key challenge in federated search is resource selection [28,39], i.e., identifying which sources should be queried. Yet many RAG pipelines query all available resources [39]. Indiscriminate querying increases communication and computation costs [15] and may introduce irrelevant context that exacerbates hallucinations [7,10].

We introduce RAGRoute, a novel and efficient routing mechanism for federated search in RAG systems that dynamically selects relevant data sources at query time using a lightweight neural network. By avoiding unnecessary queries, RAGRoute significantly reduces resource consumption and end-to-end latency while maintaining high search quality. We evaluate RAGRoute on three benchmarks: MIRAGE [42], MMLU [19] and FeB4RAG [39]. Our results show that RAGRoute achieves up to 89.70% recall in source selection, reduces retrieval communication volume by up to 80.65%, and lowers end-to-end latency by up to 52.50%, while matching the accuracy of querying all data sources. This improvement stems primarily from reducing the number of documents that must be reranked during retrieval, thereby alleviating a major computational bottleneck in RAG pipelines.

In summary, our contributions are as follows:

1. We propose and implement RAGRoute, a lightweight and effective routing mechanism for federated search in RAG that dynamically selects data sources per query, and make our code publicly available.[1]
2. We conduct extensive evaluations on three benchmarks, demonstrating that RAGRoute significantly reduces communication overhead and latency while maintaining high retrieval quality and end-to-end accuracy.

2 Background and problem description

2.1 Retrieval-augmented generation (RAG)

RAG enhances the reliability of LLM responses by integrating external information as part of the input (or *prompt*) [27]. In a typical RAG pipeline, documents are split into chunks and encoded into dense vector embeddings, which are stored in a vector database that supports similarity search. For simplicity, we refer to document chunks as documents throughout this paper. Given a user

[1] See https://github.com/sacs-epfl/ragroute.

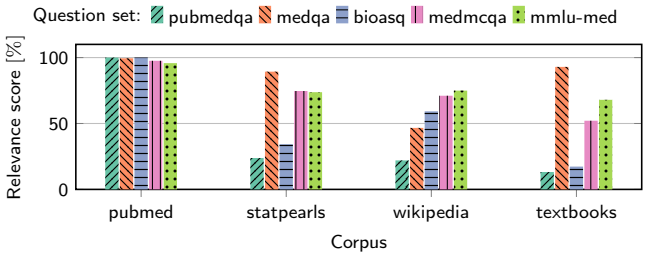

Fig. 1. The relevance of different corpora in RAG when answering questions, using question sets from the MIRAGE benchmark.

query, the query is embedded using a compatible query encoder and a nearest-neighbor search is performed to retrieve the most relevant chunks. This search is often accelerated using approximate nearest neighbor (ANN) indexing [29]. The retrieved candidates are commonly reranked and appended to the original query to form an augmented prompt for the LLM. By grounding generation in retrieved evidence, RAG reduces hallucinations and improves factual accuracy without requiring model retraining. Most existing RAG systems assume a single centralized vector database. In contrast, we consider settings where knowledge is distributed across multiple independent data sources, motivating federated search and resource selection.

2.2 Towards federated search in RAG

Federated search is an information retrieval setting in which a query is executed across multiple independent data sources and the results are aggregated without centralizing the underlying data [36]. A key challenge in federated search is resource selection, i.e., determining which data sources should be queried [39]. In RAG systems operating over multiple repositories, source relevance varies substantially across queries, making accurate relevance estimation essential for efficient retrieval.

We empirically show this by analyzing data source relevance using corpora and questions from the MIRAGE benchmark (more details can be found in Section 4.1). This benchmark contains a large number of medical multiple-choice questions and answers and is divided into five question sets [42]. As knowledge backend for RAG we use four different data sources (corpora), namely PubMed, StatPearls, Wikipedia and Textbooks. For each question, we determine which corpora are relevant by considering a corpus relevant if at least one document originating from that corpus appears in the top-15 retrieved results.

Figure 1 shows the overall relevance of different corpora, highlighting how corpus usefulness varies across question sets. For example, the bar corresponding to the MedQA question set and the StatPearls corpus shows a relevance score of 89.32%, meaning that for 89.32% of queries in MedQA, at least one document in the retrieved relevant documents originates from StatPearls. While

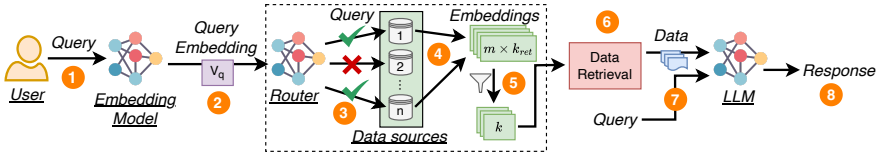

Fig. 2. The workflow of RAGRoute. The components specific to RAGRoute are indicated in the box with the dashed border.

some corpora, such as PubMed, consistently provide valuable, relevant information for all question sets, relying on a single corpus is often insufficient. Indeed, results from [42] demonstrate that combining multiple corpora improves retrieval performance. Some corpora, such as StatPearls or Wikipedia, are only useful in particular cases. The Textbooks corpus, for example, is mostly irrelevant for the PubMedQA question set. The differences in corpus relevance motivate the importance of adequate resource selection for a given query.

