Building an Efficient Multilingual Non-Profit IR System for the Islamic Domain Leveraging Multiprocessing Design in Rust

Vera Pavlova
rttl labs, UAE
v@rttl.ai

Mohammed Makhlouf
rttl labs, UAE
mm@rttl.ai

Abstract

The widespread use of large language models (LLMs) has dramatically improved many applications of Natural Language Processing (NLP), including Information Retrieval (IR). However, domains that are not driven by commercial interest often lag behind in benefiting from AI-powered solutions. One such area is religious and heritage corpora. Alongside similar domains, Islamic literature holds significant cultural value and is regularly utilized by scholars and the general public. Navigating this extensive amount of text is challenging, and there is currently no unified resource that allows for easy searching of this data using advanced AI tools. This work focuses on the development of a multilingual non-profit IR system for the Islamic domain. This process brings a few major challenges, such as preparing multilingual domain-specific corpora when data is limited in certain languages, deploying a model on resource-constrained devices, and enabling fast search on a limited budget. By employing methods like continued pre-training for domain adaptation and language reduction to decrease model size, a lightweight multilingual retrieval model was prepared, demonstrating superior performance compared to larger models pre-trained on general domain data. Furthermore, evaluating the proposed architecture that utilizes Rust Language capabilities shows the possibility of implementing efficient semantic search in a low-resource setting.

1 Introduction

Dense retrieval is an advanced approach in IR that utilizes embeddings to identify semantically similar text, known as semantic search. LLMs are a key component in creating text embeddings and performing dense retrieval (Karpukhin et al., 2020; Izacard et al., 2021). One of the first challenges in building a non-profit multilingual domain-specific IR system is that the use of publicly available multilingual large language models (MLLMs) pre-trained on a general domain could deteriorate performance due to domain shift when applied to new domains (Lee et al., 2019; Huang et al., 2019). To overcome this, we begin with pre-training an MLLM for the Islamic domain to address this issue. However, pre-training a domain-specific MLLM brings two additional challenges. Firstly, assembling a multilingual domain-specific corpus for pre-training a MLLM requires a large amount of domain-specific data that is often difficult to find in different languages. Secondly, multilingual models are heavyweight, frequently exceeding 1GB, making them challenging to deploy. To effectively tackle the issue of pre-training domain-specific MLLM, we employ a continued pre-training approach and incorporate domain-specific vocabulary to accommodate the domain shift better (Beltagy et al., 2019). To deal with the challenge of the large size of MLLM, we perform language reduction and remove languages not needed in the current deployment. This method helps us reduce the model's size by more than half, even after introducing new domain-specific vocabulary. We use this lightweight domain-specific MLLM as a backbone for the retrieval. Evaluation of this model on an in-domain IR dataset found that our model significantly outperforms general-domain multilingual and monolingual models even after performing language reduction.

Moreover, deploying non-profit AI systems implies operating on a limited budget, which makes it challenging to use embedding APIs or libraries that rely on GPU acceleration to perform search reasonably fast. To tackle this challenge and meet the requirements of implementing an ad hoc IR system on a public website, we utilize the multiprocessing capabilities of Rust Language to create an efficient and secure semantic search based on CPU architecture (Abdi et al., 2024; Seidel and Beier, 2024; Liang et al., 2024). Our system's evaluation and comparison against others, such as Faiss, indicates

