Multi-task retriever fine-tuning for domain-specific and efficient RAG

Patrice Bechard
ServiceNow
Montreal, Canada
patrice.bechard@servicenow.com

Orlando Marquez Ayala
ServiceNow
Montreal, Canada
orlando.marquez@servicenow.com

Abstract

Retrieval-Augmented Generation (RAG) has become ubiquitous when deploying Large Language Models (LLMs), as it can address typical limitations such as generating hallucinated or outdated information. However, when building real-world RAG applications, practical issues arise. First, the retrieved information is generally domain-specific. Since it is computationally expensive to fine-tune LLMs, it is more feasible to fine-tune the retriever model to improve the quality of the data included in the LLM input. Second, as more applications are deployed in the same real-world system, one cannot afford to deploy separate retrievers for different tasks. Moreover, these RAG applications normally retrieve different kinds of data. Our solution is to instruction fine-tune a small retriever model on a variety of domain-specific tasks to allow us to deploy one model that can serve many use cases, thereby achieving low-cost, scalability, and speed. We show how this model generalizes to out-of-domain settings as well as to an unseen retrieval task on real-world enterprise use cases.

CCS Concepts

• **Computing methodologies** → **Information extraction**; **Multi-task learning**; *Learning to rank*; • **Information systems** → **Retrieval models and ranking**.

Keywords

Retrieval-Augmented Generation, Workflows, Fine-Tuning, Multi-Task, Retriever

1 Introduction

As more and more Generative AI (GenAI) applications are integrated into real-world production systems, Retrieval-Augmented Generation (RAG) has been adopted in industry as a common technique to improve the output of Large Language Models (LLMs). RAG alleviates inherent LLM pitfalls such as propensity to hallucinate, generating outdated knowledge, and lack of traceability to data sources [6, 9].

Introducing a retrieval step into the generation process introduces, however, several practical challenges. While an LLM with a large number of parameters, such as GPT-4 [19], can be prompted to work with any kind of input and generate any kind of textual output, the retriever needs to be small, fast, and perform well with data sources that tend to be domain-specific.

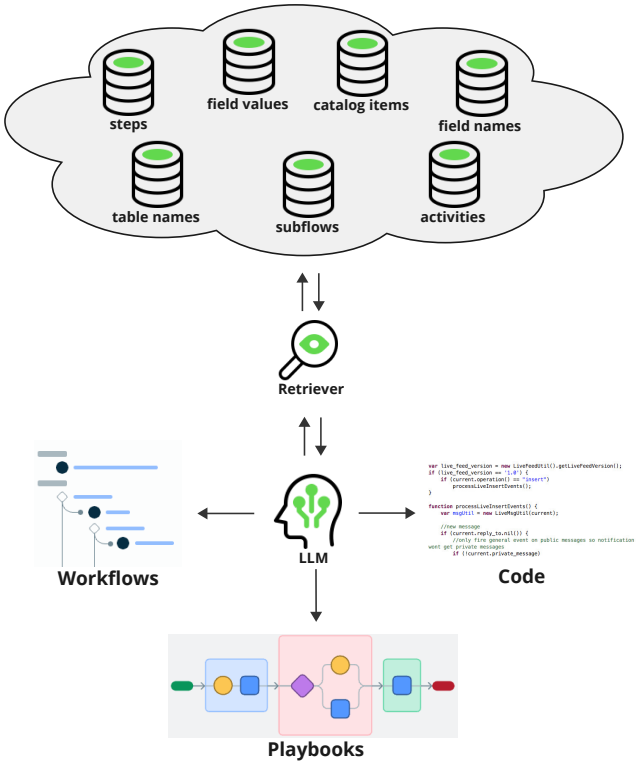

Figure 1: Given an ecosystem of RAG applications, how do we build a retriever that can adapt to a specific domain and to a variety of retrieval tasks?

Off-the-shelf retrievers of different sizes are available to AI practitioners. Embedding services such as Voyage[1] perform well on open-source benchmarks but they do not necessarily generalize to the kind of data seen in real-world settings, especially when this data is structured and comes from existing databases.

