SAGELITE: HARMONIZING TEXT AND CODE EMBEDDING VIA TWO-STAGE TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Creating versatile embedding models that excel across both text and code domains is essential, as modern applications often involve diverse, heterogeneous data. While data mixing is a typical starting point, we take a significant step forward by addressing the limitations of naive data mixing. In this work, we introduce SAGELITE, a unified embedding model capable of handling both text and code within a single framework. Our approach begins with pretraining on a blended dataset of text and code, fostering shared representations that are crucial for strong cross-domain performance. We then enhance domain-specific capabilities by independently applying large-scale contrastive learning to text and code from various web sources. Our key finding is that, despite the inherent differences between text and code, starting from a model pretrained on mixed data enables the domain-specific contrastive learning stages to produce models that remain closely aligned. This alignment allows us to effectively integrate domain-specific improvements at the constrastive learning stage into a final model through model weights interpolation. Through comprehensive ablation studies, we explore the mechanisms behind our approach, offering insights to guide future research in this area.

1 INTRODUCTION

Pre-training on large-scale heterogeneous data collected from various domains is crucial for enabling a single model to demonstrate remarkable versatility in generation tasks, including both code and text generation (OpenAI, 2023; Team et al., 2024; Dubey et al., 2024, *inter alia*). However, state-of-the-art (SOTA) embedding models are often domain-specific, typically focusing on either text (Li & Li, 2023; Wang et al., 2022; 2024; Lee et al., 2024, *inter alia*) or code (Zhang et al., 2024; Guo et al., 2022, *inter alia*) independently. There is a strong practical need for versatile embedding models that can handle heterogeneous data, as they can significantly reduce hosting costs and eliminate the need for complex routing system designs in applications requiring multi-domain processing.

To train a versatile model, data mixing is a natural starting point. However, the optimal data-mix strategy is not yet understood. The distributions of text and code are fairly different, leading many general-purpose generative models to incorporate only a small fraction of code data during pretraining (Touvron et al., 2023; Chowdhery et al., 2023, *inter alia*). Although studies by Hernandez et al. (2021); Muennighoff et al. (2024a) have empirically demonstrated positive transfer between text and code, these investigations are limited to Python, a language that intuitively shares more similarities with natural language. In our experiments, we indeed find that models trained with mixed data underperformed compared to domain-specific models on in-domain tasks at both the initial pretraining and the subsequent contrastive learning stages.

We are thus faced with a conundrum: on one hand, we aim to train a versatile embedding model; on the other hand, data mixing results in suboptimal performance at each stage of training. This challenge prompts us to explore two critical questions: (1) *Is cross-domain data mixing the only viable approach for training versatile embeddings at each stage?* (2) *What benefits can be gained from training on mixed data, and if data mixing is necessary at certain stages, can we devise a strategy that leverages its strengths in later stages to compensate the performance drop it causes earlier?*

By working backwards, we developed a multi-stage strategy that leverages two key ingredients to reach the sweet spot: the transfer learning capabilities of contrastive learning and the interpolation of properly trained models. Our key insight is that the *shared embedding model obtained by pre-training on the mixed data* can be leverage to attain strong transfer performance across-domains via contrastive learning in the subsequent stages. Such shared base model largely narrows the performance gap between the in-domain and out-of-domain contrastive learning, which coupled with contrastive learning's ability to keep models close, provides a solid foundation for constructing a versatile embedding model through the merging of models trained with domain-specific contrastive learning. Additionally, the strong cross-domain transfer learning performance enhances the performance of the merged model in each domain, compensating for the initial performance drop during the pretraining stage with mixed data. As a result, this versatile embedding model achieves comparable or even superior performance to models trained specifically for individual domains. To better understand the factors contributing to successful versatile embedding learning, we thoroughly analyze the key components of our framework and present our findings as directions for future research.

