NoteRecall: Efficient and Privacy-Preserving On-Device Notes Retrieval and Question Answering System

Nivedhitha Dhanasekaran ndhanase
Kshitish Ghate kghate
Jivitesh Jain jivitesj
Vishwa Shah vishwavs

Abstract

We present **NoteRecall**, a novel and efficient on-device retrieval-augmented generation (RAG) system for privacy-preserving question answering. Our system enables secure retrieval of user-provided documents, such as medical records, and generates accurate, evidence-based responses entirely offline. Built using state-of-the-art embedding models and instruction-tuned language models, NoteRecall leverages quantization, efficient indexing, and optimization techniques to ensure computational feasibility on consumer devices like Apple M2-powered MacBooks. We benchmark performance on the BioASQ dataset, demonstrating competitive accuracy while maintaining low latency and resource usage. Our results highlight the feasibility of deploying robust AI applications locally, safeguarding data privacy while enabling high-quality user experiences[1].

1 Introduction

The increasing reliance on cloud-based solutions for managing sensitive personal documents, such as personal notes and medical records, has raised significant concerns regarding privacy and data security. Popular productivity tools powered by cloud-based computation resources (e.g., Notion and Microsoft Loop's "AI Helpers" [2]) often process user data through third-party large language model (LLM) providers. These systems subject data to varying terms of service, geographic and jurisdictional boundaries, and potential vulnerabilities to breaches or unauthorized access. These risks are particularly critical in sensitive domains like healthcare, where maintaining patient confidentiality is paramount.

At the same time, there is growing interest in on-device note-taking and document management tools, such as simple text editors or open-source solutions like Obsidian.[3] While these tools prioritize privacy by avoiding cloud-based data processing, they often lack the AI-driven context-aware capabilities that modern users demand. This gap between privacy and utility underscores the need for innovative solutions.

2 Motivation

The development of an on-device retrieval-augmented generation (RAG) system like NoteRecall matters because it addresses a critical intersection of privacy, usability, and computational efficiency in modern AI applications. Ensuring that sensitive personal data remains private while providing intelligent and context-aware features is a key challenge that existing cloud-based systems fail to solve.

Our specific focus on enabling AI systems to run entirely on-device is crucial for several reasons:

1. **Privacy and Security:** Sensitive personal data, such as medical records and private notes, must remain secure. By eliminating the need for data to leave the user's device, NoteRecall ensures robust data privacy and eliminates the risks associated with external data processing and cloud-based breaches.

2. **Autonomy and Reliability:** On-device systems operate without reliance on an internet connection, making them suitable for offline or resource-constrained scenarios. This ensures users can access AI-driven insights regardless of connectivity.

3. **Unique Technical Challenges:** On-device systems must meet stringent requirements for computational efficiency and energy use while maintaining competitive performance. This involves balancing resource constraints with

[1] https://github.com/nive927/note-recall/
[2] https://www.notion.so/product/ai
[3] https://obsidian.md/

the need for accurate retrieval and high-quality answer generation, a trade-off often neglected in cloud-centric systems.

Our work addresses these challenges by leveraging dense retrieval, quantization, and model distillation to create an energy-efficient and computationally feasible RAG pipeline for consumer devices. Focusing on privacy-preserving architectures, NoteRecall enables applications in highly sensitive domains, such as healthcare, where existing solutions are either inadequate or unsuitable. This combination of privacy, performance, and efficiency makes NoteRecall a unique contribution to the growing field of on-device AI.

3 Related Work

3.1 Sparse & Dense Retrieval Systems

Sparse retrieval methods like BM25, although widely used for their simplicity and effectiveness, underperformed in our context as they lacked semantic depth. Instead, we adopted Dense Passage Retrieval (DPR) (Karpukhin et al., 2020), which generates semantically rich embeddings for query-document matching. Dense retrieval's superior performance, particularly in technical domains such as medicine, informed our selection of embedding models benchmarked using MTEB (Muennighoff et al., 2022). This ensured optimal retrieval performance on our task-specific BioASQ dataset, which comprises medical question-answer pairs.

