RAGthoven: A Configurable Toolkit for RAG-enabled LLM Experimentation

Gregor Karetka[1,2] **Demetris Skottis**[2] **Lucia Dutková**[2,3]
Peter Hraška[2] **Marek Šuppa**[2,4]
[1]Brno University of Technology [2]Cisco Systems
[3]Linnaeus University [4]Comenius University in Bratislava
Correspondence: marek@suppa.sk

Abstract

Large Language Models (LLMs) have significantly altered the landscape of Natural Language Processing (NLP), having topped the benchmarks of many standard tasks and problems, particularly when used in combination with Retrieval Augmented Generation (RAG). Despite their impressive performance and relative simplicity, its use as a baseline method has not been extensive. One of the reasons might be that adapting and optimizing RAG-based pipelines for specific NLP tasks generally requires custom development which is difficult to scale. In this work we introduce RAGthoven, a tool for automatic evaluation of RAG-based pipelines. It provides a simple yet powerful abstraction, which allows the user to start the evaluation process with nothing more than a single configuration file. To demonstrate its usefulness we conduct three case studies spanning text classification, question answering and code generation usecases. We release the code, as well as the documentation and tutorials, at https://github.com/ragthoven-dev/ragthoven [1]

1 Introduction

Large Language Models (LLMs), when combined with Retrieval Augmented Generation (RAG) (Lewis et al., 2020), have consistently demonstrated state-of-the-art performance across numerous NLP tasks, including question answering (Siriwardhana et al., 2023), text classification (Loukas et al., 2023), and code generation (Bassamzadeh and Methani, 2024). However, despite their proven success, as demonstrated by various surveys such as (Gao et al., 2023), RAG in combination with LLMs remains significantly underutilized, particularly in research settings where establishing strong baselines is crucial. Standardized use cases such as benchmarking new datasets or competing in shared tasks often overlook RAG's potential due to the complexities involved in its implementation. Researchers and practitioners frequently default to simpler models, leaving the full power of retrieval-augmented systems untapped.

One reason for this underutilization might be the significant custom development required to tailor RAG pipelines to specific tasks. Most RAG frameworks require users to write custom code, adjust for task-specific nuances, and optimize the pipeline stages. These barriers deter many from using RAG as a baseline method, despite its potential to improve performance in shared tasks, where strong, consistent baselines are essential. That in turn limits its broader experimentation with RAG in academia and industry, especially for those working with new datasets or aiming to quickly establish competitive baselines for NLP tasks.

The contributions of this paper address these challenges through the introduction of RAGthoven, a toolkit designed to automate the evaluation of RAG-based pipelines. First, RAGthoven provides an easy-to-use configuration-driven interface that abstracts away the complexity of RAG implementation, making it accessible even to users with limited programming expertise. Second, it supports a broad range of NLP tasks out of the box, allowing researchers to quickly set up robust baselines for new datasets or shared tasks. Third, RAGthoven includes modules for indexing, retrieval, re-ranking, and generation, all of which can be configured independently, ensuring flexibility and adaptability for task-specific optimization. Through the tool's case studies in text classification, question answering, and code generation, we demonstrate its ability to simplify RAG-based evaluations while maintaining competitive performance, thereby promoting wider adoption of retrieval-augmented models in research and practical applications.

[1]A walkthrough video can be found at https://ragthoven-dev.github.io/walkthrough.mp4

Proceedings of the 31st International Conference on Computational Linguistics: System Demonstrations, pages 117–125
January 19–24, 2025.

2 Related Work

Since its introduction, Retrieval Augmented Generation (Lee et al., 2019; Lewis et al., 2020; Guu et al., 2020) has been found useful for varied set of tasks, such as language modeling (Ram et al., 2023), machine translation (Cheng et al., 2024; Wang et al., 2022), text summarization (Li et al., 2023), question answering (Huang et al., 2023), information extraction (Wang et al., 2021; Glass et al., 2023), dialogue systems (King and Flanigan, 2023), as well as text classification (Abdullahi et al., 2024).

