A retrieval-augmented-generation pipeline to help users query system-provided documentation

Tzolkin Garduno-Alvarado*, Gunnar Wolf†, Eddie Soulier*‡, Francis Rousseaux‡, Feliu Sagols§
*Université de Technologie de Troyes, France
Email: tzolkin.garduno_alvarado@utt.fr, eddie.soulier@utt.fr
†Instituto de Investigaciones Económicas, UNAM, México, CDMX
Email: gwolf@debian.org
‡Université de Reims Champagne-Ardenne, France
Email: francis.rousseaux@univ-reims.fr
§Departamento de Matemáticas, CINVESTAV-I.P.N. México, CDMX
Email: fsagols@math.cinvestav.edu.mx

***Abstract*—The increasing integration of AI into computing workflows demands a re-evaluation of traditional operating system design. In environments like Debian, users are often faced with a vast ecosystem of command-line tools, each accompanied by extensive manual pages (man pages) detailing usage, flags, and parameters. While comprehensive, these documents are frequently dense, verbose, and not well-suited for rapid onboarding or targeted queries. We propose a Retrieval Augmented Generation (RAG) pipeline to bridge this gap, enabling natural language interaction with system documentation. By combining tokenization, embedding, and dense retrieval with a language generation model, our system allows users to query tool usage in plain language and receive concise, contextually relevant responses. This approach streamlines tool discovery and comprehension, and represents a step toward more intelligent, user-aware operating systems.**

I. Introduction

Despite their depth and comprehensiveness, Unix *man pages* remain difficult to use for many. Their dense syntax and writing style hinder users from quickly finding the right command or flag. Not only that: *man pages* are most useful when a user knows *which tool* will help them tackle a given work; tools such as `man -k` or `apropos` don't scale with current systems' complexity. This is particularly problematic given the increasing number of system tools and utilities on a modern Linux installation, as it is often evident when a new user attempts to gain familiarity with their system. The Debian project, a prominent posterchild of the Free Software world, has been chosen to highlight this issue and to serve as a basis for the project we present.

II. Background and Related Work

Traditional Unix documentation tools such as `man` (collectively known as *man pages*), `apropos`, and `whatis` rely heavily on keyword matching and provide no semantic understanding of the user's intent. While helpful for exact lookups, they often fail for open-ended or goal-oriented queries.

There is a known *cultural style* to Unix documentation *man pages* are known to be "telegraphic but complete. It does not hold you by the hand, but it usually points in the right direction. The style assumes an active reader. (...) Unix programmers tend to be good at writing references, and most Unix documentation has the flavor of a reference or *aide memoire* (...) Read every word carefully, because whatever you want to know will probably be there, or deducible from what's there" [1]. While this mindset has long been criticized [2], Unix documentation continues to be written in a style focused on each of options discussed, often disregarding a fuller view.

The Debian GNU/Linux distribution consists of over 59 000 binary (precompiled) software packages [3], and while several tools for searching them [4], they all require the user to know which package set will fulfill their needs; there is no simple way to query an open *"how can I..."*-formed question. To achieve this, we propose a combination of Large Language Models (LLMs) as tools for data creation, document comprehension, relevant document retrieval, document classification and finally text interpretation.

As we were working on the final details of this work, we learnt the Red Hat Linux distribution, for its version 10 release, includes a tool similar in spirit to what we are presenting, named "Lightspeed" or "Command Line Assistant" [5]. The service seems not to be provided for local use, but by querying an on-line service [6]. Technical information regarding its architecture, training data, and usage modes is still not widely available, so while we feel obliged to refer to it, we cannot yet include any comparisons between our implementation and theirs.

III. Retrieval Augmented Generation

LLMs represent a significant advancement in Natural Language Processing (NLP) given their proficiency in language understanding and general knowledge. Some examples worth mentioning are the GPT series [7], the LLaMA series [8], and Gemini [9], among others. However, they are prone to generating false or fabricated information [10], [11] and often struggle with queries that require domain-specific knowledge or up-to-date information [12], [13]. When questions extend beyond the scope of their training they may fail to deliver accurate responses, Retrieval Augmented Generation (RAG) architectures offer a path forward. A RAG model combines a

retriever and a generator: the retriever fetches relevant documents from an external knowledge base, and the generator uses these to produce informed, coherent responses. This allows the model to access up-to-date or specialized knowledge beyond its training data, improving factual accuracy and scalability in tasks like question answering or summarization [14].