One must strike a balance in the number of data sources being queried. While querying all available data sources guarantees full coverage, it also increases retrieval latency and computational overhead, as more requests, database searches, and document reranking operations are required. At the same time, under-selecting data sources risks missing critical information, particularly in domains where information is distributed sparsely across multiple repositories, for example, government data that resides in different portals [9]. Achieving a good trade-off between retrieval efficiency and response quality remains an open problem. Therefore, this work answers the following question: *how can we efficiently predict query-specific source relevance in federated search for RAG, while minimizing retrieval overhead?*

3 Design of RAGRoute

3.1 System model and assumptions

We assume a permissioned setting in which all data sources are known and trusted. Thus, we focus on federated search within an enterprise or institutional consortium. Each data source maintains its own data and associated local embeddings, and is responsible for computing and storing vector representations of its documents. The specifications of the embedding models used could vary across data sources. We assume that the enterprise or institution running the system has access to the embedding models used by the different data sources. User queries are submitted in natural language and converted into embedding vectors using the appropriate model(s). While we assume data sources are generally available, RAGRoute remains functional even if some sources are temporarily offline. The system can simply exclude the offline data source from selection, ensuring graceful degradation in retrieval coverage rather than system failure.

3.2 RAGRoute workflow

We visualize the RAGRoute workflow in Figure 2, enabling RAG-enhanced LLM responses by retrieving documents from n distinct data sources. We show the components specific to RAGRoute in the dashed box. When a user sends a query to the system ❶, the user query is first converted into one or more embeddings using all embedding models used by the data sources ❷. These query embeddings are then forwarded to a *router*, whose purpose is to decide which of the n data sources are relevant. The router predicts this relevance separately for each source using source specific features and query embeddings. We outline the design and operation of our router in Section 3.3.

After determining the m relevant data sources ($m \leq n$), we forward to each selected source the query embedding previously computed using its respective embedding model ❸. For example, Figure 2 shows that data source 1 is selected and receives the query embedding, while source 2 is skipped. Each selected source uses the received query embedding to retrieve top-k_{ret} documents most similar to the query. This results in $m \times k_{ret}$ total retrieved documents ❹. This is followed by a post-retrieval reranking step, which has become a standard component in modern RAG systems [14,33,32]. Rerankers, often implemented as cross-encoder models, rescore the retrieved candidates to prioritize the passages most semantically relevant to the user query. This two-stage retrieval design improves grounding accuracy while also reducing response latency, since only a compact set of highly relevant documents is passed to the language model. After reranking, only final k documents are retained where the value of k can differ from the retrieval cutoff k_{ret} (see Section 4).

Finally, the relevant documents ❻ and original user query ❼ are combined into a single prompt. This prompt is fed to the LLM and a response is generated and returned to the user ❽, therefore completing the query.

3.3 Lightweight query routing

To enable resource selection and efficient retrieval across multiple data sources, RAGRoute uses a lightweight query router, implemented as a shallow neural network (NN) with a few fully connected layers. This minimal design is intentional: the router must remain computationally inexpensive so that routing overhead is negligible compared to retrieval and generation. Despite its simplicity, the router is sufficient to estimate the relevance of each data source before retrieval. Using a shallow NN is inspired by practices in mixture of experts (MoE) models and ensembles. MoE models leverage a small router function to decide which subset of experts to activate [44]. Similarly, shallow NNs are used for decision-making in one-shot federated ensembles [3]. This work applies similar ideas to selecting relevant data sources in federated search for RAG systems. We next describe the training and inference phase of the RAGRoute router.

Training phase Let $R = \{R_1, R_2, \ldots, R_n\}$ denote the set of n data sources, where each R_i corresponds to a collection of documents. For each query–source

pair (q, R_i), the router estimates a relevance probability, which is thresholded to obtain a binary routing decision. The router is trained using ground-truth binary relevance labels $s(q, R_i) \in \{0, 1\}$, where $s(q, R_i) = 1$ indicates that source R_i is relevant to query q. We first describe how these ground-truth labels are constructed, before detailing the router input features. We consider two approaches for constructing the source-level relevance labels.

1. **Rerank based** – For a given query q, the top-k_{ret} documents are retrieved from each source $R_i \in R$. All retrieved documents are then reranked jointly using a neural reranker to produce a global top-k list. A source R_i is labeled as relevant if at least one retrieved document d from R_i appears in the global top-k:
$$s(q, R_i) = \begin{cases} 1 & \text{if } \exists\ d \in R_i \text{ s.t. } d \text{ is in the global top-}k \\ 0 & \text{otherwise.} \end{cases}$$
These labels depend on both the embedding model and the value of k. During inference, we use the same value of k as for training.
2. **LLM based** [39] – For each query q, we first retrieve the the top-k_{ret} documents from each source $R_i \in R$. We then obtain query–document relevance judgments for these retrieved documents using an external LLM that is independent of the embedding model. Each document is assigned one of four labels: not relevant, minimally relevant, highly relevant, or key, where key indicates a strong match. For each source R_i, we aggregate the LLM judgments of its retrieved documents into a graded precision score:
$$\text{Graded Precision}(q, R_i) = \frac{\sum_{j=1}^{k} w(q, d_j)}{k} \times 100$$
where d_j denotes the j-th retrieved document from source R_i for query q, and $w(q, d_j)$ is defined as:
$$w(q, d_j) = \begin{cases} 0 & \text{if not relevant} \\ 0.25 & \text{if minimally relevant} \\ 0.5 & \text{if highly relevant} \\ 1 & \text{if key.} \end{cases}$$
Finally, a source is labeled as relevant if the Graded Precision score is positive:
$$s(q, R_i) = \begin{cases} 1 & \text{Graded Precision}(q, R_i) > 0 \\ 0 & \text{otherwise.} \end{cases}$$
These labels depend only on the cutoff k_{ret} and not on the embedding model.