To support the diverse needs of the various tasks that could benefit from Retrieval Augmented Generation, a large number of frameworks and platforms have been introduced. These include extensive mature platforms such as for instance `haystack` (Pietsch et al., 2019), `Verba` [2] or `RAGFlow` [3] which are tailored towards production-oriented and/or document understanding use cases, as well as `cognita`[4], `canopy` [5], `fastRAG` (Izsak et al., 2023) and `FlashRAG` (Jin et al., 2024) which are oriented more towards experimentation. Virtually all of these frameworks and platforms require the end user to interact with them by producing custom Python code tailored to a specific task. This, however, makes them inaccessible to a significant number of users who might benefit from being able to experiment with Retrieval Augmented Generation without being proficient in Python. To the best of our knowledge RAGthoven is the only toolkit that, while also usable as a Python library, has been built with the "no code" ethos in mind and as such can be utilized by larger target audience of non-programmers as well.

3 System Description

RAGthoven is composed of four key modules: indexing, retrieval, re-ranking, and generation. Each module can be configured independently, allowing users to adapt the pipeline to specific NLP tasks. The following sections describe the function and configuration of each module in the RAGthoven pipeline.

3.1 Indexing

The indexing module is responsible for creating a searchable representation of the dataset, which is generally loaded using the `datasets` library (Lhoest et al., 2021). This module supports both vector-based indexing, which utilizes dense embeddings, and traditional text-based indexing, which leverages tokenized text data. Users can configure the indexing module to use different models for vectorization or tokenization, depending on the nature of the data and the retrieval requirements. The indexing module is powered by the ChromaDB[6] and can be configured to make use of any of the embedding models available on HuggingFace Hub. Importantly, the same model configuration is shared between the indexing and retrieval modules to ensure consistency in data representation.

3.2 Retrieval

The retrieval module fetches relevant documents or pieces of information from the indexed data. It supports vector-based retrieval, utilizing similarity search on dense embeddings. Users can configure the retrieval module with the same model used in indexing, ensuring alignment between how the data is indexed and retrieved. By default, the `sentence-transformers/all-MiniLM-L6-v2` model is utilized for embedding generation. The module can also be configured to output a specific number of retrieved items from the index based on the input using the `k` option in the `embed` section, making it a hyperparameter for end-users to optimize.

3.3 Re-ranking

After the retrieval step, the re-ranking module prioritizes the retrieved documents, ensuring that the most relevant items are forwarded to the generation module. Re-ranking is particularly useful when a large volume of data is retrieved, and further refinement is required to improve the relevance of the final output. In such a case, the Retrieval module is generally instructed to retrieve a larger number of samples whereas the re-ranking module is then tasked with reordering them based on how relevant they are to the input query. By default, the `ms-marco-MiniLM-L-12-v2` model is utilized to re-rank the samples obtained in the previous step. The number of items to finally return can again be configured using the `k` option in the `rerank` section.

This module can be configured to use different re-ranking models supported by the `flashrank` li-

[2] https://github.com/weaviate/Verba
[3] https://github.com/infiniflow/ragflow
[4] https://github.com/truefoundry/cognita
[5] https://github.com/pinecone-io/canopy
[6] https://www.trychroma.com/