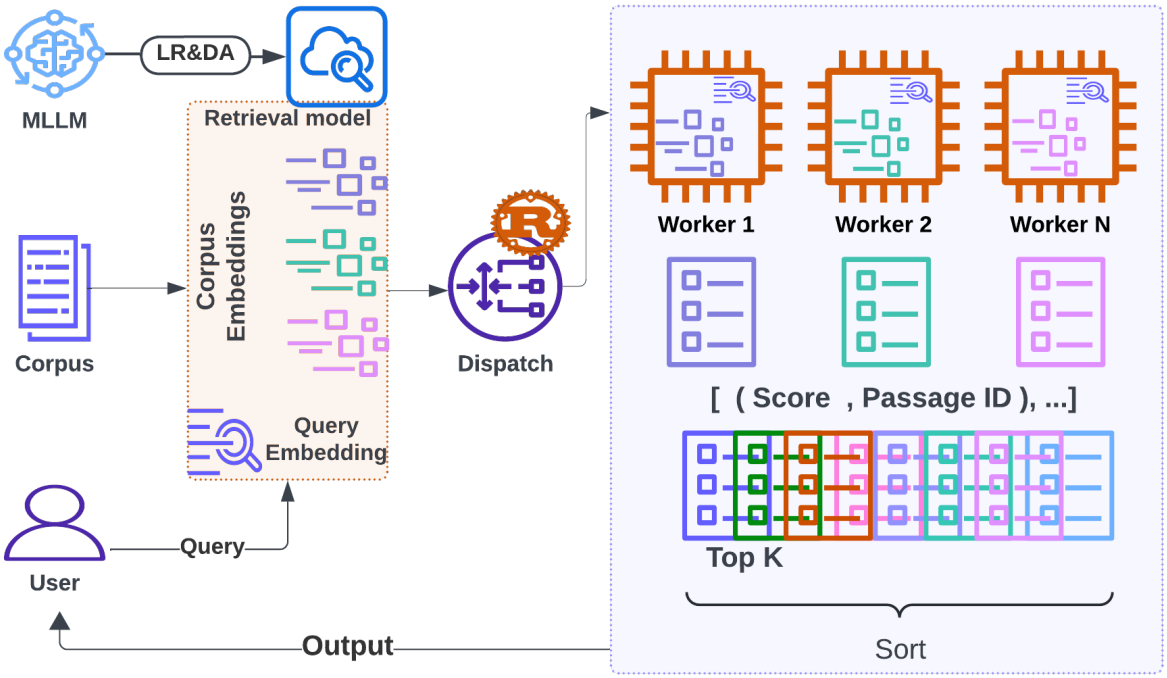

Figure 1: The main components of building a multilingual IR system. In the upper left corner is the preparation of the retrieval model that includes language reduction (LR) and domain adaptation (DA). The rest of the figure shows the implementation of semantics search in Rust with multiprocessing architecture.

that our implementation of semantic search with underlying Rust multiprocessing architecture can significantly accelerate search without compromising performance.

Our main contributions are:

- We have developed a free online multilingual search tool for exploring well-established literature in the Islamic domain.[1]
- To the best of our knowledge, we are the first to deploy open-source, non-profit semantic search leveraging multiprocessing using Rust language.

2 Lightweight Domain-Specific MLLM

2.1 Size Reduction of MLLM

MLLMs allow access to functionality in several languages using one model and enabling cross-lingual transfer. Pre-training mBERT (Devlin et al., 2019) and XLM (Lample and Conneau, 2019) on Wikipedia brought a new state-of-the-art to multilingual tasks. Conneau et al. (2020) showed that increasing MLLM's capacity and training on a larger corpus like CommonCrawl resulted in better-performing models such as XLM-R and XLM-R_{Base}. However, improved performance comes at the cost of the model's larger size (714MB for mBERT vs. 1.1GB for XLM-R_{Base}). The size of the model makes it heavy to deploy in low-resource settings. Sun et al. (2019); Tang et al. (2019); Sanh et al. (2019); Li et al. (2020) showed that distillation of transformer-based language models (Vaswani et al., 2017) leads to considerable size reduction and adequate performance. Another approach that reduces model size and retains high performance is language reduction of MLLM (Abdaoui et al., 2020). Around 50% of the parameters in mBERT and 70% in XLM-R_{Base} are assigned to the embedding matrix (see Table 2 in Appendix A). Thus, applying language reduction is more favorable in the case of deploying MLLM as it decreases the model size while preserving encoder weights, trimming only the embedding matrix by removing the languages that are not needed in deployment. Unlike Abdaoui et al. (2020), our reduction method involves training a new tokenizer (see Figure 2):

1. We compile the corpus using a multilingual variant of the C4 corpus for the languages of interest (English, Russian, Arabic, and Urdu).
2. Train the SentencePiece BPE tokenizer using

[1] A system is deployed at https://rttl.ai/

this corpus.

3. Find the intersection between the newly trained tokenizer and the original XLM-R_{Base} tokenizer available from Hugging Face,[2] the tokens inside of intersection and corresponding weights will be selected for the new embedding matrix of the XLM-R4 model (34k tokens).