Feature selection. While it is common to assume that each source uses the same embedding model to embed its documents [13], some specialized sources may instead employ their own embedding model [39]. We design our router

to support the more general scenario where each source could have its own embedding model. Let $H_i(x) \in \mathbb{R}^{z_i}$ denote the embedding of any input x (a query or a document) using the embedding model of source R_i, where z_i is its embedding dimension. The router takes the following three features as input:

(i) the query embedding $H_i(q)$,
(ii) the centroid of the data source $C_i = \frac{1}{|R_i|} \sum_{d \in R_i} H_i(d)$, and
(iii) the source-id as one hot encoded vector Id_i.

The centroid C_i, computed as the average vector representation of all document embeddings in a data source, summarizes its overall semantic content. The source-id serves as a prior signal to help the router account for systematic differences across sources. Since the size of the embedding z_i may differ across sources, we consider the highest embedding size $z = \max_{i \in [n]} z_i$ and pad zeros if $z_i < z$. We denote the padded query embedding as $\hat{H}_i(q)$ and the padded centroid as $\hat{C}_i$, where $\hat{H}_i(q), \hat{C}_i \in \mathbb{R}^z$. The router parameterized by θ and denoted by f_θ independently predicts a relevance probability for each source $i \in [n]$ based on these features. Given a dataset of queries $\mathcal{D}_{\text{train}}$ with ground truth relevance labels constructed as discussed before, the router is trained to minimize the following objective:

$$\mathcal{L}(\theta) = \sum_{q \in \mathcal{D}_{\text{train}}} \sum_{i=1}^{n} \ell \left(f_\theta\big(\hat{H}_i(q), \hat{C}_i, \mathrm{Id}_i\big),\ s\big(q, R_i\big) \right) \tag{1}$$

where ℓ is a binary classification loss, such as binary cross-entropy.

Inference phase Once trained, RAGRoute uses this model to efficiently route incoming user queries to relevant data sources. We run one forward pass for each of the available data sources individually to predict their relevance to a given inference query. This forward pass completes quickly (with sub-millisecond latency, see Section 4.3) and can be done in parallel for individual sources. Additionally, multiple queries can be batched into a single forward pass, depending on their arrival time. When new data sources are added or existing ones are updated, RAGRoute regenerates the training ground truth by querying the affected sources together with those predicted as relevant by the existing router. This targeted querying strategy ensures that new and updated sources are incorporated into the label construction process while avoiding unnecessary queries to unrelated ones, thereby minimizing update overhead. Because the router is implemented as a shallow NN with only a few fully connected layers, retraining is highly lightweight, requiring minimal computation and storage. Thus, the router can be rapidly retrained in the background whenever updates occur.

4 Evaluation

4.1 Experimental setup

Implementation. We implement RAGRoute in Python using an event-driven architecture based on `asyncio`. Each core component (coordinator, router, data

sources, and LLM engine) runs as an independent process to enable modularity and parallel execution. The coordinator orchestrates asynchronous communication across components. We use the ZeroMQ library for inter-process messaging and AIOHTTP to handle incoming HTTP queries. We use the Ollama framework for inference which provides a convenient way to load and run inference with different LLMs [31]. For the embedding models, we use the PyTorch library.

Router model. We implement the router as a lightweight fully connected NN. The network consists of hidden layers with 128, 64, and 32 neurons, each followed by Layer Normalization, ReLU activation, and Dropout to improve stability and prevent overfitting. These hyperparameters were selected through cross-validation, where we evaluated several architectures with varying numbers of layers and hidden dimensions on the validation set. The output layer consists of a single neuron that produces a raw logit score, predicting whether the corpus is relevant to the given query. The model is trained using Binary Cross-Entropy with Logits Loss with a positional weight to address class imbalance. We use a cyclic scheduler for the learning rate γ, oscillating γ between 0.001 and 0.005. Model performance is evaluated on the validation set after each epoch, and the best model is selected based on validation accuracy. Training data are split by question into 30%/10%/60% train/validation/test partitions, and all input features are standardized using a StandardScaler. The router's small size ensures fast training, negligible inference overhead, and ease of retraining when data sources evolve. We also tested alternative classifiers (*e.g.*, logistic regression and random forests) but found the shallow NN performed best overall.