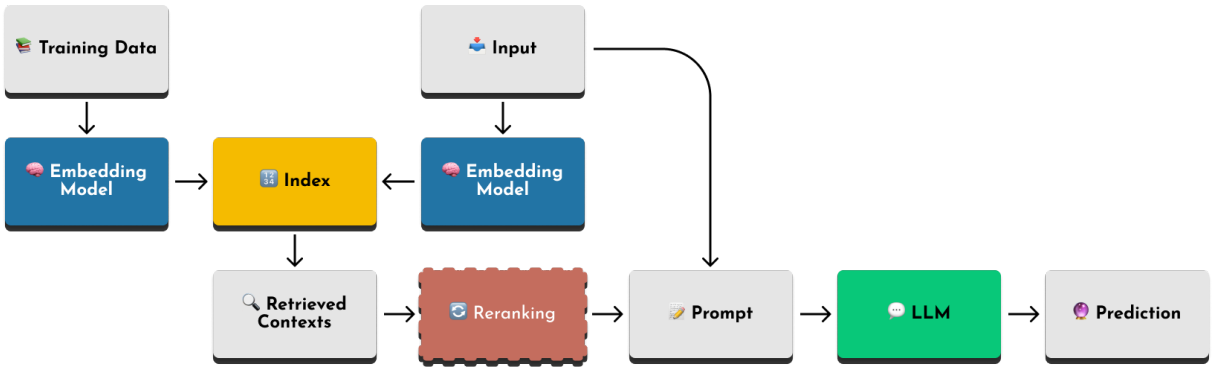

Figure 1: A diagram of a standard RAG system. Note that the Index and Re-ranking steps are optional in principle, but have been included in the diagram for clarity.

brary (Damodaran, 2023), allowing users to adjust ranking mechanisms based on task-specific needs or preferences.

3.4 Generation

The generation module uses a pre-trained LLM to generate final outputs based on the re-ranked documents. As it is built on top of the `litellm` library[7], it can be configured to make use of various LLM API providers, such as OpenAI, Anthropic, Google Vertex or Cohere. The specific model parameters, such as number of tokens to geenrate, or the temperature to use for generation, as well as the system (`sprompt`) and user (`uprompt`) prompt to be used can be specified as well in the `llm` section of the configuration file. In the interest of reproducibility, the temperature defaults to 0.

3.5 Configuration

The central part of the RAGthoven toolkit is the configuration file, a sample of which can be found in Figure 2. The configuration file aims to be a "one-stop shop" for defining the experiment RAGthoven will execute, describing both the input dataset as well as the hyperparameters to be used.

The configuration file can also be used to describe multiple hyperparameter options RAGthoven ought to iterate over during its execution. A sample configuration that makes use of this approach can be seen in Figure 3. As we can see, compared to Figure 2 the values for `k` and `model` have become of the `list` type instead of being `int` or `str`, respectively. The RAGthoven identifies these as configuration parameters that are to be iterated over, and proceeds with running the respective experiments, using a specific combination of the parameters in each of them. Thanks to this succint format, the configuration definition outlined in Figure 3 can be viewed as representing 24 fully interpolated configurations as described in Figure 2.

4 Case Studies

We present three case studies demonstrating RAGthoven's effectiveness and ease of use. These case studies, drawn from well-defined Shared Tasks, showcase how RAGthoven simplifies the implementation of competitive baselines.

4.1 Climate Activism Hate Speech Detection

The Hate Speech Detection subtask from the CASE 2024 Shared Task on Climate Activism Stance and Hate Event Detection required participants to classify tweets related to climate activism into two categories: `Hate` and `Non-Hate`. The dataset consisted of labeled tweets, with 6,385 examples of `Non-Hate` and 899 examples of `Hate` in the training set. The tweets were drawn from discussions related to climate activism, with a notable proportion of messages centered around prominent figures such as Greta Thunberg. A more detailed overview of the task can be found at (Thapa et al., 2024) and an extensive description of the approached explored in the case study can be found in (Suppa et al., 2024).