The diversity of techniques that can be subscribed under the umbrella of RAG techniques can be roughly classified as [14]:

- **Naive RAG**: Employs off-the-shelf retrievers and generators without end-to-end training. The pipeline is straightforward but often suffers from suboptimal alignment between retrieval and generation.
- **Advanced RAG**: Involves feedback loops, reranking, and fine-tuned modules. Retrievers and generators may be co-trained or jointly optimized to improve performance on downstream tasks.
- **Modular RAG**: Emphasizes separation of concerns. Components are independently replaceable, enabling flexible deployment across domains. Modular RAG supports hybrid knowledge integration and task-specific customization.

While encoder-decoder architectures excel in tasks that require tightly coupled comprehension and generation, they introduce computational overhead during retrieval due to their cross-attention mechanism between the query and candidate passages at inference time. In contrast, bi-encoder architectures offer a more scalable and efficient alternative for retrieval tasks by independently encoding queries and documents into dense vector representations. This coupled with the ease of comprehension-generation separation stages,[15] make the RAG architecture easily adaptable. This work proposes a modular RAG method setting in which the encoder models are fine-tuned for retrieving relevant *man pages* necessary to respond to a user query.

IV. System Architecture

A. Data description and preprocessing

The data used for this paper consists of *man pages* which where used for training data creation, training and evaluation. For this a full Debian mirror was used, that is, all binary packages in the current *unstable* Debian distribution as of early June, 2025, were downloaded and the contents of their `/usr/share/man/man1` directories were extracted. All of the manual pages were converted from the `troff` sources to regular text files, as flowed text is prefered for the approach we are following over annotated text.

Readers will note we limit our work to the `man1` directory, that is, to the *section 1 of the man pages*, limiting the knowledge ingested in our RAG to that of *executable programs or shell commands* [16]. This decision was taken due to the limits of the system we were able to do the training on; for productive use, a GPU with larger memory should be used. At the time of our sampling, the full `/usr/share/man` directory contained 128 458 individual man pages (109 544 in English and the rest translated to other languages), and the `man1` subdirectory contained 34 579 files. The largest manpage, for the Python package `ezdxf`, is 2.6MB long, and the shortest one, for `exiv2`, is only 162 bytes long; over 25 000 manpages (75% of them) are 5KB or shorter, confirming Raymond's characterization [1].

B. Synthetic training data generation

In the first stage of our pipeline, we generate synthetic training data to fine-tune dense retrieval models [17]. We begin by semantically chunking the *man pages* into coherent, token-limited segments using sentence-aware splitting [18], then uniformly sample 1% of the chunks. For each selected chunk, also called document, we use GPT-4o-mini [19] to generate a positive natural language query tailored to its contents. These $(query, document)$ pairs are then compiled into a training dataset suitable for supervised fine-tuning of bi-encoder models.

C. Encoder Fine-Tuning

For the second stage, four different dense encoder models were downloaded from the *HuggingFace* service in a binary, pre-trained format known as "checkpoint", and were fine-tuned based on our generated $(query, document)$ pairs, creating a shared embedding space. Fine-tuning is guided by a contrastive loss function. Given a batch of query embeddings $\{q_1, ..., q_n\}$ and corresponding document, or chunk, embeddings $\{d_1, ..., d_n\}$, the contrastive loss is defined as:

$$L = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{\exp\left(q_i \cdot d_i/\tau\right)}{\sum_{j=1}^{n}\exp\left(q_i \cdot d_j/\tau\right)}\right)$$

where τ is the temperature, which controlls the sharpness of the similarity distribution. This formulation promotes high dot-product similarity between true query-document pairs (q_i, d_i) while penalizing similarities to all other documents d_j with $j \neq i$ in the batch. Since the loss incorporates all possible combinations of query and document embeddings within a batch, it is crucial that each pair (q_i, d_i) represents a meaningful and relevant match. We ensure this by generating positive pairs on the creation of the fine tune training data, where a semantically aligned query is synthesized, thereby enforcing that each training pair corresponds to a high-confidence positive example. This pairing strategy strengthens the learning signal during contrastive training by avoiding ambiguous or noisy alignments.

Embeddings are computed using mean pooling over the token representations of each sequence. Model parameters are optimized using the AdamW algorithm [20], which combines Adam's adaptive learning rate with decoupled weight decay for improved generalization. Training is conducted over 50 epochs with mini-batches, and all hyperparameters (e.g., learning rate, batch size, maximum sequence length) are specified via a structured configuration file. The resulting fine-tuned models are stored locally and later used for dense retrieval.

This synthetic training approach enables the creation of task-specific, scalable retrieval fo documents, or chunks, with-

TABLE I
SIZE OF THE RESULTING TRAINED MODELS AND VECTOR STORES

Model	Trained model	Vector store
T5-small	136M	860M
MiniLM-L6-v2	88M	701M
BGE-small-en	129M	701M
Multilingual-e5-small	470M	701M

out the need for manual annotation [17], and significantly enhances the model's ability to retrieve relevant passages.