Datasets. We evaluate RAGRoute with the following three benchmarks:

(i) **MIRAGE** is a benchmark designed to evaluate RAG systems for medical question answering [42]. It consists of 7663 questions drawn from five widely used medical QA datasets. We use MedRAG as knowledge source, which includes four corpora with documents related to healthcare [42]. For generating embeddings, we use MedCPT [22], a domain-specific model designed for biomedical contexts. For retrieval, we use the IndexFlatL2 index structure, provided by the FAISS library [12], ensuring exact search and eliminating sources of approximation in our experiments. We treat each corpus as a separate data source. For MIRAGE, we construct the ground truth relevance labels using the rerank based approach. To run RAGRoute with a RAG pipeline, we leverage the code provided by the MedRAG toolkit.

(ii) **MMLU** is a benchmark that evaluates LLM systems across tasks ranging from elementary mathematics to legal reasoning [19]. For our experiments, we use eight subject-specific subsets of MMLU with a total of 2803 questions. As a knowledge source, we use a Wikipedia dataset [25]. From this dataset, we cluster the documents into ten groups using the k-means algorithm to simulate different data sources. After clustering, we observe variance in the cluster size, ranging from 1.41 M to 2.88 M vectors per cluster. For MMLU, we construct the ground truth relevance labels using the rerank based ap-

proach. To run MMLU, we leverage the code provided by the RQABench framework [37].

(iii) **FeB4RAG** is a benchmark designed to evaluate federated search methods for RAG systems [39]. It consists of 790 user queries spanning diverse domains and complexity levels. FeB4RAG is derived from BEIR [38] and includes 13 heterogeneous data sources powered by eight distinct embedding models, enabling evaluation under realistic federated retrieval settings. For FeB4RAG, ground truth relevance labels are obtained using the LLM based approach. Unlike MIRAGE and MMLU, FeB4RAG does not provide verifiable ground-truth answers.

Evaluation. To facilitate automated evaluation, we developed a separate benchmarking script that iterates over all questions in a given dataset and sends each query, along with the associated answer choices if applicable, to the RAGRoute system via HTTP requests. Queries are sent one by one: the script waits for the response to a given query before proceeding to the next. Upon receiving a response from the system, the script verifies the correctness of the answer against the ground truth answer, if applicable. This setup enables systematic and reproducible evaluation across multiple benchmarks.

Retrieval and reranking. For all datasets, we retrieve a global top-$k = 15$ list of documents for generation. To construct this final set, each selected data source retrieves top-$k_{ret} = 50$ documents most similar to the query using exact similarity search with FAISS [12] incurring negligible latency compared to the reranker. We set k_{ret} sufficiently large to ensure high recall, while balancing the trade-off with reranking cost. All retrieved candidates are then reranked to produce the global top-15 list. This two-stage retrieval strategy balances recall and precision: a sufficiently large retrieval pool ensures coverage, while reranking improves semantic relevance and reduces noise in the final context. We employ the `BAAI/bge-reranker-v2-m3` model [1], a lightweight cross-encoder reranker designed for multilingual reranking with efficient inference.

LLM models. As LLM, we use the open-source LLaMA 3.1 8B Instruct model for all above datasets [17] as it is commonly considered in related work [2,35]. We adopt a zero-shot chain-of-thought prompting scheme, instructing the model to reason step-by-step before providing the final answer. The output is formatted in JSON to ensure interpretable reasoning and structured evaluation.

Hardware. We run our experiments on a compute cluster equipped with an NVIDIA A100 GPU for LLM answer generation, and 500 GB of main memory.

Routing baselines. To analyze the effectiveness and efficiency of RAGRoute, we experiment with the following four routing strategies.

(i) **none.** This routing strategy does not query any data source and the input prompt to the LLM is not enhanced with retrieved documents.

(ii) **all.** Under this routing strategy, all data sources are queried. This can be considered as a naive baseline for federated search that lacks a mechanism for strategic resource selection.

(iii) **RAGRoute.** This routing strategy uses the RAGRoute router to identify and retrieve documents only from relevant data sources.

Table 1. Classification metrics (averages) for our router and for different benchmarks. RAGROUTE router achieves high accuracy and recall, demonstrating good generalization across benchmarks.

Benchmark	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)
MIRAGE	86.63	86.79	83.35	84.96	92.94
MMLU	90.93	71.64	82.92	76.87	95.77
FEB4RAG	83.05	87.37	89.70	88.51	84.00

(iv) **RANDOM.** This routing strategy randomly selects a fixed number of data sources, matching the number selected by RAGROUTE but without using relevance predictions. We incorporate this strategy to evaluate the effectiveness of RAGROUTE beyond extremes like querying all or no data sources. This allows us to ignore the effect of the number of contacted sources, isolating the impact of how the sources are selected, demonstrating that RAGROUTE gains arise from intelligent, query-aware routing.

Metrics. Our experiments primarily focus on the classification performance of the RAGROUTE router model and the system efficiency of the entire RAGROUTE system. For the former, we report standard classification metrics such as accuracy, recall, precision, F1-Score and AUC. For the latter, we monitor, for each user query, relevant system metrics such as the number of data sources contacted, communication volume and latency. We also determine the end-to-end RAG accuracy for the MMLU and MIRAGE benchmarks. We are unable to do so for FEB4RAG since this benchmark does not provide ground-truth answers.