4.1.1 Baseline and Task Approaches

In this case study, GPT-4 was evaluated on its ability to perform hate speech detection using a zero-shot setup, few-shot prompting, and retrieval-augmented generation (RAG). The baseline established for the subtask was the majority class and the baseline model was GPT-4 in a zero-shot setting, where no specific examples were provided in the prompt. The RAGthoven toolkit

[7] https://github.com/BerriAI/litellm

```yaml
training_data:
  dataset: "Jinyan1/COLING_2025_MGT_en"
  input_feature: "text"
  label_feature: "label"
  split_name: "train"
validation_data:
  input_feature: "text"
  split_name: "valid"
  dataset: "data/data.jsonl"
embed:
  k: 10
rerank:
  k: 3
llm:
  sprompt: |
    You are a helpful assistant that classifies
    text as Human or Machine generated.

    Here are some of very similar texts and
    their respective labels:
    {{ examples }}
  uprompt: |
    Please determine the category of the text
    use "0" for Human and "1" for Machine Generated:

    {{ text }}
    ANSWER ONLY WITH SINGLE NUMBER!
```

Figure 2: A sample RAGthoven configuration file. Note that the configuration describes a real use-case – a Shared Task on "Binary Multilingual Machine-Generated Text Detection" introduced as part of the COLING 2025 Workshop on Detecting AI Generated Content. More details can be found at https://genai-content-detection.gitlab.io/sharedtasks

was configured to index the dataset using the `sentence-transformers/all-MiniLM-L6-v2` model and retrieve relevant examples from the dataset to augment the input prompts. Re-ranking was performed using the `ms-marco-MiniLM-L-12-v2` model to prioritize the most relevant retrieved examples for hate speech classification.

4.1.2 Results and Analysis

The results, as shown in Table 1, demonstrate the effectiveness of retrieval-augmented generation in improving hate speech classification performance. The baseline zero-shot GPT-4 model achieved an accuracy of 93.5% and an F1 score of 85.6. When augmented with retrieval, the performance improved, with the best RAG configuration (`RAG all`, $k = 6$) achieving an F1 score of 88.1, outperforming the few-shot model. Notably, re-ranking did not enhance performance over basic retrieval, suggesting that the retrieval mechanism alone contributed the most significant gains.

```yaml
training_data:
  dataset: "Jinyan1/COLING_2025_MGT_en"
  input_feature: "text"
  label_feature: "label"
  split_name: "train"
validation_data:
  input_feature: "text"
  split_name: "valid"
  dataset: "data/data.jsonl"
embed:
  k: [10, 20, 30]
  model: [
    "sentence-transformers/all-MiniLM-L6-v2",
    "sentence-transformers/all-mpnet-base-v1"
  ]
rerank:
  k: [3, 5]
  model: [
    "ms-marco-MiniLM-L-12-v2",
    "ms-marco-MultiBERT-L-12"
  ]
llm:
  sprompt: |
    You are a helpful assistant that classifies
    text as Human or Machine generated.

    Here are some of very similar texts and
    their respective labels:
    {{ examples }}
  uprompt: |
    Determine the text category and respond
    "0" for Human and "1" for Machine Generated:

    {{ text }}
    ANSWER WITH SINGLE NUMBER ONLY!
```

Figure 3: A sample RAGthoven configuration file denoting different hyperparameters to run experiments over. Note in particular the `k` and `model` keys in the `embed` and `rerank` sections.

The results indicate that retrieval augmentation, particularly with `all-mpnet-base-v2` embeddings, substantially improves performance in hate speech detection, confirming that task-specific information retrieval helps GPT-4 adapt to the nuances of the dataset. However, re-ranking did not yield additional benefits in this task, likely due to the already high relevance of the top-retrieved examples. The full RAGthoven configuration can be found in Appendix A and Figure 4.