Another practical challenge is achieving scalability and generalization across different GenAI use cases that depend on retrieval. A crucial advantage of LLMs compared to traditional machine learning models is that they can generalize to a myriad of tasks due to vast amounts of pretraining data and instruction fine-tuning [20, 31, 34]. But if the retriever does not perform well and fast across many retrieval tasks, the downstream generation will be negatively affected.

The problem we are trying to solve is then: *how to adapt the retrieval step to a specific domain and to a variety of retrieval tasks?* In this work, we are not interested in the choice of LLM,

KDD Workshop '25, Toronto, Canada
2025.

[1] https://docs.voyageai.com/docs/embeddings

assuming that improvements in the retrieved results translate into improvements in the downstream generation task.

The context is an enterprise company that deploys several GenAI applications that currently rely or will rely on RAG: Flow Generation [3], Playbook Generation[2], and Code Generation. Workflows are step-by-step processes that automate one or more goals while playbooks contain workflows and other UI components such as forms. The objective of these applications is to generate domain-specific workflows, playbooks, and code from textual input.

Our solution is to instruction fine-tune a small retriever on a variety of tasks. This retriever is deployed in the ecosystem of GenAI applications, which prompt it to retrieve desired structured data from databases. Information such as workflow step names, table names, and field names are then passed to LLMs to generate workflows or playbooks, leading to higher output quality.

To build a multi-task retriever dataset, we extracted data from internal databases and reused the Flow Generation training set. For our solution, we fine-tune mGTE [36] because of its large context length (8,192 tokens), allowing it to receive long instructions, and because of its multilingual capabilities. We compare it with BM25, a simple yet powerful term-frequency method [23], and with recent open-source multilingual embedding models: mE5 [30] and mGTE. We also include evaluation of larger instruction-tuned models based on LLMs, namely GTE-Qwen-2 [32], E5-mistral [29], and GritLM [17].

We perform several evaluations. First, we evaluate on the tasks that the retriever was trained on, but on out-of-domain (OOD) settings. The internal training datasets come from the IT domain, but the OOD splits come from diverse domains such as HR and finance. We then evaluate on a related but different retrieval task to test the generalization ability of the model. For example, while the model was trained to retrieve step names based on text, we also want to retrieve relevant workflow structures based on text. This is a related task because workflows are made up of steps. Lastly, to see whether the multilingual abilities of models are preserved after our multi-task fine-tuning, we evaluate on input from different languages even though our retrieval training dataset is only in English.

Our contributions are the following:

- We provide a case study for how to build a domain-specific and efficient retriever for real-world RAG.
- We demonstrate that multi-task retriever fine-tuning can lead to generalization to out-of-domain datasets and to related but different retrieval tasks.

2 Related Work

There have been several efforts to achieve **domain-specific RAG**. While more complex, jointly training the retriever and the generator has been shown to work well for question-answering [25]. Others encode the domain-specific information in knowledge graphs in addition to vector databases [2]. Retrieval-Augmented Fine Tuning [35] helps the LLM adapt to the domain by modifying how the retrieved information is present in the LLM training dataset. In contrast, our approach relies solely on training a better retriever, which can be used with any LLM, thereby reducing coupling.

Multi-task retriever models [15, 28] enhance embedding versatility by simultaneously training on multiple tasks or by utilizing a large dataset spanning diverse topics. Building upon this concept, **instruction-following embedding models** [1, 4, 26] integrate explicit instructions into the embedding process, thereby enabling the generation of more nuanced and task-specific embeddings. We leverage this kind of instruction fine-tuning to train a retriever on many datasets containing structured data extracted from databases.

Lastly, generating workflows is a structured output task, similar to code generation, that requires specialized embeddings. **Code embedding models** [7, 10, 13, 18, 33] adapt common training techniques to the domain of code representation learning and retrieval. A parallel research trajectory focuses on **structured data embedding models**, including Synchromesh [21], which fine-tunes a model to retrieve relevant samples for few-shot prompting using a similarity metric derived from tree edit distance, and SANTA [14], which implements a modified pretraining objective to generate structure-aware representations. Prior work already uses RAG for a similar but more limited structured output task [3]. This paper is a natural extension to increase reusability of the retriever in an expanding ecosystem of GenAI applications.