4.2 RAGROUTE routing effectiveness

We evaluate the effectiveness of our router and show its classification performance in predicting data source relevance for a given query in the test set for each benchmark. Table 1 presents various classification metrics, *i.e.*, accuracy, precision, recall, F1-score, and AUC, for all three benchmarks. Here, recall measures the router's ability to identify all relevant data sources, while accuracy reflects the overall correctness of the router's binary predictions (and not the end-to-end LLM accuracy in generating final responses).

We achieve consistently strong results across all benchmarks, with accuracy ranging from 83.05% for FEB4RAG to 90.93% for MMLU, and recall values between 89.70% and 82.92% respectively, indicating that the router reliably identifies relevant data sources across diverse settings. The slightly lower accuracy on some benchmarks primarily stems from our design choice to favor recall which is particularly important for imbalanced datasets where only a few sources are relevant per query. This trade-off is desirable in federated retrieval settings, where missing a relevant data source is typically more detrimental to the quality of LLM answers than querying an additional one. Overall, the RAGROUTE router demonstrates strong and balanced generalization across benchmarks, confirming its effectiveness for real-world federated search in RAG systems.

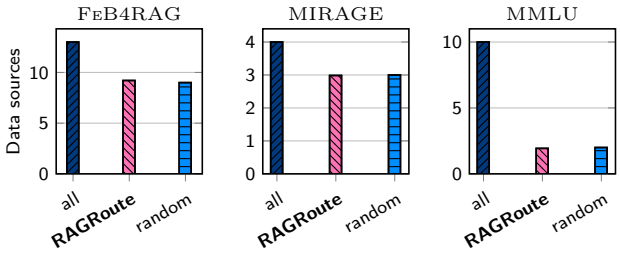

Fig. 3. The average number of queries (data sources contacted) for all benchmarks and for different routing strategies. RAGRoute significantly reduces the number of contacted data sources compared to ALL baseline.

4.3 RAGRoute efficiency gains

We now quantify the reduction by RAGRoute in the number of data sources contacted and communication volume related to document retrieval for all routing baselines. We also analyze the end-to-end RAG accuracy and provide a time breakdown of different operations in the RAG workflow.

Number of data sources contacted Figure 3 shows the total number of data sources contacted across all queries, for all routing strategies and benchmarks. We find that the number of data sources contacted for the RAGRoute routing strategy is always lower compared to querying all data sources (which is the ALL routing strategy). This effect is the most pronounced on the MMLU dataset, where the number of contacted data sources decreases from 16 810 to 3250, representing an *80.67% reduction* in the number of messages exchanged for document retrieval. In other words, under the RAGRoute routing strategy, only 1.93 out of ten data sources are contacted on average. On average, RAGRoute contacts 2.98 out of four data sources per query on MIRAGE, and 9.20 out of thirteen on FeB4RAG. These results highlight the effectiveness of RAGRoute in minimizing communication and computation overhead during federated retrieval, while maintaining high routing accuracy.

Communication volume We next show the reduction in communication volume achieved by querying only relevant data sources. By selecting a subset of sources predicted as relevant, RAGRoute significantly decreases the communication volume between the coordinator and data sources. Figure 4 (left column) reports the total communication volume for the three routing strategies: ALL, RAGRoute, and RANDOM. Compared to the ALL baseline, RAGRoute reduces total communication volume by 19.94% (1955.4 MiB → 1565.4 MiB) on MIRAGE, by 80.65% (554.4 MiB → 107.3 MiB) on MMLU, and by 32.52% (905.3 MiB → 610.9 MiB) on FeB4RAG. The RANDOM baseline, which queries a similar number of sources as RAGRoute, also lowers communication volume compared to ALL, but at the cost of lower accuracy, as shown next.