4.2 Multilingual Text Detoxification 2024 Case Study

The Multilingual Text Detoxification (TextDetox) 2024 task aimed to address the problem of toxicity in text by rewriting toxic content to maintain its original meaning while removing offensive language. Participants were provided with multilingual datasets containing toxic texts from various languages such as German, Chinese, Russian, and

Model	Acc	P	R	F1
baseline	0.901	-	-	0.708
GPT-4 Zero-Shot	0.935	0.835	0.880	0.856
+ Few-Shot (k=6)	0.932	0.826	0.895	0.855
+ RAG (k=6)	0.941	0.851	0.889	0.868
+ RAG all (k=6)	**0.948**	**0.866**	**0.899**	**0.881**
+ RAG + Re-Rank (k=6)	0.941	0.853	0.877	0.864

Table 1: Results for the Hate Speech Detection Task in terms of Accuracy (Acc), Precision (P), Recall (R) and F1 score. The best performance is bolded.

more. The objective was to generate neutralized outputs that were both non-toxic and contextually accurate. Automatic and manual evaluations were performed by the task organizers to assess the quality of the submissions. A more detailed overview of the task itself can be found in (Dementieva et al., 2024) and an extensive description of the approach our case study is based on can be found in (Řehulka and Šuppa, 2024).

4.2.1 Baseline and Task Approaches

The baseline approach chosen by the organizers was the "duplicate" baseline, which simply responded with the original text, without making any changes.

Our baseline approach utilized a zero-shot setup of the Llama3 model (Dubey et al., 2024). This configuration simply provided instructions for detoxification without any specific contextual examples. However, this method proved suboptimal, as the model tended to rewrite the entire sentence, altering the meaning significantly. To address this, we employed few-shot prompting with relevant examples in English and other languages.

The system was then enhanced using Retrieval Augmented Generation (RAG), which integrated toxic-neutral text pairs from external datasets, including `textdetox/multilingual_paradetox` (TextDetox, 2023). For each input, dynamically generated prompts included relevant detoxified examples and instructions to maintain the original context. This approach significantly improved detoxification results across all languages.

To further refine the outputs, we also introduced a *reverse* approach. In this method, retrieved examples were presented in reverse order in the prompt, placing the most relevant examples at the end. This subtle change helped the model prioritize closer matches in detoxification, particularly for challenging languages like Russian.

4.2.2 Results and Analysis

In the final evaluation, we assessed the performance using three metrics: (1) the absence of toxic content, measured via an `xlm-roberta-large` classifier, (2) semantic preservation, quantified by cosine similarity using LaBSE embeddings, and (3) grammatical correctness, measured using the ChrF metric. The composite score was the product of these three metrics. Table 2 summarizes the performance across different languages for the various configurations.

The reverse approach, where we reordered examples in the prompt, improved the model's performance across several languages, particularly German and Russian. The only language in which our system did not manage to beat the original, organizer-provided baseline was Amharic. We attribute this to the specific script of the langauge as well as its low-resource nature.

4.2.3 Manual Evaluation

In addition to automatic evaluation, a manual evaluation of 100 detoxified samples per language was conducted. In this evaluation, our system ranked 5th overall, with notable performance in German, where it surpassed the human reference. The reverse approach further contributed to gains in this manual evaluation.

In conclusion, the combination of RAG with dynamic prompt generation and the reverse approach allowed us to effectively tackle the Multilingual Text Detoxification 2024 task. Our system demonstrated strong cross-lingual detoxification performance, especially for German and Chinese, showing the flexibility and power of the RAGthoven system in handling diverse NLP tasks. The full RAGthoven configuration can be found in Appendix A and Figure 5.

4.3 Code Generation for Optimization Problems

In this case study, we apply RAGthoven to the task of automating code generation for solving optimization problems. This task was inspired by the ICML 2024 Automated Math Reasoning Challenge, specifically Track 3, which involves generating Python code that solves optimization problems using the PuLP library[8]. The challenge requires models to understand problem descriptions presented in natural language and produce executable

[8] https://www.codabench.org/competitions/2438/

Model	avg	en	es	de	zh	ar	hi	uk	ru	am
baseline	0.126	0.061	0.090	0.287	0.069	0.294	0.035	0.032	0.048	**0.217**
Llama 3	0.380	0.525	0.448	0.530	0.161	0.488	0.185	0.507	0.461	0.112
+ RAG	0.403	**0.527**	0.483	**0.576**	0.152	0.483	0.176	0.534	0.504	0.193
+ reverse	**0.420**	**0.527**	**0.499**	0.563	**0.169**	**0.538**	**0.193**	**0.602**	**0.523**	0.167

Table 2: Performance across different languages for various models and configurations (average scores of automatic evaluation) in the Text Detoxification 2024 Shared Task. The highest performance per language is boldfaced.