3.2 Retrieval Augmented Generation (RAG) & Question Answering (QA) Systems

RAG combines retrieval mechanisms with generative models for robust QA systems. Our architecture aligns with previous work in leveraging instruction-tuned models like LLaMA (Meta, 2024) for reader tasks. This approach strikes a balance between reasoning over retrieved contexts and maintaining computational efficiency. While prior RAG frameworks often focus on cloud-based setups, our on-device adaptation with state-of-the-art embedding and reader models demonstrated comparable performance, as measured by retrieval recall and BERTScore.

3.3 Vector Search and FAISS

Efficient vector search was a critical component, handled using Facebook AI Similarity Search (FAISS) (Douze et al., 2024a). FAISS's Hierarchical Navigable Small World (HNSW) index offered a balance between memory efficiency and retrieval accuracy, suitable for our relatively small knowledge corpus. Unlike earlier studies focused on scaling FAISS to massive corpora, our work concentrated on achieving interactive latency in a constrained on-device environment.

3.4 Quantization Techniques for Efficient Embedding and Retrieval

Quantization was instrumental in achieving our efficiency goals. Guided by methodologies in Hugging Face's quantization practices (Shakir et al., 2024), we employed naive and k-means quantization for weight compression. The Q5_K_M and Q3_K_S configurations from llama.cpp (Levrard, 2018) provided an effective trade-off between latency and model performance.

3.5 Model Output Distillation

Our work aligns with the capabilities of efficient models such as the Phi series (Gunasekar et al., 2023; Li et al., 2023a), which leverage similar strategies for preserving task performance while reducing computational requirements. Model output distillation proved effective for adapting large, state-of-the-art models to on-device applications without sacrificing quality.

3.6 Application

Incorporating private contexts is a growing area of focus in QA systems, as demonstrated by benchmarks like ConcurrentQA, which integrate private data sources (e.g., emails) alongside public knowledge (e.g., Wikipedia) (Arora et al., 2023). These systems highlight the importance of managing privacy-quality tradeoffs, often requiring selective predictions based on confidence scores to optimize performance over sensitive data. Unlike ConcurrentQA, which uses Split Iterative Retrieval (SPIRAL) for hybrid public-private retrieval, NoteRecall prioritizes strict data isolation by ensuring all computations occur on-device, addressing privacy concerns more directly. While ConcurrentQA sacrifices up to 19% performance due to reliance on public retrieval models, NoteRecall's dense retrievers, fine-tuned on medical QA tasks like BioASQ, minimize this gap. By focusing on a fully private retrieval pipeline, our system extends these approaches to environments where private data must remain entirely local, ensuring data privacy while

maintaining competitive retrieval and QA performance.

3.7 Energy Efficiency

Energy considerations were central to our design choices. Using tools like CodeCarbon [4], we quantified the energy and carbon footprint of different configurations, highlighting the environmental benefits of on-device ML systems. For example, we observed a significant reduction in energy consumption when using quantized models compared to their full-precision counterparts. By contextualizing these measurements with real-world activities (e.g., smartphone charging, video streaming) (iea; Kilgore), we emphasized the broader environmental implications of our approach.

4 Task Definition

This section defines the NoteRecall system, including its inputs, outputs, evaluation metrics, and datasets used.

Data Modalities. NoteRecall consumes and produces textual data in the form of documents used for context, user queries, and generated answers.

- **Inputs:** User-provided textual documents (e.g., medical records) and natural language queries. We note that documents can be embedded and indexed by NoteRecall in a batch – appropriate for pre-existing documents – as well as individually on-the-fly,useful as new documents come in.
- **Outputs:** A ranked list of relevant documents retrieved from the input corpus and a concise, generated answer to the query based on the retrieved documents.

Target Hardware Platform. As most people use personal laptops for creating, reading, and storing documents, we optimize NoteRecall to run on personal laptops and computers. Specifically, we consider the Apple MacBook Pro with an Apple Silicon M2 processor as our target hardware. The processor contains 10 CPU cores and 16 GPU cores.