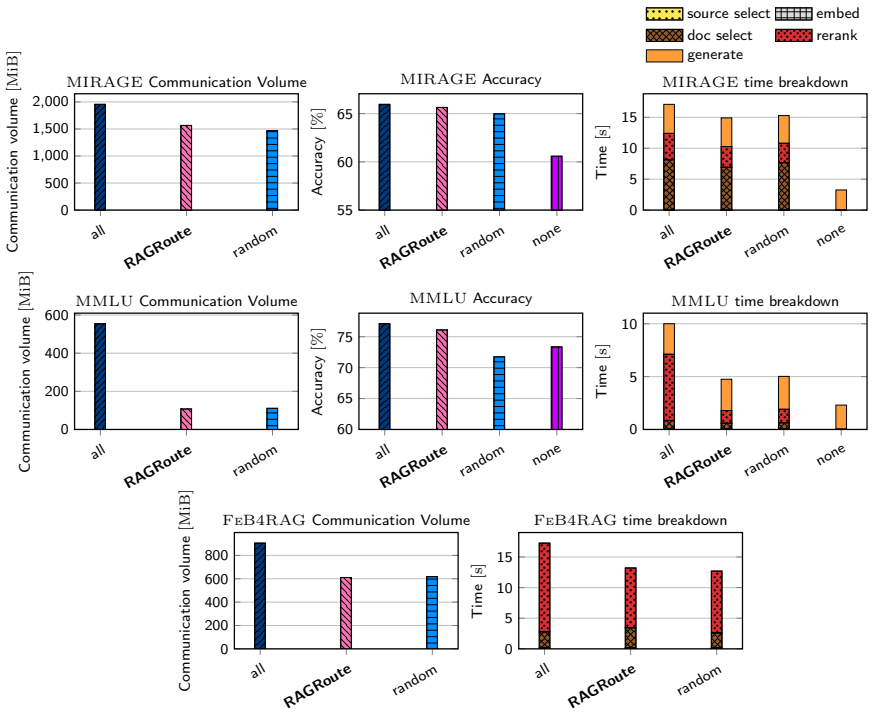

Fig. 4. The communication volume required for document retrieval (left), RAG test accuracy (middle) and query time breakdown (right), for all routing strategies and benchmarks. RAGRoute consistently reduces communication volume and maintains near-optimal accuracy, with improved query latency.

End-to-end RAG accuracy Finally, we report the average end-to-end accuracy on the MIRAGE and MMLU benchmarks. For the FeB4RAG benchmark, this metric is omitted, as no ground-truth answers are available to evaluate the correctness of generated outputs.

Figure 4 (middle column) shows the end-to-end accuracy results. On the MMLU benchmark, RAGRoute achieves an average accuracy of 76.09%, nearly matching the ALL baseline (77.10%). In contrast, the RANDOM baseline reaches only 71.74%, underperforming even the NONE baseline (73.35%). This drop illustrates the negative impact of indiscriminate retrieval: introducing irrelevant or noisy documents can distract the language model and ultimately reduce answer quality. By contrast, RAGRoute 's selective routing ensures that documents only from relevant sources contribute to the answer quality.

On the MIRAGE benchmark, the NONE baseline achieves 60.58% accuracy, while randomly querying data sources increases it to 64.98%. When the data sources are selected via RAGRoute, accuracy further improves to 65.64%,

closely matching the ALL baseline at 66.96%. Overall, RAGROUTE maintains nearly optimal RAG accuracy while greatly reducing communication volume.

Time breakdown We further explore the efficiency gains of RAGROUTE and provide for each dataset a time breakdown when answering a user query. These results are shown in Figure 4 (right column) for each of the routing baselines, and we measure the time spent in the following five components. SOURCE SELECTION refers to the time spent on the inference request of the RAGROUTE router, which predicts the set of relevant data sources for a given query. EMBEDDING denotes the time required to compute the query embeddings using the appropriate embedding models for all data sources. DOC SELECTION measures the time elapsed from when the coordinator dispatches the query to the selected data sources until all retrieval results are received. This includes network communication, client-side retrieval, and result transmission. RERANK represents the time consumed by the reranker to rescore and reorder the retrieved candidates, producing the global top-k set of documents. Finally, GENERATE corresponds to the time required by the LLM to synthesize the final response given the reranked context documents and user query. The SOURCE SELECTION, EMBEDDING, DOC SELECTION, and RERANK times are zero for the NONE routing baseline, as no data sources are queried in this configuration. For the FEB4RAG benchmark, we report only the SOURCE SELECTION, EMBEDDING, DOC SELECTION, and RERANK times, since no ground-truth answers are available to evaluate the GENERATE phase.

We first observe that the end-to-end query latency of RAGROUTE varies across datasets, reflecting differences in corpus scale. When using the RAGROUTE routing strategy, queries complete on average in 14.91 s on MIRAGE and 4.75 s on MMLU. These variations are explained by two factors. First, MIRAGE contains larger and more textually rich corpora, increasing the time needed to fetch and process documents. Second, the number of tokens included in the LLM input prompt is approximately twice as high for MIRAGE compared to MMLU, resulting in longer generation times (4.63 s vs. 2.96 s). For FEB4RAG, the query answer latency averages around 13.21 s, but this value excludes generation time because no verifiable ground-truth answers are available for this dataset.

An interesting observation is that reranking is a major contributor to overall latency. For example, in the ALL baseline, reranking alone accounts for 6.27 s on MMLU (around 62.64% of total latency) and 14.49 s on FEB4RAG (around 83.90% of the retrieval pipeline, excluding generation), showing that the cross-encoder reranker can become a major computational bottleneck in RAG systems operating over multiple data sources, as the number of considered sources grows. In contrast, RAGROUTE effectively mitigates this bottleneck by fetching documents from fewer sources, effectively reducing the number of candidate documents that must be reranked. With RAGROUTE, reranking time decreases by 80.70% on MMLU (from 6.27 s to 1.21 s) and by 32.51% on FEB4RAG (from 14.49 s to 9.78 s). Importantly, this reranking cost reduction directly translates into faster end-to-end query execution, without compromising retrieval accuracy or grounding quality.