D. Vector-store index

This step converts preprocessed documents into dense vector representations using the previously fine-tuned models and stores them in a FAISS index [21] to enable efficient semantic retrieval. Each document is encoded and the resulting embeddings are pooled and indexed.

The system integrates the downloaded models described in Subsection IV-C with LangChain's FAISS wrapper, allowing for scalable and configurable indexing. The result is a reusable vector store ready for query-time retrieval.

The resulting training models and vector store sizes are presented in Table I.

E. Document retrieval

Once the vector store has been populated, we perform an automated dense retrieval of document chunks using a set of queries synthetically generated that are of types positive, absolute negative and mildly negative. Then we embedd batches of queries into dense vectors and retrieve the $top - k$ nearest neighbors from the vector store based on $\mathcal{L}_2$ similarity.

F. Generation

In the final stage of the pipeline, we perform classification and synthesis of the document chunks retrieved in response to user queries. This step aims to filter out semantically irrelevant content and generate concise, query-specific summaries that can be consumed directly or used in downstream generation tasks.

We begin by applying a zero-shot classification model (*facebook/bart-large-mnli*) to each retrieved chunk, using the query as contextual grounding. Given a query q and a document chunk d, we evaluate whether the chunk answers the query using the hypothesis template: "This text answers the question: *'q'* – yes/no". Chunks are retained only if the classifier assigns the "yes" label a confidence score greater than 0.6. This filtering process allows us to isolate only the most relevant results from the retriever step.

For each retained chunk, we then generate a natural language synthesis using a pretrained summarization model (*facebook/bart-base*). These summaries are constrained to 40–150 tokens and are designed to extract the core insight or fact relevant to the original query.

The outcome of this step is a structured dataset containing the query, the filtered classification label, and a synthesized summary of the supporting evidence. This dual-stage process of zero-shot filtering and abstractive summarization ensures that the retrieved outputs are both accurate and information-dense, significantly improving the quality and utility of the RAG system's final output.

V. ENCODER MODEL SELECTION

Retrieval Augmented Generation (RAG) critically depends on the choice of encoder models, which must effectively and efficiently map both queries and document chunks into a shared semantic embedding space. In this section, we evaluate four pre-trained models for dense retrieval, to be able to compare their architectures, suitability for RAG pipelines, and computational demands.

- **google/t5-small** [22]: Originally designed as an encoder-decoder model for general-purpose text-to-text tasks, this model is not optimized for embedding-only operations. Although the encoder component can be repurposed for retrieval, the overall architecture is less efficient for embedding tasks in production settings.
- **sentence-transformers/all-MiniLM-L6-v2** [23]: A lightweight transformer fine-tuned for semantic textual similarity tasks. It delivers fast, compact embeddings, making it well-suited for latency-sensitive or resource-constrained environments. However, it may show limitations when dealing with specialized or technical language.
- **BAAI/bge-small-en** [24]: A retrieval-focused encoder pre-trained with instruction tuning to better capture user intent. It exhibits strong performance on general-purpose retrieval tasks and shows excellent alignment with natural-language queries, making it particularly suitable for real-world knowledge retrieval.
- **intfloat/multilingual-e5-small** [25]: A multilingual bi-encoder trained with contrastive learning objectives. Its robustness in zero-shot settings and compatibility with diverse linguistic inputs make it a strong candidate for multilingual RAG applications.

Table II summarizes key characteristics of the models considered, including their architectural type, number of parameters, encoder setup, and best-fit use case.

VI. EVALUATION

To assess the performance of the retrievers used in the RAG pipeline, we need to distinguish between relevant and irrelevant documents with respect to a given query, thus a binary classification evaluation approach was used. For this, we used the following:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$

Where TP means *True Positives*, FP means *False Positives*, and FN means *False Negatives*.

Receiver Operating Characteristic (ROC) curves are used for evaluation. These curves offer a threshold-independent view of

Model	Architecture	Params (M)	Type	Best Use
T5-small	Encoder-Decoder	60.5	Seq2Seq	Generation, Full RAG
MiniLM-L6-v2	Encoder-only	22.7	Bi-Encoder	Local embedding
BGE-small-en	Encoder-only	33.4	Bi-Encoder	Query alignment
Multilingual-e5-small	Encoder-only	118	Bi-Encoder	Multilingual RAG

TABLE II
OVERVIEW OF MODEL ARCHITECTURES, PARAMETER SIZES, AND RECOMMENDED USE CASES. ALL PERFORMANCE METRICS WERE COLLECTED IN INFERENCE MODE ON AN NVIDIA T4 GPU.