Evaluation Metrics. We report the following metrics for quantifying NoteRecall's performance:

- **Performance Metrics:** We use BERTScore F1 between the generated and ground truth answers as a measure of their semantic similarity. While we experimented with other metrics such as ROUGE and exact match scores, we found BERTScore to be the most correlated with our qualitative assessment of output quality without overly penalizing phrasing variations.
- **Efficiency Metrics:** We report the end-to-end inference latency (in seconds) of the entire pipeline as our efficiency metric. As ours is an interactive system, this end-to-end latency at inference time is what the user notices and cares about. We also report the number of inference cycles we can execute on a 10 Wh power supply as a measure of NoteRecall's power consumption.

Dataset. As private medical documents are our most prominent use case, we train and test NoteRecall on the **BioASQ Task B** dataset,[5] a question-answering dataset on Bio-Medical documents. The dataset contains the following:

- **Documents:** The dataset contains 3,680 medical documents. We use all documents to construct our knowledge corpus.
- **Question-Answer Pairs:** The dataset contains 300 question-answer pairs with gold-standard corresponding document label. We use these for evaluating our system.

The BioASQ dataset is particularly suited for evaluating the NoteRecall system as it emphasizes domain-specific, high-quality retrieval and QA tasks, aligning closely with the project's objectives.

5 Methods

The NoteRecall system employs a retrieval-augmented generation (RAG) pipeline for privacy-preserving, on-device question answering. The pipeline integrates three key components: a vector database for efficient similarity-based retrieval, an embedding model for converting text into dense representations, and a reader model for generating answers from retrieved documents. At the indexing stage, the knowledge corpus is divided into chunks of 512 tokens, which are embedded using the selected embedding model and stored in the vector

[4] https://github.com/mlco2/codecarbon

[5] https://huggingface.co/datasets/BastienHot/BioASQ-Task-B-Revised

database. During inference, the user query is embedded similarly and matched against the stored document embeddings to retrieve the top $k = 3$ most relevant documents. These retrieved documents are provided as context to the reader model, which synthesizes a natural language answer based on the query and context. We leverage hardware-specific optimizations such as Apple's Metal framework [6] to optimize the system for on-device deployment. This includes leveraging PyTorch's Metal Performance Shading (MPS), llama.cpp and Apple's ML-eXplore (MLX) for GPU acceleration. The comparative performance for some of these implementations is analyzed in Section 6.6.

We use the GTE Qwen-2 1.5B Instruct model (Li et al., 2023b) for embedding generation ($d_{\text{embed}} = 1536$) due to its competitive performance on the MTEB benchmark (Muennighoff et al., 2022), Meta's Llama 3.2 1B Instruct model (et al., 2024) as the reader, and FAISS (FaceBook AI Similarity Search) (Douze et al., 2024b) as the vector database. Owing to the relatively small size of our knowledge corpus, we are able to use a high-fidelity nearest neighbor search algorithm (Hierarchical Navigable Small World) (Malkov and Yashunin, 2016) despite its relatively large memory consumption.

As embedding generation is not autoregressive, it can be parallelized effectively and takes significantly lesser time than text generation. As a result, the reader model tends to be efficiency bottleneck of our pipeline and the focus of our optimization efforts. We explore this tradeoff in Section 6.3. Further, any performance degradation in retrieval takes a significant toll on our task performance, as relevant context is essential for accurate generation. Consequently, unless specified otherwise, the following model compression techniques are applied to the reader model.

5.1 Pruning

We perform the following structured and unstructured pruning techniques on the reader model and report results in Section 6.1.

Random Layer Pruning. This structured pruning technique removes entire non-embedding, non-causal-LM layers of the model at random. While it achieves significant efficiency improvements in terms of reduced computation and memory, it also adversely affects performance as the removed layers disrupt both their own functions and the downstream input distributions.

L_1 Magnitude Pruning of Feed-Forward Layers. This unstructured pruning technique is applied to feed-forward layers, the most parameter-dense layers. Weights with low L_1 norms along specific dimensions are pruned under the hypothesis that their low magnitude indicates their low importance in maintaining model functionality.

L_1 Magnitude Pruning of Attention Projection Matrices. Rows and columns of the attention matrices (Q, K, V, and O) are pruned based on their L_1 norms, effectively nullifying low-activation regions in the query-key and value spaces. While this is a structured pruning technique, it requires the use of specialized sparsity-aware libraries to take advantage of the nullified rows and columns.