2 Related Work

Text Embedding Model The early success of transformer-based text embeddings (Reimers & Gurevych, 2019; Gao et al., 2021; Ni et al., 2021; Zhang et al., 2021, *inter alia*) is largely driven by the use of human-labeled datasets (Bowman et al., 2015; Williams et al., 2017; Nguyen et al., 2016). However, the high cost of human annotations often results in small-scale datasets with limited diversity and complexity. Consequently, these models often struggle with complex tasks and broad generalization. To improve general-purpose text embeddings, Izacard et al. (2021); Neelakantan et al. (2022); Wang et al. (2022) leverage large-scale, naturally occurring text pairs curated from web data to provide diverse training signals, *e.g.,* title and passage, or question and answer pairs mined from various web sources. More recently, Wang et al. (2024) and its follow-up work (Meng et al., 2023; Muennighoff et al., 2024b; Lee et al., 2024)demonstrated that fine-tuning Mistral-7B (Jiang et al., 2023) with high-quality synthetic data generated by proprietary LLMs (OpenAI, 2023) can push the boundaries of encoder-based embedding models.

Code Embedding Model Early work by Feng et al. (2020); Kanade et al. (2020) primarily followed BERT's training paradigm (Devlin et al., 2019), optimizing the Masked Language Modeling objective alongside various auxiliary tasks on linearized code data. Another line of research focuses on exploiting the structural aspects of code to provide additional training signals, either by explicitly utilizing data-flow information (Guo et al., 2021) or abstract syntax trees (ASTs) (Wang et al., 2021a; Jiang et al., 2021), or by implicitly encoding code structures through deobfuscation (Wang et al., 2021b; anne Lachaux et al., 2021; Zhang et al., 2024). More recently, Wang et al. (2021a); Guo et al. (2022); Neelakantan et al. (2022); Zhang et al. (2024) have substantially improved the quality of sequence-level representations by employing contrastive learning on (text, code) pairs mined from GitHub data.

Model Weights Interpolation Model weights interpolation dates back to the practice of averaging points along the training trajectory of stochastic gradient descent (Ruppert, 1988; Polyak & Juditsky, 1992; Szegedy et al., 2016; Izmailov et al., 2018), which has been shown to lead to wider optima and improved generalization. More recent research has explored model weight interpolation during the finetuning stage. Wortsman et al. (2022a) demonstrate better generalization performance by averaging models that were fine-tuned independently on the same dataset with varying configurations, while Wortsman et al. (2022b); Ilharco et al. (2022); Matena & Raffel (2022) extend this idea by including the pretrained model in the averaging process to achieve broader generalization. In contrast, our work mainly focuses on systematically training one versatile model with steered performance in each single domain, where model weight interpolation is one step of our entire pipeline.

Transfer Learning Between Text and Code The impact of incorporating code data on text performance, and vice versa, remains underexplored. Hernandez et al. (2021) empirically show that training exclusively on text achieves similar performance gains, measured by cross-entropy loss on test data, as those obtained by training on other programming languages. Conversely, Muennighoff et al. (2024a) demonstrate that incorporating Python into pretraining can enhance text performance, particularly in data-constrained scenarios. However, these studies focus solely on Python, a language that intuitively shares more similarities with natural language, and only verify on the loss function. The distributions of text and code are fairly different, leading many general-purpose generative models to incorporate only a small fraction of code data during pretraining (Touvron et al., 2023; Chowdhery et al., 2023, *inter alia*). In our work, we empirically show that, for embedding models, naïve mixing of text and code data for both MLM-based pretraining and contrastive learning leads to suboptimal performance on downstream tasks. A strategic approach is therefore necessary to train a general-purpose cross-domain embedding model, which is the primary focus of this paper.

3 Method

We train SageLite through two primary stages: token-level pretraining on a mix of text and code data, followed by sequence-level domain-specific contrastive learning.

3.1 Pretraining for Shared Embeddings between Text and Code

We first train the model with the mask language modeling (MLM) Devlin et al. (2019) objective. Given an input sequence with N tokens, *i.e.,* $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N,]$, the MLM objective (Devlin et al., 2019) is formed as

$$\mathcal{L}_{\text{MLM}}(\mathbf{x}) = -\sum_{i\in\mathcal{M}} \log \mathbb{P}\left(\mathbf{x}_i | \mathbf{x}^{\mathcal{M}}\right) \tag{1}$$

Here $\mathcal{M}$ denotes the mask applied on the given input $\mathbf{x}$. Equation (1) is essentially a denoising objective with the task to predict the original tokens given the masked sequence $\mathbf{x}^{\mathcal{M}}$.

We pretrain the embedding model on a mixture of text (Penedo et al., 2023) and code (Lozhkov et al., 2024) data for one epoch. As discussed in Section 4, this mixed-data pretraining produces a shared embedding model for both code and text, which may result in suboptimal in-domain performance but establishes a strong foundation for enhanced transfer learning in subsequent contrastive learning stages.