3 Methodology

To build a multi-task retriever, we first had to define the tasks and then build their datasets, keeping in mind that we could not do any labeling for these tasks. The starting point was to define the data that the existing RAG applications would need.

The GenAI applications currently deployed in our system require diverse types of retrieved data to generate output acceptable to users. As these applications become more sophisticated, the extent and type of data retrieved can only increase. But currently we focus mostly on *steps*, which are used by workflows and playbooks, and *tables*, which are used by workflows and playbooks, and can be used in code generation. Steps are building blocks of processes; they can be *actions*, *subflows*, or *activities*. When it comes to tables, the crucial information are *table names* and *table field names*. For instance, a user may want to generate code that makes a database call, but without grounding the LLM on actual table information, the LLM may refer to a non-existent table name.

We extracted data from two sources to create the multi-task retrieval dataset. Complex and semantically diverse examples were created from the Flow Generation training set. Figure 2 shows an example from this set in YAML format, which includes step names (`definition`), table names, and table field names (part of `conditions` and `values`). The other source are database tables that include other elements we are interested in, besides steps and tables, that contain a text field such as *description*.

3.1 Tasks

From the Flow generation training set, we constructed three categories of tasks, constituting around half of all retrieval examples:

(1) Retrieve **steps** that can be used in workflows or playbooks.
(2) Retrieve **tables** that can be used in step inputs or in code.

[2]www.servicenow.com/docs/bundle/xanadu-intelligent-experiences/page/administer/now-assist-platform/concept/now-assist-playbook-generation-skill.html

```
requirement: Every day, look up incident tasks that do not have
assignees and close them.
trigger:
  annotation: every day
  type: daily
  inputs:
  - name: time
    value: '1970-01-02 00:00:00'
steps:
- annotation: look up incident tasks that do not have assignees
  definition: look_up_records
  order: 1
  inputs:
  - name: table
    value: incident_task
  - name: conditions
    value: assigned_toISEMPTY
- annotation: for each record
  definition: FOREACH
  order: 2
  inputs:
  - name: items
    value: '{{1.Records}}'
- annotation: close them
  definition: update_record
  order: 3
  block: 2
  inputs:
  - name: record
    value: '{{2.item}}'
  - name: table_name
    value: incident_task
  - name: values
    value: state=3
```

Figure 2: Dataset example of *Flow Generation*. We constructed retrieval multi-task examples from them.

(3) Retrieve **table fields** that can be used in step inputs or in code, given a table name.

To add diversity to each of the tasks, we included several permutations of the input, such as:

- Retrieve all the steps used in the workflow given the requirement. In the example in Figure 2, this means to retrieve `look_up_records`, `FOREACH`, and `update_record` given the requirement *Every day, look up incident tasks that do not have assignees and close them.*
- Retrieve a specific step from an annotation. Given the example in Figure 2, we can ask a retriever to get the step `look_up_records` given the text *Look up incident tasks that do not have assignees*.
- Retrieve a table name given a workflow context. We would want the table name *incident_task* given all the previous steps (all YAML lines up to `definition: update_record` in Figure 2).
- Retrieve a table field name from text. Given the annotation *Look up incident tasks that do not have assignees*, retrieve `assigned_to`.

The remaining half of retrieval examples come from database tables, such as *catalog items*. This table represents items such as laptops that can be requested in the system. There is a field *description* along the catalog item name, yielding a task where the item name is retrieved given a description.