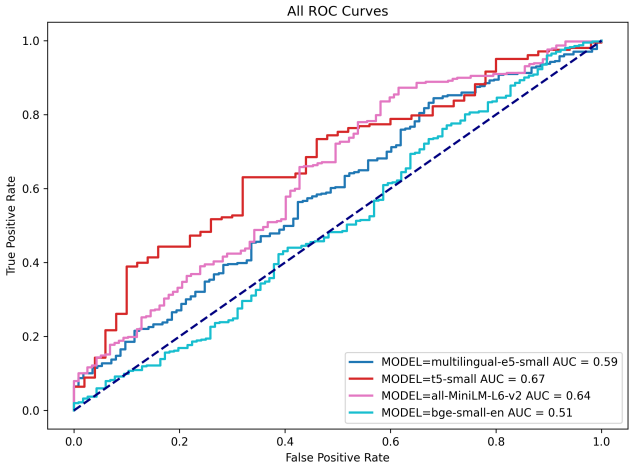

Fig. 1. Simulation results for the network.

the model's discriminative capacity. For each retrieval model, system outputs were matched with synthetic ground truth labels on pairs *man_page–query*. True positives were defined as those documents labeled `positive` in the ground truth. For a given retrieval model, documents were ranked according to their $\mathcal{L}_2$ distance to the query embedding, and thresholding was used to compute the trade-off between true positive and false positive rates. This allows for the construction of ROC curves, and the computation of the Area Under the Curve (AUC) as a summary performance metric. The use of ROC curves is justified by the nature of the evaluation task, which involves finding the right *chunks* answering the user-provided query.

Finally, the AUC score, which quantifies the overall ranking quality of the retriever: a score of 1.0 indicates perfect ranking, whereas a score of 0.5 reflects performance no better than random selection, was used. This is particularly valuable in early-stage evaluations, where threshold selection might be arbitrary or application-dependent. Table III presents the AUC scores obtained for the four fine tuned models.

TABLE III
AUC SCORES FOR DIFFERENT PRETRAINED MODELS ON THE RETRIEVAL EVALUATION TASK

Model	AUC Score
`multilingual-e5-small`	0.5883
`t5-small`	0.6722
`all-MiniLM-L6-v2`	0.6380
`bge-small-en`	0.5066

Among the tested models, `t5-small` achieved the highest AUC (0.6722), indicating relatively better discriminative performance in ranking relevant documents higher. `bge-small-en`, on the other hand, performed close to random (AUC = 0.5066), suggesting its inadequacy for this specific retrieval task in its current form. These findings highlight the importance of empirical evaluation when selecting models for RAG pipelines, as performance varies significantly across model architectures and training regimes.

VII. DISCUSSION

Our approach offers a significant usability improvement for Unix documentation, but not without its trade-offs. While semantic retrieval is powerful, it can miss low-frequency technical edge cases. Further, generative models are prone to hallucinating content if used for summarization.

The separation of the encoding stage allows the bi-encoder to precompute and index document embeddings, enabling fast approximate nearest-neighbor search during retrieval. Although this architecture does not leverage joint attention between query and context at inference, its performance heavily depends on the alignment quality of the learned embedding space.

Fine-tuning the bi-encoder with synthetic (query, chunk) pairs, as described above, directly optimizes this alignment. The encoder learns to map semantically related queries and passages to nearby points in the vector space, improving retrieval accuracy. This makes bi-encoders particularly suitable for RAG pipelines, where fast and accurate retrieval is a prerequisite for high-quality downstream generation. Furthermore, the use of contrastive learning reinforces this alignment by explicitly penalizing mismatched pairs, ensuring that the embedding space reflects meaningful semantic structure even without shared attention layers.

We have published the scripts used to download the man pages, generate, train and query the model at https://github.com/tzolkin-garduno/RAG_man_page; all models were downloaded from *Huggingface* (https://huggingface.co/).

VIII. FUTURE WORK

The implementation hereby described is all but an early proof of concept. In particular, the decoder architecture is to be fine tuned along with the bi-encoder component. We suspect that the results obtained from fine tuning a coupled bi-encoder and decoder architecture will yield better results, yet this remains to be verified.

We plan to include the other relevant sections of the man pages [16] (including multilingual content), improve chunk

linking for contextual navigation, and integrate with CLI environments like Bash and Fish. We also envision a feedback loop where user interactions help improve relevance over time.