Table 2. Ablation study of router's input features. Using all three features results in the highest performance on both benchmarks.

query	centroid	source-id	Recall	
			MIRAGE	FeB4RAG
✓	✓	✗	81.53	89.07
✓	✗	✓	82.33	88.33
✓	✓	✓	**83.35**	**89.70**

Meanwhile, the latency overhead of pre-retrieval components (embedding generation and routing inference) is almost negligible. Even for FeB4RAG, where the query must be embedded multiple times using different models, the overall embedding and routing time remains imperceptible on the figure. In standalone inference measurements with a batch size of 32, the router imposes an average latency of only 0.4 ms using an NVIDIA A100 GPU and 0.8 ms using an AMD EPYC 7543 32-Core CPU. This highlights that the routing step performed by RAGRoute adds negligible computational overhead and has an insignificant impact on the end-to-end query latency. Overall, these results show that the pre-retrieval routing in RAGRoute significantly alleviates the reranking bottleneck while adding almost no overhead. Our design enables scalable, low-latency federated retrieval even when the number of data sources grows.

4.4 Ablation study

We conduct an ablation study to evaluate the contribution of different input features to the router's performance on the MIRAGE and FeB4RAG benchmarks. Specifically, we train the router to predict relevance using the following combinations of features: *(i)* the query embedding and the centroid, *(ii)* the query embedding and the source-id, and *(iii)* all three features (Section 3.3). The query embedding must always be present as an input feature to be able to predict relevance for that query. Table 2 summarizes the results. The model trained with all three features achieves the highest recall on both the MIRAGE and FeB4RAG benchmarks, indicating that both the centroid and the source-id are important for effective routing. We also explored additional features such as the number of documents per source and the density around the centroid. However, including these features did not lead to further performance improvements.

5 Related work

RAG with multiple data sources. FeB4RAG examines federated search within the RAG paradigm and focuses on optimizing resource selection and result merging to enhance retrieval efficiency [39]. The underlying idea consists of introducing a dataset for federated search and incorporating LLM-based relevance judgments to benchmark resource selection strategies. Notably, the paper

emphasizes the importance of developing novel federated search strategies for RAG. Salve et al. propose a multi-agent RAG system where different agents handle the querying of databases with differing data formats (*e.g.*, relational or NoSQL) [35].

Other approaches focus on privacy in federated search. RAFFLE is a framework that integrates RAG into the federated learning pipeline and leverages public datasets during training while using private data only at inference time [30]. C-FEDRAG is a federated RAG approach that enables queries across multiple data sources and leverages hardware-based trusted execution environments (TEEs) to ensure data confidentiality [2]. FRAG leverages homomorphic encryption to enable parties to collaboratively perform ANN searches on encrypted query vectors and data stored in distributed vector databases, ensuring that no party can access others' data or queries [43]. These schemes can benefit from RAGROUTE while ensuring privacy-preserving federated search.

Machine learning (ML)-assisted resource selection. ML models have been explored to support resource selection in federated search [16]. Arguello *et al.* leverage different features, *e.g.*, the topic of queries, and train a classifier for resource selection [4]. Learn-to-rank approaches such as SVMRANK [11] and the LambdaMART-based LTRRS [41] refine relevance rankings by leveraging diverse feature sets. Ergashev *et al.* construct a heterogeneous graph to capture query-source and source-source relationships and then predict the query-source relevance ranking using a graph neural network (GNN) [13]. Wang *et al.* use an LLM as a resource selector, introducing a novel prompting approach called ReSLLM [40]. They also propose to fine-tune ReSLLM through previously logged queries and snippets from data sources. However, these approaches are either more computationally expensive than the lightweight RAGROUTE router or cannot handle heterogeneous embedding models across data sources.

6 Conclusion

We presented RAGROUTE, a novel and efficient routing mechanism for federated search in RAG systems. By dynamically choosing relevant data sources at query time via a lightweight neural classifier, RAGROUTE minimizes unnecessary queries while preserving high retrieval quality. Evaluations on MIRAGE, MMLU, and FEB4RAG demonstrate that RAGROUTE reduces the number of contacted sources and document retrieval communication volume by up to 80.65%, and decreases end-to-end latency by up to 52.50%, with minimal impact on end-to-end accuracy. These gains are primarily achieved by reducing reranking overhead which constitutes a major bottleneck in RAG pipelines. Our results confirm that querying all data sources is often unnecessary, underscoring the importance of query-aware retrieval strategies in federated search workflows for RAG.

Acknowledgments. This work has been co-funded by the Swiss National Science Foundation under the project *FRIDAY: Frugal, Privacy-Aware and Practical Decentralized Learning*, SNSF proposal No. 10.001.796.