3.2 Dataset Generation

Once we defined the tasks to train the retriever, we proceeded to create a set of pairs consisting of text and objects, which can be steps, tables, table field names, catalog items, etc. The text portion of the pair is an instruction describing what has to be retrieved. Below we instruct the retriever to find steps given only a requirement.

```
Represent this requirement for searching relevant steps:
requirement: Every day, look up incident tasks that do
     not have assignees and close them.
```

Given that there may be more metadata available to find steps, we can add extra information to the instruction. Below we tell the retriever that this flow is part of the `teams` scope.

```
Represent this flow for searching relevant steps:
type: flow
scope: sn_ms_teams_ah
requirement: Every day at 9am, notify me of pending tasks
      in a teams msg.
```

Because we instruction fine-tune the retriever, we can even add more details. For instance, the instruction can contain the workflow context. This is useful when we want the retriever to be influenced by what is already included in the workflow in addition to the last annotation.

```
Represent this flow for searching relevant steps for the
     last step:
type: flow
rcope: sn_ms_teams_ah
requirement: Every day, look up incident tasks that do
     not have assignees and close them.
trigger:
  annotation: every day
  type: daily
  inputs:
  - name: time
  - value: '1970-01-02 00:00:00'
steps:
- annotation: look up incident tasks that do not have
     assignees
```

We can use similar instructions to retrieve any element. Given annotated workflows and extracts from database tables, we can therefore create a large number of samples for text/step, text/table, text/-field, text/catalog item, etc. In total, we use 15 instruction templates to add variety to the input, and we plan to add more.

Following established practices in sentence embedding model training [22], we create both positive and negative pairs, enabling the model to effectively discriminate and retrieve relevant elements.

Positive samples are extracted from the labeled workflows as described above. To mine negative samples, we use two sampling strategies:

- **Random negative** sampling simply takes an unrelated step, table, or field and matches it to a piece of text used in another positive sample.
- **Hard negative** sampling uses the *structure* of the data to find harder negative examples. For example, as steps tend to be grouped in *scopes*, we sample another step coming from the same scope as the positive step. Both `post_response_to_slack` and `post_a_message` are in the `slack` scope, but they are used in different scenarios.

4 Experiments

We describe the datasets used in training and evaluation, the baselines we compare against, as well as the retrieval metrics we used.

4.1 Datasets

For training, we use all pairs we could extract from the Flow generation training set and dataset tables, but for our development set, we evaluate only on step, table name, and table field retrieval, as these are the most important retrieval tasks. The development examples come only from the Flow Generation development set. Table 1 shows the number of examples for steps, tables, and fields in the multi-task development set.

Table 1: Number of training pairs and examples in the development dataset.

Train Pairs	Dev Steps	Dev Tables	Dev Fields
172,658	279	307	355

Although there are close to 5K unique steps in the datasets, there are around 20 that are used most frequently. This creates a large imbalance in the step retrieval task. For example, the `look_up_record` step is used much more often than a step in the `slack` scope, since retrieving database records is a frequent operation in workflows. We then experimented with downsampling very frequent components based on their frequency in an exponential fashion (e.g., downsample steps occurring 50 times in our dataset by 4x and downsample steps occuring 500 times by 16x). A similar procedure is applied to tables and fields although their imbalance is less drastic.

For evaluation, we use 10 splits from deployments of the enterprise system, thereby simulating evaluating on real-world customer settings. While the labeled dataset is from our internal IT domain, these other splits come from different domains and include steps and tables not present in the IT domain. Their statistics are shown in Table 2.

Table 2: Number of retrieval examples in OOD splits.

Dataset ID	Flows	Steps samples	Tables samples	Fields samples
OOD1	103	166	58	107
OOD2	100	94	209	392
OOD3	100	119	261	450
OOD4	100	136	188	275
OOD5	100	141	206	305
OOD6	100	179	255	529
OOD7	98	140	329	665
OOD8	100	145	188	246
OOD9	100	207	158	301
OOD10	171	134	477	690
TOTAL	**1,072**	**1,461**	**2,329**	**3,960**

To test whether the multilingual abilities of the base model (mGTE) are preserved after our multi-task fine-tuning, we translate the development dataset using the Google Translate API.

Lastly, our aim is to generalize to similar retrieval tasks, allowing the retriever to be used in *any* RAG application that needs similar kind of data. To this end, we create a task called *workflow retrieval*, consisting of retrieving workflows (in YAML format) from text. This could be useful for few-shot prompting in Flow Generation. For this task, we evaluate on flows from OOD1, OOD8, and OOD9, resulting in 303 examples.