Random Pruning of Entire Attention Heads. This structured pruning technique randomly removes a subset of attention heads. We implement this in two ways, (1) by only nullifying the corresponding matrices, and (2) by physically removing those matrices. While these two techniques lead to identical performance, the latter leads to significant efficiency gains due to the physical removal of matrices corresponding to certain heads.

Why Pruning? Although pruning is a commonly used model compression technique, it is of limited utility for our use case due to the significant drop in model performance (see Section 6.1). Further, many pruning techniques require the use of sparsity-aware matrix operations to realize efficiency gains, which are not available for our hardware.

5.2 Quantization

We experiment with several post-training quantization techniques on the reader model. Specifically, we focus on k-means quantization, which groups and quantizes blocks of parameters with similar magnitude separately, insulating each block from outlier effects of other blocks. We report our results in Section 6.2.

Llama.cpp[7] and Hugging Face Optimum Quanto[8] are two popular quantization frameworks. We find no efficiency gains from static or dynamic

[6] https://developer.apple.com/metal/

[7] https://github.com/ggerganov/llama.cpp

[8] https://github.com/huggingface/optimum-quanto

quantization with the latter, owing to its lack of comprehensive support for our target hardware. As a result, we use Llama.cpp for our experiments, owing to its native support for Apple Silicon via Metal, support for k-means quantization, and optimized serialization format (GGUF).

Why Quantization? Quantization is essential for reducing the computational and memory demands of large models, making them more suitable for on-device deployment. By representing model weights with lower precision, such as 8-bit or even fewer bits, quantization significantly reduces model size and inference latency while maintaining comparable performance. This is particularly important for our task, as the NoteRecall system must operate efficiently on devices with constrained resources. Additionally, quantization enables a seamless trade-off between efficiency and accuracy, allowing us to adapt to the diverse hardware capabilities of user devices. It is a practical and scalable approach to achieve the high performance required for privacy-preserving, on-device retrieval-augmented generation systems.

5.3 Mixture-of-Experts Architecture

We also compare standard decoder models to the recent mixture-of-expert style models (MoE) (Jiang et al., 2024; Team, 2024). The key to MoEs' efficiency lies in their sparse activation mechanisms. Unlike dense models that activate all parameters for every input, MoEs selectively activate only a subset of experts for each task. For this ablation, we compare the performance of Qwen-1.5-MoE-A2.7B (MoE) and Qwen1.5-7B (standard). In theory, this MoE model should achieve the performance of a 7B model with the inference latency of a 2B model. In practice, we do not see these performance gains (see Section 6.4).

Why MoE? By activating only the most relevant experts for each input, MoE models can handle massive amounts of data with far fewer computational resources. This is particularly important for larger models like in our case, where processing time and memory requirements can quickly become prohibitive.

5.4 Model Distillation

Model distillation is a technique used to transfer knowledge from a larger, more capable teacher model to a smaller, more efficient student model. We apply this technique to the reader, distilling Llama's knowledge into the much smaller Flan-T5-Base model (Chung et al., 2022). We generated 3,327 additional questions from unutilized passages in the corpus with Llama 3.1 7B Instruct, and then generated answers for each using our reader model with the ground truth context to use as fine-tuning data for the student model. See Section 6.5 for our results.

Why Distillation? Model distillation is a practical and scalable solution for achieving high performance in resource-constrained environments. While using synthetic data instead of gold-standard ground truth seems counter-intuitive, it allows the student model to focus on signals already extracted by the larger teacher model, reducing the effect of noise present in the gold-standard training data. It allows for the teacher model's capabilities to be retained to a large extent with the significantly smaller computational footprint of the student. This makes it an ideal technique for our pipeline, where both computational efficiency and task relevance are critical.

6 Results and Discussion

For all our results below, the latency is the total latency of retrieval using query and answer generation with reader for a single instance. We compute this as the average latency across instances. We present a combined plot of a subset of our experiments for performance vs latency in Fig 4 and performance vs budget in 5. Our baseline is denoted by 1 which involves both reader and retriever in full precision.