References

1. Baai/bge-reranker-v2-m3, https://huggingface.co/BAAI/bge-reranker-v2-m3
2. Addison, P., Nguyen, M.T.H., et al.: C-fedrag: A confidential federated retrieval-augmented generation system. arXiv preprint arXiv:2412.13163 (2024)
3. Allouah, Y., Dhasade, A., et al.: Revisiting ensembling in one-shot federated learning. In: NeurIPS (2025)
4. Arguello, J., Callan, J., Diaz, F.: Classification-based resource selection. In: CIKM (2009)
5. Bharathi Mohan, G., Prasanna Kumar, R., et al.: An analysis of large language models: their impact and potential applications. Knowl. Inf. Syst. (2024)
6. Bhavnani, S.K., Wilson, C.S.: Information scattering. In: Encyclopedia of library and information sciences (2009)
7. Bian, N., Lin, H., et al.: Influence of external information on large language models mirrors social cognitive patterns. IEEE Trans. Comput. Soc. Syst. (2025)
8. Callan, J.: Distributed information retrieval. In: Advances in information retrieval: recent research from the center for intelligent information retrieval (2002)
9. Clifton, C.: Federated search (2016), https://www.cs.purdue.edu/homes/clifton/cs54701/FederatedSearch_0310.pdf
10. Cuconasu, F., Trappolini, G., et al.: The power of noise: Redefining retrieval for rag systems. In: SIGIR (2024)
11. Dai, Z., Kim, Y., Callan, J.: Learning to rank resources. In: SIGIR (2017)
12. Douze, M., Guzhva, A., et al.: The faiss library. IEEE Trans. Big Data (2025)
13. Ergashev, U., Dragut, E., Meng, W.: Learning to rank resources with gnn. In: WWW (2023)
14. Gao, Y., Xiong, Y., et al.: Retrieval-augmented generation for large language models: A survey. arXiv preprint 2312.10997 (2024)
15. Garba, A., Khalid, S., et al.: Embedding based learning for collection selection in federated search. Data Technologies and Applications (2020)
16. Garba, A., Wu, S., Khalid, S.: Federated search techniques: an overview of the trends and state of the art. Knowl. Inf. Syst. (2023)
17. Grattafiori, A., Dubey, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)
18. Haltaufderheide, J., Ranisch, R.: The ethics of chatgpt in medicine and healthcare: a systematic review on large language models (llms). NPJ digital medicine (2024)
19. Hendrycks, D., Burns, C., et al.: Measuring massive multitask language understanding. arXiv preprint 2009.03300 (2021)
20. Ji, Z., Lee, N., et al.: Survey of hallucination in natural language generation. ACM Comput. Surv. (2023)
21. Ji, Z., Yu, T., et al.: Towards mitigating llm hallucination via self reflection. In: Findings of EMNLP (2023)
22. Jin, Q., Kim, W., et al.: MedCPT: Contrastive pre-trained transformers with large-scale pubmed search logs for zero-shot biomedical information retrieval. Bioinformatics (2023)
23. Kairouz, P., McMahan, H.B., et al.: Advances and open problems in federated learning. Found. Trends Mach. Learn. (2021)
24. Kaplan, J., McCandlish, S., et al.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020)
25. Karpukhin, V., Oguz, B., et al.: Dense passage retrieval for open-domain question answering. In: EMNLP (2020)

26. Kukreja, S., Kumar, T., et al.: Performance evaluation of vector embeddings with retrieval-augmented generation. In: ICCCS (2024)
27. Lewis, P., Perez, E., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: NeurIPS (2020)
28. Li, L., Zhang, Z., Wu, S.: Lda-based resource selection for results diversification in federated search. In: WISA (2018)
29. Li, W., Zhang, Y., et al.: Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. IEEE Trans. Knowl. Data Eng. (2020)
30. Muhamed, A., Thaker, P., et al.: Cache me if you can: The case for retrieval augmentation in federated learning. In: ICLR Workshop on Privacy Regulation and Protection in Machine Learning (2024)
31. Ollama: Ollama: Get up and running with large language models. GitHub repository (2025), https://github.com/ollama/ollama, accessed: February 8, 2025
32. OpenAI Cookbook: Search reranking with cross-encoders (2024), https://cookbook.openai.com/examples/search_reranking_with_cross-encoders
33. Pinecone: Refine with rerank: Better rag results using cross-encoders (2024), https://www.pinecone.io/learn/refine-with-rerank/
34. Şakar, T., Emekci, H.: Maximizing rag efficiency: A comparative analysis of rag methods. Natural Language Processing (2025)
35. Salve, A., Attar, S., et al.: A collaborative multi-agent approach to retrieval-augmented generation across diverse data. arXiv preprint arXiv:2412.05838 (2024)
36. Shokouhi, M., Si, L.: Federated search. Found. Trends Inf. Retr. (2011)
37. Team, M.: Retrieval-qa-benchmark: A benchmark for evaluating retrieval-augmented qa systems (2024), https://github.com/myscale/Retrieval-QA-Benchmark
38. Thakur, N., Reimers, N., et al.: Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In: NeurIPS Datasets and Benchmarks (2021)
39. Wang, S., Khramtsova, E., Zhuang, S., Zuccon, G.: Feb4rag: Evaluating federated search in the context of retrieval augmented generation. In: SIGIR (2024)
40. Wang, S., Zhuang, S., et al.: ReSLLM: Large language models are strong resource selectors for federated search. In: WWW Companion (2025)
41. Wu, T., Liu, X., Dong, S.: LTRRS: A learning to rank based algorithm for resource selection in distributed information retrieval. In: CCIR (2019)
42. Xiong, G., Jin, Q., et al.: Benchmarking retrieval-augmented generation for medicine. In: Findings of ACL (2024)
43. Zhao, D.: Frag: Toward federated vector database management for collaborative and secure retrieval-augmented generation. arXiv preprint arXiv:2410.13272 (2024)
44. Zhou, Y., Lei, T., et al.: Mixture-of-experts with expert choice routing. In: NeurIPS (2022)