4.2 Models

The simplest baseline we compare against is BM25, the simplest in terms of RAG deployment. Stronger baselines are open-source multilingual models. We select the small (118M), base (278M), and large (560M) versions of mE5 [30] to see whether more parameters help retrieval. Since mE5 has a context length of 512, we also pick mGTE-base [36], which has 8,192 context length and 305M parameters.

Our solution is to fine-tune mGTE-base using a contrastive loss objective [11]. We prefer a retriever with large context as, once deployed, we do not control the length of instructions/context that it will receive. We also prefer a small model to allow the retrieval part of RAG to scale to many LLM concurrent calls. Appendix A provides details of our training process, including hyperparameters.

4.3 Evaluation Metrics

We use Recall@K [16] as our metric across all tasks. While we briefly explored more sophisticated metrics such as Normalized Discounted Cumulative Gain (NDCG) [12] and Mean Reciprocal Rank (MRR) [27], we found that these metrics highly correlate with recall in our use case.

We report results on the most important retrieval tasks. When generating a workflow or a playbook with many steps, the LLM needs to be grounded on the customer installation of the enterprise system. In these cases, as we suggest to show many steps to the LLM, we evaluate Step@15 (K=15). When the LLM needs to include a table or field name in its generation, we suggest fewer choices, as there is typically one or two tables that are needed; hence, here we use Table@5 and Field@5 (K=5).

5 Results

We first show how we arrived at our best multi-task retriever using the development set, comparing it to models fine-tuned on each task separately. We then show the OOD, multilingual, and *workflow retrieval* results.

5.1 Multi-task Retriever

We expect multi-task fine-tuning to be better than single task fine-tuning due to a larger dataset of related tasks. However, we did not expect the imbalance in the steps distribution to cause such degradation. Table 3 shows that downsampling the data causes an improvement of 8% in step retrieval with a small loss in field retrieval, on a fine-tuned mGTE-base model.

These findings suggest that when fine-tuning a retriever for RAG, one needs to be careful on the dataset make-up. In real-world settings, there typically is a large data imbalance that needs to be handled according to the domain.

Table 3: Single task vs Multi-task and effect of balanced dataset. Evaluation on development split.

Setup	Step@15	Table@5	Field@5
Single Task	0.78	0.82	0.71
Multi-Task	0.77	0.86	**0.73**
+ Downsampled data	**0.86**	**0.88**	0.71

5.2 Comparison to Baselines on OOD Splits

Table 4 shows average results on all OOD splits, weighted by dataset size.

Table 4: Performance of BM25, small and large open source models, and multi-task instruction fine-tuned mGTE-base (ours). Results are weighted averages across all OOD splits.

Model	Step@15	Table@5	Field@5
BM25	0.82	0.79	0.26
mE5-Small	0.72	0.54	0.15
mE5-Base	0.74	0.64	0.15
mE5-Large	0.72	0.59	0.13
mE5-Large-Instruct	0.80	0.74	0.14
mGTE-Base	0.72	0.63	0.08
GTE-Qwen2-1.5B-Instruct	0.21	0.12	0.01
GTE-Qwen2-7B-Instruct	0.18	0.08	0.03
GritLM-7B	0.75	0.59	0.16
E5-Mistral-7B-Instruct	0.57	0.51	0.16
Finetuned mGTE-Base (Ours)	**0.90**	**0.90**	**0.60**

On our domain-specific retrieval benchmarks, our multi-task fine-tuned mGTE-base model achieves the highest performance across all metrics, substantially outperforming both BM25 and all evaluated open-source embedding models. Notably, BM25 consistently outperforms the open-source embedding models — regardless of model size — including larger variants such as mE5-large and GTE-Qwen2. Increasing the scale of the mE5 retrievers from small to large yields no improvement, and in some cases results in lower scores. Only our model delivers strong, consistent gains, reaching 0.90 on both Step@15 and Table@5, and 0.60 on Field@5.