4. We modify the SentencePiece model according to the new tokenizer.

5. At the final stage, we copy the encoder weights from XLM-R_{Base} to the new XLM-R4 model.

The main difference in parameter size between the mBERT and XLM-R_{Base} model is in the size of the embedding matrix (mBERT has 119K tokens, while the XLM-R_{Base} has 252K tokens), while the size of encoder parameters of mBERT and XLM-R_{Base} are the same. By only reducing the size of the embedding matrix of the XLM-R_{Base}, we can significantly decrease the model's size to the size of the bert model or even smaller while benefiting from the extensive training that the XLM-R_{Base} model underwent. The resulting XLM-R4 model, with a size of 481 MB and 119M parameters, is significantly smaller than XLM-R_{Base}, demonstrating the practical implications of our method and its potential for real-world applications (see Table 2).

Table 1 compares how the models perform on the XNLI dataset (Conneau et al., 2018) in the cross-lingual transfer (fine-tuning multilingual model on English training set). As a baseline model, we use an XLM-R_{Base}. Hugging Face implementation of the tokenizer of XLM-R_{Base} is different from the original implementation (Conneau et al., 2020). For a fair comparison, we fine-tune the XLM-R_{Base} and the XLM-R4 model with the same hyperparameters on the English training set of the XNLI dataset (see Appendix A). We also include in comparison mBERT, DistilmBERT (Sanh et al., 2019), and a reduced version of mBERT that consists of 15 languages (Abdaoui et al., 2020). We compare the four languages left after performing the language reduction technique (English, Russian, Arabic, Urdu). Table 1 shows that the best-performing model for all languages is the XLM-R_{Base} (in bold), and the second best-performing model (underlined) is the XLM-R4. We can observe a slight drop in performance of the XLM-R4 in comparison to the XLM-R_{Base}, which is the smallest for Urdu (0.5%) and English and Arabic (1.16% and 1.65% correspondingly), with a more noticeable drop in Russian (3.76%). However, XLM-R4 performs better than the rest of the models, including mBERT. DistilmBERT shows the lowest results in all languages.

[2] https://huggingface.co/FacebookAI/xlm-roberta-base

Model	en	ru	ar	ur
XLM-R_{Base}	**84.19**	**75.59**	**71.66**	**65.27**
XLM-R4	83.21	72.75	70.48	64.95
mBERT	82.1	68.4	64.5	57
mBERT 15lang	82.2	68.7	64.9	57.1
DistillmBERT	78.5	63.9	58.6	53.3

Table 1: Results on cross-lingual transfer for four languages of the XNLI dataset. XLM-R_{Base} and XLM-R4 results are averaged over five different seeds.

Model	Size	#params	EM
mBERT	714 MB	178 M	92 M
XLM-R_{Base}	1.1 GB	278 M	192 M
XLM-R4	481 MB	119 M	33M

Table 2: Comparison of models' size

2.2 Domain Adaptation of MLLM

The XLM-R_{Base} model on which we perform language reduction to get the XLM-R4 model is pretrained on the general domain. We perform domain adaptation of XLM-R4 to account for the domain shift (Lee et al., 2019; Huang et al., 2019). One of the challenges here is the preparation of a multilingual Islamic corpus to adapt the XLM-R4 to the Islamic domain. The situation regarding constructing a multilingual corpus in the Islamic Domain is unusual. In most multilingual corpora, the data is predominantly in English, but in the Islamic domain, it is predominantly in Arabic. The Open Islamicate Texts Initiative (OpenITI) (Romanov and Seydi, 2019) has provided a sizable corpus (1 billion words) for pre-training LLMs in Classical Arabic, which is the language of Arabic Islamic literature. For English, Russian, and Urdu (50 million words altogether), the available text mainly consists of Tafseer (Qur'an exegesis) and Hadith. To avoid having a corpus heavily skewed towards Arabic, we selected a random subset of the OpenITI corpus containing approximately 250 million words. We combine it with content from other lan-

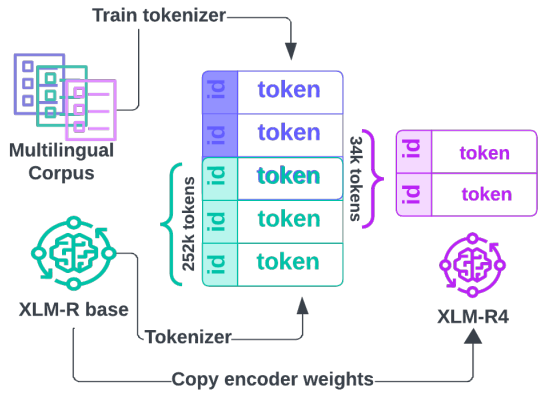

Figure 2: Language Reduction technique that gives us the multilingual XLM-R4 model for four languages (English, Russian, Arabic, and Urdu).