IX. Conclusion

By bringing AI-driven retrieval to Unix documentation, we attempt to make system tools more accessible and discoverable without compromising their technical accuracy. This work demonstrates how traditional software ecosystems can be modernized using NLP, benefiting both new users and seasoned administrators.

Acknowledgment

The authors would like to thank the Felipe Esquivel Fund for Scientific and Cultural Support for kindly providing the needed hardware to train the vector store and run the service.

Acronyms Used

AUC Area Under the Curve
LLM Large Language Model
NLP Natural Language Processing
RAG Retrieval Augmented Generation
ROC Receiver Operating Characteristic

References

[1] E. S. Raymond, *The Art of Unix Programming*. Addison-Wesley Professional, 2003, pp. 461–462. [Online]. Available: https://archive.org/details/ost-computer-science-the_art_of_unix_programming-1

[2] J. Spolsky, "Biculturalism," 2003. [Online]. Available: https://www.joelonsoftware.com/2003/12/14/biculturalism/

[3] D. Project, "About debian," 2025. [Online]. Available: https://www.debian.org/intro/about

[4] Maddox, "How to search for debian packages in linux." [Online]. Available: https://itslinuxfoss.com/search-debian-packages-linux/

[5] B. Smith, "Let ai teach you linux with red hat enterprise linux lightspeed," 5 2025. [Online]. Available: https://www.redhat.com/en/blog/red-hat-enterprise-linux-lightspeed-let-ai-teach-you-linux

[6] L. Proven, "As rhel clones hit version 10, rocky and alma chart diverging paths," 5 2025. [Online]. Available: https://www.theregister.com/2025/06/14/rocky_alma_and_rhel_10/

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[8] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open Foundation and Fine-Tuned Chat Models," Jul. 2023, arXiv:2307.09288 [cs]. [Online]. Available: http://arxiv.org/abs/2307.09288

[9] R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, and others, "Gemini: A family of highly capable multimodal models." *arXiv:2312.11805v5*, pp. 24–28, May 2025.

[10] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, L. Wang, A. T. Luu, W. Bi, F. Shi, and S. Shi, "Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models," Sep. 2023, arXiv:2309.01219 [cs]. [Online]. Available: http://arxiv.org/abs/2309.01219

[11] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, Mar. 2025, arXiv:2311.05232 [cs]. [Online]. Available: http://arxiv.org/abs/2311.05232

[12] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. Rana, and S. Nanayakkara, "Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1–17, Jan. 2023, _eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00530/2067834/tacl_a_00530.pdf. [Online]. Available: https://doi.org/10.1162/tacl_a_00530

[13] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel, "Large language models struggle to learn long-tail knowledge," in *International Conference on Machine Learning*. PMLR, 2023, pp. 15 696–15 707.

[14] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," Mar. 2024, arXiv:2312.10997 [cs]. [Online]. Available: http://arxiv.org/abs/2312.10997

[15] M. Elfeki, R. Liu, and C. Voegele, "Return of the Encoder: Maximizing Parameter Efficiency for SLMs," Jan. 2025, arXiv:2501.16273 [cs]. [Online]. Available: http://arxiv.org/abs/2501.16273

[16] C. Watson, F. Polacco, G. Wilford, R. Faith, and J. Eaton, "man - an interface to the system reference manuals," 2001. [Online]. Available: https://manpages.debian.org/bookworm/man-db/man.1.en.html

[17] Y. Lu, L. Chen, Y. Zhang, M. Shen, H. Wang, X. Wang, C. v. Rechem, T. Fu, and W. Wei, "Machine Learning for Synthetic Data Generation: A Review," Apr. 2025, arXiv:2302.04062 [cs]. [Online]. Available: http://arxiv.org/abs/2302.04062

[18] R. Qu, R. Tu, and F. Bao, "Is Semantic Chunking Worth the Computational Cost?" Oct. 2024, arXiv:2410.13070 [cs]. [Online]. Available: http://arxiv.org/abs/2410.13070

[19] OpenAI, "GPT-4o mini: advancing cost-efficient intelligence," Jul. 2024. [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

[20] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," Jan. 2019, arXiv:1711.05101 [cs]. [Online]. Available: http://arxiv.org/abs/1711.05101

[21] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The Faiss library," Feb. 2025, arXiv:2401.08281 [cs]. [Online]. Available: http://arxiv.org/abs/2401.08281

[22] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[23] H. Face, "sentence-transformers/all-MiniLM-L6-v2," 2022. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[24] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, "C-Pack: Packaged Resources To Advance General Chinese Embedding," 2023, _eprint: 2309.07597.

[25] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Multilingual E5 Text Embeddings: A Technical Report," *arXiv preprint arXiv:2402.05672*, 2024.