Python code that computes optimal solutions using specific solvers. For example, the input description might include details such as the objective function and constraints, with the expected output being Python code that computes the solution and prints specific variables.

4.3.1 Baseline and Task Approaches

This Shared Task did not feature a baseline established by its organizers.

Our baseline for this task involved directly prompting models like GPT-4 and GPT-4o using a system prompt that instructed the LLM to generate Python code for optimization problems. The model used retrieval to augment its output by pulling relevant examples from a dataset indexed using the `all-MiniLM-L6-v2` model. If the initial code generation failed—due to syntax errors or failure to produce expected outputs—a second pipeline was introduced, wherein the generated code, along with a Python traceback, was sent to GPT-4 for correction. We experimented with two LLMs: GPT-4 and GPT-4o. In addition, an ensemble approach was tested, where the final output was determined by combining the outputs from multiple models and selecting the most frequent one.

4.3.2 Results and Discussion

The performance of each model on the public and private leaderboards of the ICML challenge is summarized in Table 3. The performance is measured based on the accuracy of the generated code in solving the optimization problems.

The results demonstrate that the RAGthoven-augmented pipeline, especially when paired with GPT-4o, performs robustly on both the public and private leaderboards. Interestingly, increasing the number of examples in the prompt (`k`) did not consistently lead to performance improvements, and the ensemble approach did not significantly outperform the individual models. This suggests that retrieval-based augmentation, combined with prompt-tuning, can provide strong baselines for code generation tasks. However, there may be diminishing returns when adding more examples or using model ensembles.

Model	Public	Private
`GPT-4 (k=2)`	0.840	**0.8171**
`GPT-4 (k=3)`	0.815	0.7862
`GPT-4 (k=4)`	0.835	0.7933
`GPT-4o (k=3)`	**0.850**	0.8147
`GPT-4o (k=4)`	0.820	0.8052
`GPT-4o (k=5)`	0.845	0.8147
`Ensemble`	**0.850**	**0.8171**

Table 3: Results for code generation models on the public and private leaderboards of the ICML 2024 Automated Math Reasoning Challenge. `k` denotes the number of examples in the prompt. Bold values represent the highest performance for each leaderboard.

The full RAGthoven configuration for the outlined experiments can be found in Appendix A and seen in Figure 6.

5 Conclusion

In this paper, we introduce RAGthoven, a toolkit designed to simplify the evaluation of RAG-based pipelines. By offering a configuration-driven interface, RAGthoven makes retrieval-augmented models accessible to both technical users and non-programmers. Its modular nature allows for easy extension, ensuring future adaptability across diverse NLP tasks. Through case studies in text classification, question answering, and code generation, we demonstrate that RAGthoven maintains competitive performance while streamlining setup. Our results highlight its potential to lower the barriers for wider adoption of RAG across academia and industry. To this end we release RAGthoven under the terms of the MIT license, hoping it will contribute increased RAG usage in the future.

References

Tassallah Abdullahi, Ritambhara Singh, and Carsten Eickhoff. 2024. Retrieval augmented zero-shot text classification. In *Proceedings of the 2024 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 195–203.

Nastaran Bassamzadeh and Chhaya Methani. 2024. A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation. *arXiv preprint arXiv:2407.02742*.

Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. 2024. Lift yourself up: Retrieval-augmented text generation with self-memory. *Advances in Neural Information Processing Systems*, 36.