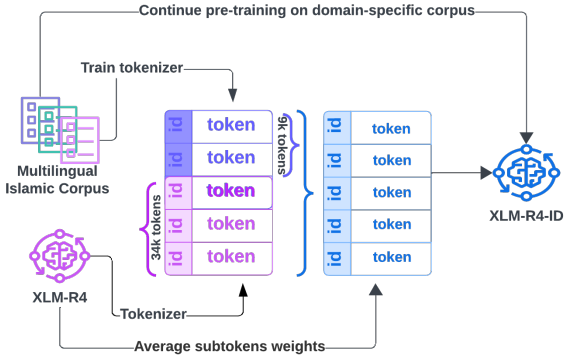

Figure 3: Domain Adaptation of XLM-R4 utilizing continued pre-training approach on Multilingual Islamic Corpus. The final domain-specific model is XLM-R4-ID.

guages, resulting in a corpus of size 300M words for domain adaptation. The corpus size is relatively small; nevertheless, since the weights of the XLM-R4 model are not initialized from scratch, we can apply continued pre-training. To address domain shift more effectively, we introduce new domain-specific vocabulary (Gu et al., 2020; Beltagy et al., 2019; Poerner et al., 2020; Pavlova and Makhlouf, 2023). The domain adaptation of XLM-R4 involves the following steps (see Figure 3):

1. We train a new SentencePiece BPE tokenizer using a multilingual Islamic Corpus.

2. We find the intersection between the new Islamic tokenizer and the XLM-R4 tokenizer. All the tokens outside of the intersection (9k tokens) are added to the embedding matrix, and the weights for new tokens are assigned by averaging existing weights of subtokens from the XLM-R4 model.

3. We continue pre-training XLM-R4 using the domain-specific corpus mentioned above to get the XLM-R4-ID (Islamic domain) model. For more details on the hyperparameters, refer to Appendix A.

3 Domain-specific IR

To prepare the retrieval model, we utilize a dense retrieval approach (Karpukhin et al., 2020) that employs dual-encoder architecture (Bromley et al., 1993). We use the sentence transformer framework that adds a pooling layer on top of LLM embeddings and produces fixed-sized sentence embedding (Reimers and Gurevych, 2019). The loss function is formulated in the framework of contrastive learning that enables learning an embedding space that brings closer queries and their relevant passages and pushes further queries and irrelevant passages (van den Oord et al., 2018). For efficient training, we use in-batch negatives (Henderson et al., 2017; Gillick et al., 2019; Karpukhin et al., 2020). The transfer language of the XLM-R_{Base} is English, while XLM-R4-ID was adapted for the Islamic Domain, predominately using Arabic. We experiment with both English and Arabic as transfer languages to compare their transfer potential for solving the IR task at hand. We utilize the MS MARCO IR dataset, which contains more than half a million queries and a collection of 8.8M passages in English (Bajaj et al., 2018) to allow cross-lingual transfer from English and we use an Arabic machine-translated version of MS MARCO (Bonifacio et al., 2021) employing Arabic as transfer language. Consequently, we prepared four retrieval models, training XLM-R_{Base} and XLM-R4-ID, using English and Arabic MS MARCO (for hyperparameters details see Appendix A). For evaluation, we use Arabic QRCD (Qur'anic Reading Comprehension Dataset) (Malhas and Elsayed, 2020) as IR Dataset and its verified translation to English, Russian and Urdu. We use train and development sets (169 queries) for testing. As a collection for retrieval, we use the Holy Quran text (Arabic), Sahih International translation (English), Elmir Kuliev (Russian) and Ahmed Raza Khan (Urdu) are available on tanzil.net.[3] We evaluate the models' performance using Recall@100 and the order-aware