6.1 Pruning

We present our results of pruning the reader in Table 1. We apply different structured pruning methods as described earlier along with unstructured pruning. Since we use torch for implementing these pruning methods, the results are based on the torch's mps accelerator for Mac. In Table1, We see a clear drop in the disk space as the global sparsity rate increases, however, the change in latency is quite low and further leads to significant drop in performance. We also experiment with unstructured pruning without and with quantization config Q5_K_M with llama.cpp and present the results in points 10, 11 in Fig 4.

Table 1: Results of various pruning techniques. Unless specified otherwise, the local sparsity rate of the module being pruned is 0.33.

Pruning Technique	Global Sparsity Rate	BertScore F1	Latency (s)	Disk Space (MB)
Base Model	0.000	0.5715	8.5	2358
Random Layers	0.330	0.2631	5.58	1778
L1 MLP Layers	0.177	0.246	7.92	2370
L1 Attn. Matrices	0.037	0.2916	8.24	2358
Random Attn. Q Heads	0.028	0.3638	7.63	2358
Random Attn. KV Heads	0.034	0.4015	8.38	2370
Random Attn. KV Heads (Physical)	0.034	0.4015	6.96	2278
Pruning + Quantization	0.330	0.27150	4.61	1260

Model LLaMA3.2-1B-inst	Latency (s)	BERTScore
F16	1.99	0.5693
+ Q5_K_M	1.9	0.5582
+ Q3_K_S	1.45	0.4783

Table 2: Quantizing the reader model

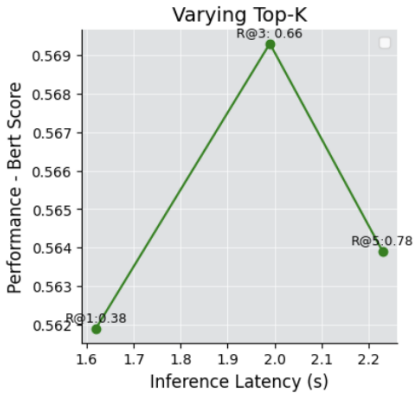

Figure 1: Varying Top-K passages to be retrieved

6.2 Quantization

GGML via llama.cpp provides several quantization settings. [9] We experiment with different range of compressions: Q3_K_S, Q_5_K_M, Q8_0 for the reader model, where the number denote the bit precision, K denotes k-means quantization and M/S denote the size depending on number of Feed forward and Attention layers/weight blocks quantized. You can find more implementation details about the specific quantized layers in the PR[9]. llama.cpp utilizes GPU and provides considerable speed-up with 3-bit and 5-bit quantization without significant drop in performance unlike pruning as seen in Table 2.

6.3 Retriever vs Reader Trade-off

Top-K We analyse the number passages to be retrieved as input sequence length is proportional to the latency. As shown in points 1 (Top-3), 2 (Top-1), 3 (Top-5) in Fig4 and 1, Top-3 works best considering trade-off between latency and context performance. However, retrieval recall drop is recoverable owing to reader's inherent memory which leads to lesser drop in final BERTScore, making 2: Top-1 an efficient system.

Quantizing Retriever vs Reader Additionally, since we use similar capacity retriever and reader, we compare quantization for the retriever and reader to study its tradeoff for efficiency. We apply Quantization settings Q5_K_M, Q3_K_S to both retriever and reader with llama.cpp in 4, 5, 6, 7 respectively in Fig 4. Efficiency gains are predominantly driven by latency of the reader as retrieval encoding is largely parallelized while reader decoding is proportional to generated sequence length as can be seen in Fig 2. We note that 7: Q3_K_S Reader quantization provides decent performance with improvement while also reducing latency.