We also see that field retrieval is the most difficult task given that there is a greater variety of field names compared to tables and steps. The retriever seems to generalize well across domains as the recall numbers do not vary much from the development set (shown in Table 3).

5.3 Effect on Multilingual Capabilities

We picked mGTE as our base model not only for its long context length but also because it supports multiple languages. Many open source and commercial LLMs already support multiple languages. To make RAG work properly, the retriever also needs to work well in non-English languages.

We translated the development dataset into German (DE), Spanish (ES), French (FR), Japanese (JA), and Hebrew (HE) using the Google Translate API. Table 5 shows the results comparing mGTE-base and our fine-tuned version.

Our multi-task fine-tuning on English datasets allows the retriever to work better than the base model on these non-English datasets.

Table 5: Results on multilingual development datasets across step, table, and field retrieval.

Model	DE	ES	FR	JA	HE	Avg.
Step@15						
mGTE-base	0.53	0.66	0.66	0.60	**0.47**	0.58
Ours	**0.65**	**0.75**	**0.76**	**0.64**	0.42	**0.64**
Table@5						
mGTE-base	0.34	0.38	0.42	0.36	0.39	0.38
Ours	**0.65**	**0.76**	**0.71**	**0.62**	**0.57**	**0.66**
Field@5						
mGTE-base	0.15	0.17	0.16	0.15	0.17	0.16
Ours	**0.49**	**0.58**	**0.57**	**0.41**	**0.36**	**0.48**

The only instance where the base model performs better is with Hebrew step retrieval. However, the average results across all three retrieval tasks are significantly lower than the English results shown in Table 4: step retrieval goes down from 0.90 to 0.64, table retrieval from 0.90 to 0.66, and field retrieval from 0.60 to 0.48. This suggests that we may need to add some multilingual domain-specific data to the multi-task dataset.

5.4 Generalization to *Workflow Retrieval*

Once our retriever is deployed, it can be used by any other RAG application. We therefore evaluate it on a similar task that entails retrieving workflows given text. We use recall@5 where we want to find a single workflow for a specific text. We picked three OOD splits to reduce evaluation time.

Table 6 shows that the multi-task fine-tuned retriever performs better than both BM25 and the base model. But this may be too easy a task as the base model obtains 0.87 recall@5 on average. Nevertheless, our fine-tuning improved performance, showing that the multi-task dataset can transfer knowledge to similar retrieval tasks.

Table 6: Generalization on *workflow retrieval* task. Metric is recall@5.

Model	OOD1	OOD8	OOD9	Avg.
BM25	0.73	0.60	0.66	0.66
mGTE-base	0.94	0.76	**0.90**	0.87
Ours	**0.98**	**0.93**	**0.90**	**0.94**

6 Conclusion

We present an approach to build a small retriever for domain-specific RAG. Via multi-task training and instruction fine-tuning, we can deploy a retriever of only 305M parameters for many applications, so that hardware costs and latency are minimized. Moreover, this retriever performs well on multilingual datasets in domain-specific use cases and shows promising results generalizing to similar retrieval tasks. Future work includes expanding the retrieval tasks and improving the multilingual capabilities of the fine-tuned model.

References

[1] Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. 2023. Task-aware Retrieval with Instructions. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 3650–3675. doi:10.18653/v1/2023.findings-acl.225

[2] Ryan C. Barron, Ves Grantcharov, Selma Wanna, Maksim E. Eren, Manish Bhattarai, Nicholas Solovyev, George Tompkins, Charles Nicholas, Kim Ø. Rasmussen, Cynthia Matuszek, and Boian S. Alexandrov. 2024. Domain-Specific Retrieval-Augmented Generation Using Vector Stores, Knowledge Graphs, and Tensor Factorization. arXiv:2410.02721 [cs.CL] https://arxiv.org/abs/2410.02721

[3] Patrice Bechard and Orlando Ayala. 2024. Reducing hallucination in structured outputs via Retrieval-Augmented Generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, Yi Yang, Aida Davani, Avi Sil, and Anoop Kumar (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 228–238. doi:10.18653/v1/2024.naacl-industry.19

[4] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961* (2024).