[3] https://tanzil.net/trans/

Model	EN		AR		RU		UR	
	Recall@100	MRR@10	Recall@100	MRR@10	Recall@100	MRR@10	Recall@100	MRR@10
XLM-R_{Base} (en)	18.7	34	2.94	6.94	17.9	31.8	20.4	33.7
XLM-R_{Base} (ar)	17.8	32.9	5.3	6.3	20	30.1	20.7	33.9
XLM-R4-ID (en)	27.2	43.8	28.6	**45.5**	**24.5**	34.7	26.8	40
XLM-R4-ID (ar)	**27.8**	**45.5**	**29.3**	**45.5**	24.1	**37.5**	**27.3**	**41.5**
ST/multilingual-mpnet-base-v2	21.6	34.3	4.8	5.2	17.2	22.4	13.5	19.1
ST/all-mpnet-base-v2	25	40.9	-	-	-	-	-	-

Table 3: Performance on in-domain IR dataset for four languages. The best scores are in bold, and color codes correspond to different languages.

metric MRR@10 (MS MARCO's official metric).

In Table 3, we compare different models, including the SentenceTransformer model (paraphrase-multilingual-mpnet-base-v2), which was trained by distilling knowledge from the teacher model paraphrase-mpnet-base-v2 and using XLM-R_{Base} as the student model. Additionally, we assess the performance of the monolingual teacher model paraphrase-mpnet-base-v2 in English. The table shows that both XLM-R4-ID models outperform the others, including the monolingual model (ST/all-mpnet-base-v2). Even though XLM-R4 is a reduced version of XLM-R_{Base}, it significantly outperforms XLM-R_{Base}. This improvement in performance shows that domain adaptation was beneficial. It is also important to mention that both the XLM-R_{Base} and the multilingual-mpnet-base-v2 models perform poorly in Arabic. This observation may indicate that domain shift might have a significant impact, particularly with the Arabic language. Moreover, we observe that XLM-R4-ID trained on the Arabic machine-translated version of MS MARCO outperforms XLM-R4-ID trained on English MS MARCO for all languages with one exception of Recall@100 metric for Russian. These results can be explained by the fact that a significant part of the corpus for domain adaptation was in Arabic (around 85%). We can suggest that Arabic can effectively function as a transfer language for the Islamic domain. For all subsequent sections of the paper and for deployment, we will be using XLM-R4-ID (ar).

4 Deploying Domain-Specific IR System

Using GPUs to train transformer-based LLMs and retrieval models is often a necessity. However, GPUs for inference in a production environment are cost-prohibitive, especially in non-profit organizations. Additionally, given supply availability to ensure the right size of cloud machines with GPUs often imposes a fixed set of resources in predefined bundles of size, which typically leads to vast over-provisioning and grossly underutilized resources. Our goal is to maximize software performance and resource efficiency on widely-used, cost-effective CPU servers. We argue that leveraging the ubiquity and flexibility of CPU servers makes it possible to build a system and improve efficiency independently of the underlying substrate, allowing deployment even on serverless infrastructure, which is predominately CPU-based.

4.1 Rust for Production AI Workloads

Production use of IR systems requires real-time processing capabilities. However, the main challenge of using state-of-the-art retrieval models in production is their high inference time. Deploying such models on resource-constrained devices is even more problematic. A few approaches like model quantization (Guo, 2018; Jacob et al., 2017; Bondarenko et al., 2021; Tian et al., 2023), embedding size compression (Zhu et al., 2018; Gupta et al., 2019; Kusupati et al., 2024; Li et al., 2024) can help to address this issue at the cost of model performance. However, in specific applications of semantic search, such as Islamic Domain, even a slight decrease in performance is highly undesirable. We argue that it is possible to improve inference times without compromising search quality. To minimize the trade-off between latency and performance, we leverage the advantages of the Rust language.

Rust is a safe and efficient systems programming language that addresses many pain points in other commonly used interpreted languages, such as Python, which imposes the presence of the Python interpreter in the production environment. Providing zero-cost abstractions to the hard-

SUT	Python (e.s.)	HNSW	SQ (e.s.)	PQ (e.s.)	Rust 1 w. (e.s.)	Rust 2 w. (e.s.)	Rust 4 w. (e.s.)	Rust 6 w. (e.s.)
Speedup	1x	5x	3.9x	9x	2.6x	3.8x	4.5x	4.9x
Recall	100%	90%	90%	85%	100%	100%	100%	100%

Table 4: Comparisons of SUTs for the speedup of retrieval against baseline and percentage of baseline Recall (e.s stands for exact search and w. for worker).