6.4 Mixture-of-Experts Architecture

To compare architecture variance in 8 and 9, we analyze difference between Qwen-1.5-MoE-A2.7B and Qwen1.5-7B (Team, 2024), where the MoE has

[9] https://github.com/ggerganov/llama.cpp/pull/1684

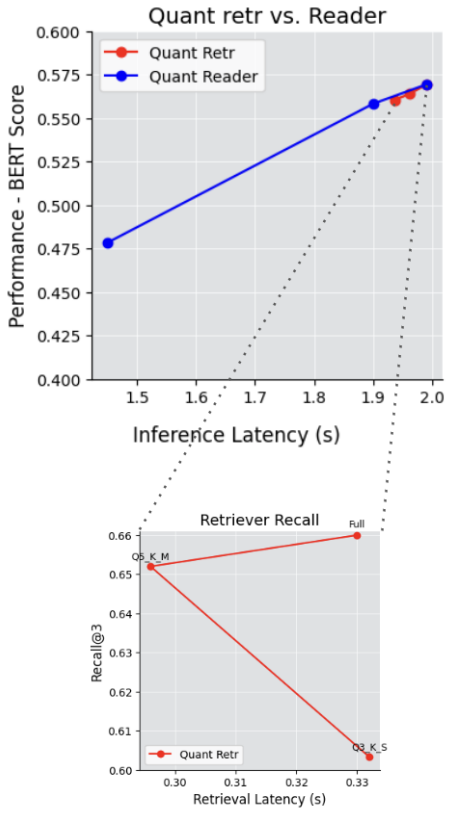

Figure 2: Quantizing Retriever vs Reader

Model	Latency(s)	BERT Score
Qwen1.5-7B	3.1	0.4162
Qwen-1.5-MoE-A2.7B	8.3	0.5683

Table 3: Comparing different model architectures

Model	Latency (s)	BERTScore
Flan-T5-Base	1.02	0.4732
+ Distilled FT	1.03	0.5597

Table 4: Model Distillation with output fine-tuning

2.7B activated parameters. Performance with MoE is much better as it is higher capacity However latency is higher as M2's unified memory cannot fully load 14B model, leading to loading + unloading bottlenecks with MoE routing as shown in Table 3.

6.5 Model Distillation

Flan-T5's zero-shot performance is not comparable with LLaMA but is almost 2x more efficienct with respect to latency. To leverage this speedup, we use model output distillation, This style of fine-tuning helps substantially improve Flan-T5's performance, making it comparable to LlaMA as seen in base and fine-tuned ablations 12, 13 in Fig 4 and Table 4. Since Flan-T5 is not supported fully by llama.cpp, we use MLX for inference. (MLX and llama.cpp give similar runtime for other models)

6.6 Comparing Frameworks

Lastly, we also evaluate difference in frameworks - we observe MLX provides better performance than MPS, owing to MLX being optimized for M2 chips and with the ability to completely utilize unified memory as seen in Fig 3.

Conclusion Lastly we also provide a performance under budget curve to compare BERTScore vs Number of inference steps we can run under 10Wh in Fig 5. From both Fig 5 and Fig 4, we see that our fine-tuned FlanT5 model (point 13) can perform the highest number of inferences with competent performance, almost 2.4x speedup with respect to the baseline performance. Quantizing LLaMA Q3_K_S (point 7) provides the next best speedup with some drop in performance and a beneficial method for training or finetuning free optimization.