Prithiviraj Damodaran. 2023. FlashRank, Lightest and Fastest 2nd Stage Reranker for search pipelines.

Daryna Dementieva, Daniil Moskovskiy, Nikolay Babakov, Abinew Ali Ayele, Naquee Rizwan, Frolian Schneider, Xintog Wang, Seid Muhie Yimam, Dmitry Ustalov, Elisei Stakovskii, Alisa Smirnova, Ashraf Elnagar, Animesh Mukherjee, and Alexander Panchenko. 2024. Overview of the multilingual text detoxification task at pan 2024. In *Working Notes of CLEF 2024 - Conference and Labs of the Evaluation Forum*. CEUR-WS.org.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Michael Glass, Xueqing Wu, Ankita Rajaram Naik, Gaetano Rossiello, and Alfio Gliozzo. 2023. Retrieval-based transformer for table augmentation. *arXiv preprint arXiv:2306.11843*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Jie Huang, Wei Ping, Peng Xu, Mohammad Shoeybi, Kevin Chen-Chuan Chang, and Bryan Catanzaro. 2023. Raven: In-context learning with retrieval augmented encoder-decoder language models. *arXiv preprint arXiv:2308.07922*.

Peter Izsak, Moshe Berchansky, Daniel Fleischer, and Ronen Laperdon. 2023. fastRAG: Efficient Retrieval Augmentation and Generation Framework.

Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *Preprint*, arXiv:2405.13576.

Brendan King and Jeffrey Flanigan. 2023. Diverse retrieval-augmented in-context learning for dialogue state tracking. *arXiv preprint arXiv:2307.01453*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Quentin Lhoest, Albert Villanova Del Moral, Yacine Jernite, Abhishek Thakur, Patrick Von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. 2021. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*.

Xiaonan Li, Kai Lv, Hang Yan, Tianyang Lin, Wei Zhu, Yuan Ni, Guotong Xie, Xiaoling Wang, and Xipeng Qiu. 2023. Unified demonstration retriever for in-context learning. *arXiv preprint arXiv:2305.04320*.

Lefteris Loukas, Ilias Stogiannidis, Odysseas Diamantopoulos, Prodromos Malakasiotis, and Stavros Vassos. 2023. Making llms worth every penny: Resource-limited text classification in banking. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 392–400.

Malte Pietsch, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann, Tanay Soni, and Sebastian Lee. 2019. Haystack: the end-to-end NLP framework for pragmatic builders.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Erik Řehulka and Marek Šuppa. 2024. Rag meets detox: Enhancing text detoxification using open large language models with retrieval augmented generation.

Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (rag) models for open domain question answering. *Transactions of the Association for Computational Linguistics*, 11:1–17.

Marek Suppa, Daniel Skala, Daniela Jass, Samuel Sucik, Andrej Svec, and Peter Hraska. 2024. Bryndza at climateactivism 2024: Stance, target and hate event detection via retrieval-augmented gpt-4 and llama. *ArXiv*, abs/2402.06549.

TextDetox. 2023. Multilingual paradetox. https://huggingface.co/datasets/textdetox/multilingual_paradetox. Accessed: 2024-05-30.

Surendrabikram Thapa, Kritesh Rauniyar, Farhan Ahmad Jafri, Shuvam Shiwakoti, Hariram Veeramani, Raghav Jain, Guneet Singh Kohli, Ali Hürriyetoğlu, and Usman Naseem. 2024. Stance and hate event detection in tweets related to climate activism-shared task at case 2024. In *7th Workshop on Challenges and Applications of Automated Extraction of Socio-Political Events from Text, CASE 2024*, pages 234–247. Association for Computational Linguistics.

Shuohang Wang, Yichong Xu, Yuwei Fang, Yang Liu, Siqi Sun, Ruochen Xu, Chenguang Zhu, and Michael Zeng. 2022. Training data is more valuable than you think: A simple and effective method by retrieving from training data. *arXiv preprint arXiv:2203.08773*.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Improving named entity recognition by external context retrieving and cooperative learning. *arXiv preprint arXiv:2105.03654*.