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[6] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6491–6501. doi:10.1145/3637528.3671470

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling deep contrastive learning batch size under memory limited setup. *arXiv preprint arXiv:2101.06983* (2021).

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[10] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[11] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, Vol. 2. IEEE, 1735–1742.

[12] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[13] Xiaonan Li, Yeyun Gong, Yelong Shen, Xipeng Qiu, Hang Zhang, Bolun Yao, Weizhen Qi, Daxin Jiang, Weizhu Chen, and Nan Duan. 2022. Coderetriever: A large scale contrastive pre-training method for code search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2898–2910.

[14] Xinze Li, Zhenghao Liu, Chenyan Xiong, Shi Yu, Yu Gu, and Zhiyuan Liu. 2023. Ge Yu. Structure-aware language model pretraining improves dense retrieval on structured data. *arXiv preprint arXiv:2305.19912* (2023).

[15] Jean Maillard, Vladimir Karpukhin, Fabio Petroni, Wen-tau Yih, Barlas Oğuz, Veselin Stoyanov, and Gargi Ghosh. 2021. Multi-task retrieval for knowledge-intensive tasks. *arXiv preprint arXiv:2101.00117* (2021).

[16] Christopher D Manning. 2008. *Introduction to information retrieval*. Syngress Publishing,.

[17] Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2024. Generative Representational Instruction Tuning. arXiv:2402.09906 [cs.CL]

[18] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005* (2022).

[19] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[20] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 27730–27744. https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

[21] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227* (2022).

[22] 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)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. doi:10.18653/v1/D19-1410

[23] S. E. Robertson and S. Walker. 1994. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Dublin, Ireland) *(SIGIR '94)*. Springer-Verlag, Berlin, Heidelberg, 232–241.

[24] Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*. PMLR, 4596–4604.

[25] 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 (2023), 1–17. doi:10.1162/tacl_a_00530

[26] Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A Smith, Luke Zettlemoyer, and Tao Yu. 2022. One embedder, any task: Instruction-finetuned text embeddings. *arXiv preprint arXiv:2212.09741* (2022).

[27] Ellen M Voorhees, Dawn M Tice, et al. 1999. The TREC-8 Question Answering Track Evaluation.. In *TREC*, Vol. 1999. 82.

[28] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533* (2022).

[29] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving Text Embeddings with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 11897–11916. doi:10.18653/v1/2024.acl-long.642

[30] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual E5 Text Embeddings: A Technical Report. arXiv:2402.05672 [cs.CL] https://arxiv.org/abs/2402.05672

[31] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*. https://openreview.net/forum?id=gEZrGCozdqR

[32] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115* (2024).

[33] Dejiao Zhang*, Wasi Ahmad*, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. 2024. CodeSage: Code Representation Learning At Scale. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=vfzRRjumpX

[34] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2024. Instruction Tuning for Large Language Models: A Survey. arXiv:2308.10792 [cs.CL] https://arxiv.org/abs/2308.10792

[35] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E. Gonzalez. 2024. RAFT: Adapting Language Model to Domain Specific RAG. In *First Conference on Language Modeling*. https://openreview.net/forum?id=rzQGHXNReU

[36] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina (Eds.). Association for Computational Linguistics, Miami, Florida, US, 1393–1412. doi:10.18653/v1/2024.emnlp-industry.103

A Training Details

We fine-tuned mGTE-base for 5,000 steps, equivalent to approximately 2 epochs on the dataset. Training hyperparameters were as follows: a batch size of 32 was employed, with a per-device batch size of 2 and 16 gradient accumulation steps. Since we are not using in-batch negatives, we do not need to employ techniques such as GradCache [8] to account for larger batch sizes. The learning rate was set to 5e-5, with a weight decay of 0.01. A warmup period of 500 steps was implemented, followed by a cosine learning rate scheduler. To optimize memory usage, gradient checkpointing was utilized [5]. The Adafactor optimizer [24] was chosen for limited memory consumption.

Received 30 May 2025