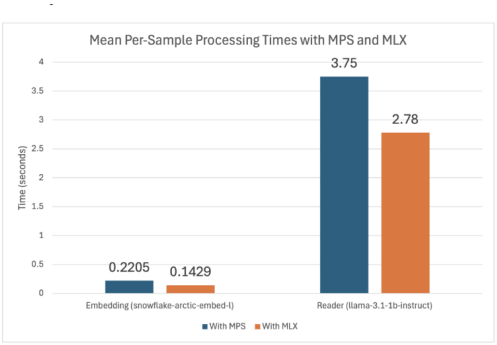

Figure 3: MPS vs MLX performance

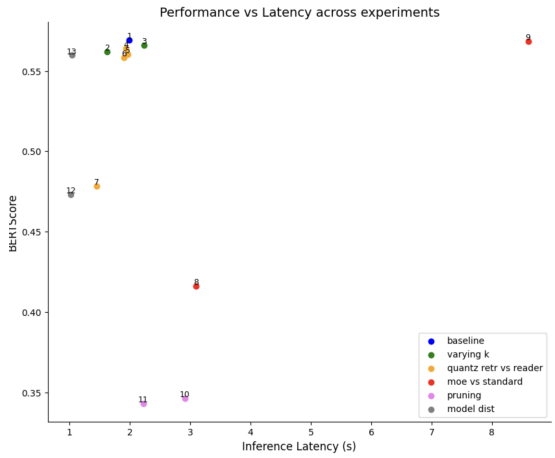

Figure 4: Performance vs Inference Latency across experiments

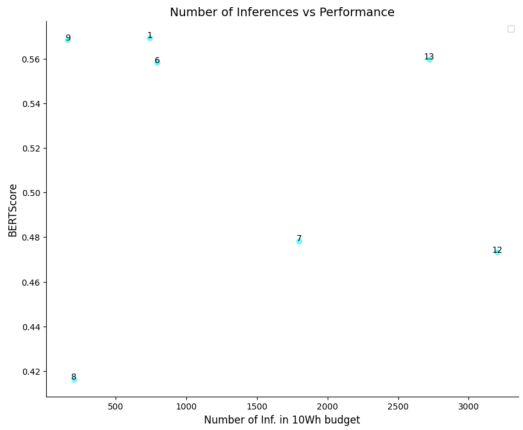

Figure 5: Performance under 10Wh Budget across experiments

7 Key Challenges

1. **Latency Bottleneck in the Reader:** The interactive nature of NoteRecall necessitates serial execution of the retriever and reader models. While the retriever's latency can be minimized through parallelization, the reader remains a significant bottleneck due to its proportional dependence on the sequence length during decoding.

2. **Performance Dependency on Retriever:** The generation quality is tightly coupled to the retriever's ability to provide relevant context. This limitation is particularly evident in technical domains like medicine, where highly accurate retrieval is necessary for meaningful answers.

3. **Nascent On-Device ML Libraries for Apple Silicon:** PyTorch's Metal Performance Shaders (MPS), while available, are not as optimized as Apple's proprietary MLX framework. Experimenting with MLX required substantial reimplementation, slowing development.

4. **Resource Constraints on GPU:** Apple Silicon's unified memory architecture creates challenges in simultaneously loading both the retriever and reader models. This limitation either caps the model size or necessitates frequent on- and off-loading, increasing latency.

8 Insights and Future Work

1. **Model Distillation:** Reducing the reader model's size through model distillation is a promising avenue. Training a smaller model using the outputs of the current reader can maintain performance while reducing resource requirements.

2. **Task-Specific Fine-Tuning:** Fine-tuning models on specific domains, such as medical documents, could enhance performance. For broader applicability, a mixture of expert models fine-tuned on distinct domains may be an effective strategy.

3. **Optimizing Retrieval Index:** While HNSW provided high fidelity for the small corpus, it is memory-intensive. Exploring alternative indexing methods and distance metrics may

allow scalability to larger corpora while preserving efficiency.

4. **Enhanced On-Device ML Libraries:** Advocating for broader support and optimization of on-device libraries like MLX can significantly boost efficiency and enable seamless integration of larger models without reimplementation overhead.

References

The carbon footprint of streaming video: fact-checking the headlines – Analysis - IEA — iea.org. [Accessed 14-10-2024].

Simran Arora, Patrick Lewis, Angela Fan, Jacob Kahn, and Christopher Ré. 2023. Reasoning over public and private data in retrieval-based systems. *Transactions of the Association for Computational Linguistics*, 11:902–921.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024a. The faiss library.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024b. The faiss library.

Dubey et al. 2024. The llama 3 herd of models.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Georgette Kilgore. Carbon Footprint of a Laptop vs MacBook vs Desktop Computer vs iPhone — 8billiontrees.com. https://8billiontrees.com/carbon-offsets-credits/carbon-footprint-of-a-laptop/#:~:text=A%20desktop%20computer%20emits%20778,electricity%20consumption%20when%20in%20use. [Accessed 14-10-2024].

Clément Levrard. 2018. Quantization/clustering: when and why does k-means work?

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023a. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023b. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320.

Meta. 2024. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Aamir Shakir, Tom Aarsen, and Sean Lee. 2024. Binary and scalar embedding quantization for significantly faster cheaper retrieval. *Hugging Face Blog*. Https://huggingface.co/blog/embedding-quantization.

Qwen Team. 2024. Qwen1.5-moe: Matching 7b model performance with 1/3 activated parameters".