ware substrate with a lightweight memory footprint, idiomatically written Rust outperforms identical equivalents written in JVM-based languages such as Java (Perkel, 2020). The absence of garbage collection mechanics in Rust makes systems written in Rust more deterministic and better suited for production deployments in serverless and compact runtimes where compute is billed by milliseconds (Liang et al., 2024). The borrow checker of Rust eliminates an entire class of security vulnerabilities introduced by references outliving the data they point to. This feature guarantees safety, especially when writing concurrent and multiprocessing code, without sacrificing performance gains (Seidel and Beier, 2024; Jung et al., 2021; Abdi et al., 2024). Energy efficiency and reduced carbon footprint are other crucial features of using Rust in AI production workloads (Pereira et al., 2017).

4.2 System Design for Rust-based Semantic Search

Such libraries as Faiss[4] offer the best speedup using GPU architecture, which significantly increases deployment costs. Faiss also provides multi-threading capabilities but lacks native cost-efficient multiprocessing and true parallelism for individual search queries. The best CPU performance is achieved by sending queries in batches, which does not align with real-world online search. Utilizing the Rust language's capabilities enables us to implement a multiprocessing architecture efficiently and securely for our IR system. We built the system on top of the Candle framework,[5] a minimalist machine-learning framework for Rust. The system's architectural design goes as follows (see Figure 1):

1. The passages from the corpus are converted to embeddings and stored for caching during the search.

[4] https://ai.meta.com/tools/faiss/

[5] https://github.com/huggingface/candle

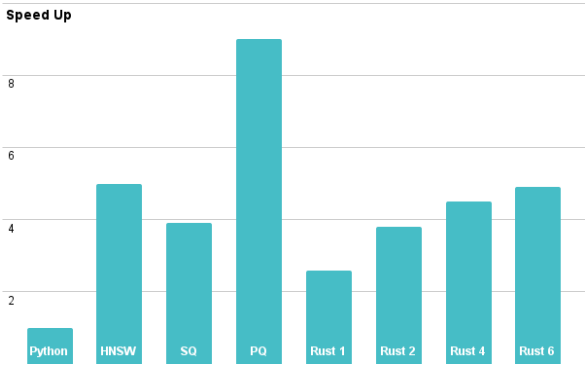

Figure 4: Speedup and Recall of SUTs.

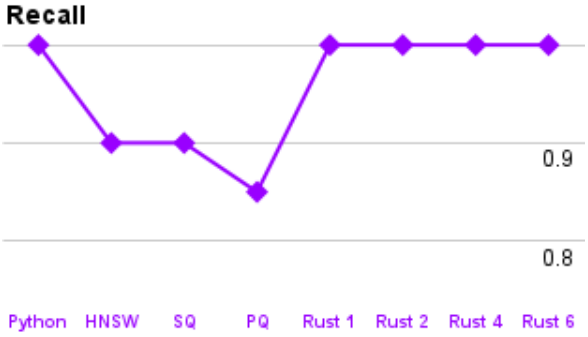

Figure 5: Speedup and Recall of SUTs.

2. The corpus embeddings are divided into chunks and distributed across the specified number of workers.

3. For multiprocessing during the search, an embedding of a search query is sent to each worker asynchronously.

4. Each worker conducts an exact search by comparing the query with each passage within the allocated chunk and then assigns a score using the similarity function.

5. The workers then return scores to the main thread as a list of tuples, each containing a score and a passage ID for sorting.

6. At the final stage, the scores are sorted in descending order, and the corresponding pas-

sages are returned to the user based on the topk parameter.

We compare our system's performance against Faiss implementation of the following algorithms: Hierarchical Navigable Small World graph (HNSW), Scalar Quantization with fp16 (SQ), and Product Quantization (PQ). To compare the Systems Under Test (SUTs), we assume the following conditions: the corpus and query embeddings are precomputed and preloaded in memory. To accommodate different minute fluctuations posed by potential hardware condition variance, we average ten runs of each system across test queries that are provided linearly. All the systems perform the search and retrieval using CPU-based architecture. To test on a bigger retrieval corpus (approx 50k passages), the dataset used for measuring the time of retrieval and Recall of SUTs is Hadith Question-Answer pairs (HAQA) (Alnefaie et al., 2023). The similarity function utilized during the search is cosine similarity. All the systems employ XLM-R4-ID (ar) as a retrieval model. The hardware used for the test is a cloud instance (1x NVIDIA A10) provided by Lambda Labs's public cloud.