A RAGthoven Configurations

```yaml
training_data:
  dataset: SubTaskA-train.csv
  input_feature: "text"
  output_feature: "prediction"
validation_data:
  dataset: SubTaskA-test.csv
  input_feature: "text"
embed:
  k: [6, 12]
  model: [
    "sentence-transformers/all-MiniLM-L6-v2",
    "sentence-transformers/all-mpnet-base-v1"
  ]
rerank:
  k: 6
llm:
  model: "gpt4"
  sprompt: |
    Analyze the input tweet to determine if it
    is hate speech or not, based on the
    following criteria:

    ## Hate Speech Patterns

    ...

    ## Non-Hate Speech Patterns

    ...

    ## Evaluation

    - If the tweet aligns more with the
    Hate Speech Patterns, output:
    'Prediction: 1' (indicating it is
    hate speech).

    - If the tweet aligns more with the
    Non-Hate Speech Patterns, output:
    'Prediction: 0' (indicating it is
    not hate speech).

    ## Examples
    {{ examples }}
  uprompt: |
    {{ text }}
```

Figure 4: The RAGthoven configuration file associated with the The Hate Speech Detection subtask from the CASE 2024 Shared Task on Climate Activism Stance and Hate Event Detection. The ellipsis (...) in both Hate and Non-Hate Speech patterns denotes parts of the prompt that has been omitted for brevity.

```yaml
training_data:
  dataset: "textdetox/multilingual_paradetox"
  input_feature: "toxic_sentence"
  output_feature: "neutral_sentence"
validation_data:
  dataset: "textdetox/multilingual_paradetox_test"
  input_feature: "toxic_sentence"
embed:
  k: 10
llm:
  model: "ollama/llama3"
  sprompt: |
    ## Task

    You are a text detoxifier. On input you receive
    a text which may be toxic or harmful. Your task
    is to rewrite this text in a way that does not
    contain any toxicity or harmful words, while
    preserving the original content and context.

    The Output contains only the detoxified text
    and nothing else like notes or additional
    information. You do not add any more context
    to the resulting text, which is not in the
    original text.
    Do not rewrite the original text too much,
    just either remove the toxic part completely,
    or replace it with some non-toxic words while
    preserving the meaning and context.

    The language of the input is {{ language }}
    and the language of the response must be the
    same.

    ## Examples
    {{ examples }}
  uprompt: |
    {{ text }}
```

Figure 5: The RAGthoven configuration file associated with the Text Detoxification 2024 Shared Task.

```yaml
training_data:
  dataset: optim-with-code-train.json
  input_feature: "task_definition"
  output_feature: "code"
validation_data:
  dataset: optim-with-code-val.json
  input_feature: "task_definition"
embed:
  k: [2, 3, 4, 5]
llm:
  model: ["gpt-4", "gpt-4o"]
  sprompt: |
    You are an Optimization Problem-Solving
    expert that solves optimization tasks using
    the PuLP library in Python.

    # Task
    You will be given a description of an
    optimization problem on the input and will be
    asked to output a Python script that uses the
    PuLP library that solves the task.

    The Python code should compute ALL of the
    expected variables and print their output out.
    Use the EXACT SAME variable names in the
    output VERBATIM as the ones provided in the
    input.

    Do not add any additional formatting to the
    output (e.g. no '$' signs, no commas, etc.).

    Format the output as float values.

    Output ONLY the resulting Python code and
    nothing else.

    Follow the format in the examples below.
    Do not attempt to compute anything yourself,
    only output Python code that does the
    computation.

    # Examples
    {{examples}}
  uprompt: |
    {{ text }}
```

Figure 6: The RAGthoven configuration file associated with the ICML 2024 Automated Math Reasoning Challenge.