Table 4 highlights the trade-off between retrieval time and performance for different SUTs. The main focus of comparison is the speed of retrieval. Python implementation of exact search is a baseline with its score for Recall@100 (Recall) taken as 100%. We can observe that the speedup of retrieval time of Faiss algorithms always comes at the cost of lower Recall. At the same time, the implementation of semantic search in Rust doesn't endure the trade-off between retrieval time and performance. Figure 5 illustrates the dip in Recall plot for the highest speedup of the PQ algorithm while Recall for Rust implementation stays flat at 100% for all instances. Moreover, a speedup of 2.6 times is achievable with Rust implementation without applying multiprocessing (using one worker), and further speedup is possible by adding more workers.

5 Related work

There is a substantial amount of work written on the topic of pre-training domain-specific LLM; some of them describe more costly approaches like pre-training a new LLM from scratch Gu et al. (2020); Beltagy et al. (2019), some more resource-efficient approaches like continued pre-training Lee et al. (2019); Huang et al. (2019), and there is a body of work that research methods of domain-adaptation in a low resource setting Poerner et al. (2020); Sachidananda et al. (2021); Pavlova (2023). The survey Zhao et al. (2022) covers in detail the topic of dense retrieval, discussing different types of models' architecture and training approaches, including the selection of high-quality negatives. There is a growing body of research on Rust Language memory-safe features that came to be known as fearless concurrency (Jung et al., 2021; Abdi et al., 2024; Evans et al., 2020; Perkel, 2020).

6 Conclusion

This work outlines the development of a non-profit multilingual IR system for the Islamic domain. We also address the challenges it presents and propose potential solutions for handling these challenges in low-resource settings. Our research demonstrates that utilizing continued pre-training and integrating new domain-specific vocabulary can help mitigate domain shift, even when pre-training on a small corpus. The retrieval model we built using a domain-adapted MLLM as a foundation exhibited better performance compared to general domain models. Additionally, we found that implementing language reduction can significantly decrease the model size without deteriorating performance. Furthermore, we showed that leveraging the multiprocessing capabilities of the Rust language can decrease inference time without compromising performance or requiring expensive acceleration hardware like GPUs.

Limitations

To measure the inference time and recall of SUTs we are restricted to using a smaller retrieval corpus (around 50k passages). The real size of the data for retrieval is above 150k passages.

Acknowledgment

Developing a multiprocessing CPU-based search with Rust would not have been possible without Mohamed Samir from SYWA AI. We would also like to express our gratitude to Osama Khalid from SYWA AI for assisting in verifying the quality of the Urdu translation of the QRCD queries. We extend our thanks to the anonymous Reviewers and the Area Chair for their valuable feedback and to the Program Chairs for promptly addressing and resolving all related matters.

References

Amine Abdaoui, Camille Pradel, and Grégoire Sigel. 2020. Load what you need: Smaller versions of mutililingual BERT. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 119–123, Online. Association for Computational Linguistics.

Javad Abdi, Gilead Posluns, Guozheng Zhang, Boxuan Wang, and Mark C. Jeffrey. 2024. When is parallelism fearless and zero-cost with rust? *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*.

Sarah Alnefaie, Eric Atwell, and Mohammad Ammar Alsalka. 2023. HAQA and QUQA: Constructing two Arabic question-answering corpora for the Quran and Hadith. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 90–97, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. Ms marco: A human generated machine reading comprehension dataset. *Preprint*, arXiv:1611.09268.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7947–7969, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Luiz Henrique Bonifacio, Israel Campiotti, Roberto de Alencar Lotufo, and Rodrigo Frassetto Nogueira. 2021. mmarco: A multilingual version of MS MARCO passage ranking dataset. *CoRR*, abs/2108.13897.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 737–744, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ana Nora Evans, Bradford Campbell, and Mary Lou Soffa. 2020. Is rust used safely by software developers? *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 246–257.

Daniel Gillick, Sayali Kulkarni, Larry Lansing, Alessandro Presta, Jason Baldridge, Eugene Ie, and Diego Garcia-Olano. 2019. Learning dense representations for entity retrieval. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 528–537, Hong Kong, China. Association for Computational Linguistics.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2020. Domain-specific language model pretraining for biomedical natural language processing. *CoRR*, abs/2007.15779.

Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks. *Preprint*, arXiv:1808.04752.

Vishwani Gupta, Sven Giesselbach, Stefan Rüping, and Christian Bauckhage. 2019. Improving word embeddings using kernel PCA. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 200–208, Florence, Italy. Association for Computational Linguistics.

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

Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. 2019. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *CoRR*, abs/1904.05342.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Preprint*, arXiv:1712.05877.

Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2021. Safe systems programming in rust. *Communications of the ACM*, 64:144 – 152.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2024. Matryoshka representation learning. *Preprint*, arXiv:2205.13147.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*.

Xianming Li, Zongxi Li, Jing Li, Haoran Xie, and Qing Li. 2024. Ese: Espresso sentence embeddings. *Preprint*, arXiv:2402.14776.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. 2020. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *CoRR*, abs/2002.11794.

Zhiying Liang, Vahab Jabrayilov, Aleksey Charapko, and Abutalib Aghayev. 2024. The cost of garbage collection for state machine replication. *Preprint*, arXiv:2405.11182.

Rana Malhas and Tamer Elsayed. 2020. Ayatec: building a reusable verse-based test collection for arabic question answering on the holy qur'an. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 19(6):1–21.

Vera Pavlova. 2023. Leveraging domain adaptation and data augmentation to improve qur'anic IR in English and Arabic. In *Proceedings of ArabicNLP 2023*, pages 76–88, Singapore (Hybrid). Association for Computational Linguistics.

Vera Pavlova and Mohammed Makhlouf. 2023. BIOptimus: Pre-training an optimal biomedical language model with curriculum learning for named entity recognition. In *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 337–349, Toronto, Canada. Association for Computational Linguistics.

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2017, page 256–267, New York, NY, USA. Association for Computing Machinery.

Jeffrey Perkel. 2020. Why scientists are turning to rust. *Nature*, 588:185 – 186.

Nina Poerner, Ulli Waltinger, and Hinrich Schütze. 2020. Inexpensive domain adaptation of pretrained language models: Case studies on biomedical NER and covid-19 QA. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1482–1490, Online. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Maxim Romanov and Masoumeh Seydi. 2019. Openiti: a machine-readable corpus of islamicate texts. *Zenodo, URL: https://doi. org/10.5281/zenodo*, 3082464.

Vin Sachidananda, Jason Kessler, and Yi-An Lai. 2021. Efficient domain adaptation of language models via adaptive tokenization. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, pages 155–165, Virtual. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Lukas Seidel and Julian Beier. 2024. Bringing rust to safety-critical systems in space. *Preprint*, arXiv:2405.18135.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages

4323–4332, Hong Kong, China. Association for Computational Linguistics.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136.

Rong Tian, Zijing Zhao, Weijie Liu, Haoyan Liu, Weiquan Mao, Zhe Zhao, and Kan Zhou. 2023. SAMP: A model inference toolkit of post-training quantization for text processing via self-adaptive mixed-precision. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 123–130, Singapore. Association for Computational Linguistics.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji rong Wen. 2022. Dense text retrieval based on pretrained language models: A survey. *ACM Transactions on Information Systems*, 42:1 – 60.

Yonghua Zhu, Xuejun Zhang, Ruili Wang, Wei Zheng, and Yingying Zhu. 2018. Self-representation and pca embedding for unsupervised feature selection. *World Wide Web*, 21(6):1675–1688.

A Appendix

Computing Infrastructure	1x H100 (80 GB)
Hyperparameter	**Assignment**
number of epochs	60
batch size	128
maximum learning rate	0.0005
learning rate optimizer	Adam
learning rate scheduler	None or Warmup linear
Weight decay	0.01
Warmup proportion	0.06
learning rate decay	linear

Table 5: Hyperparameters for pre-training of XLM-R4-ID model.

Computing Infrastructure	2 x NVIDIA RTX 3090 GPU
Hyperparameter	**Assignment**
number of epochs	10
batch size	8
learning rate	2e-5
weight decay	0.01

Table 6: Hyperparameters for fine-tuning on XNLI dataset.

Computing Infrastructure	1x H100 (80 GB)
Hyperparameter	**Assignment**
number of epochs	10
batch size	256
learning rate	2e-5
pooling	mean

Table 7: Hyperparameters for training retrieval models.