3.2 Domain-Specific Contrastive Learning

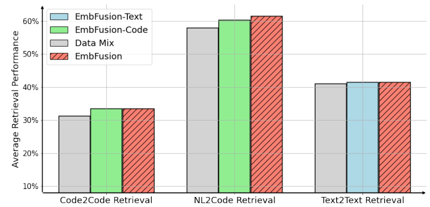
Next, we refine the representations through contrastive learning. Unlike the pretraining stage, we apply domain-specific contrastive learning on top of the pretrained model, producing domain-specific models at this stage. Our objectives are two-fold. First, to assess whether starting from a model pretrained on mixed text and code data leads to any performance drop during this contrastive learning stage, compared to performing contrastive learning on domain-specific models. Second, to determine whether domain-specific contrastive learning can effectively steer the embedding model performance in each domain, and whether this improvement can be transferred to the final model through model weight interpolation.

We leverage the naturally occuring pairs from various web sources as positives. Our datasets include (summary, code) for code with summary mined from the docstring of each function or class (Zhang et al., 2024), and (question, answer) or (title, body) mined from Web data (Wang et al., 2022). Let $\mathcal{B} = \{(\mathbf{q}_i, \mathbf{p}_i)\}_{i=1}^{M}$ denote the representations of a randomly sampled batch with M pairs, we then minimize the following to separate each positive pair from other examples within the same batch,

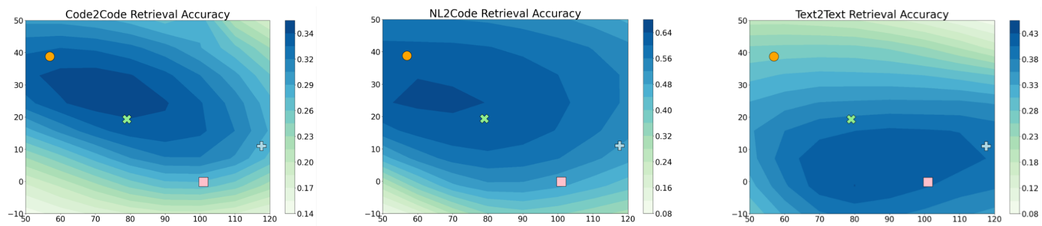
$$\mathcal{L}_{\text{CL}} = \frac{1}{M}\sum_{i=1}^{M} -\log \frac{\exp(\mathbf{p}_i \diamond \mathbf{q}_i/\tau)}{\exp(\mathbf{p}_i \diamond \mathbf{q}_i/\tau) + \sum_{k\neq i}\exp(\mathbf{q}_i \diamond \mathbf{p}_k/\tau)} \ . \tag{2}$$

τ is the temperature hyperparameter and $\diamond$ denotes cosine similarity between two vectors. We set it to 0.05 in this paper.

Given that these datasets are scraped from the internet using simple heuristics, a substantial portion of the pairs exhibit weak semantic relevance. To address this, we first apply deduplication and rule-based data filtering, followed by a consistency-based filter (Wang et al., 2022). Specifically, we train the model on the entire dataset for one epoch and remove any pair whose similarity score falls below the top-K ($K = 3$) similarity scores relative to a presampled set of 100,000 examples. After these two filtering stages, we retain 72 million summary-code pairs for code-related contrastive learning and 250 million positive pairs for text-related tasks.



(a) SAGELITE outperforms the model trained on naively mixed data.



(b) Retrieval accuracy landscape.

Figure 1: **(a)** Naive-Mix, despite starting from the same pretrained model as SAGELITE, exhibits suboptimal performance due to the gap between domain-specific and mixed contrastive learning (*i.e.,* , Naive-Mix vs SAGELITE-TEXT and SAGELITE-CODE in their respective domains). **(b)** Models achieving optimal performance tend to lie between SAGELITE-TEXT and SAGELITE-CODE, validating the effectiveness of leveraging model weights interpolation to obtain SAGELITE.

We found that consistency-based filtering is particularly effective for code-related contrastive learning. This is likely due to the fact that rule-based summary extraction from docstrings can be highly noisy. Not all docstrings in GitHub codebases are of high quality, and summaries mined based on handcrafted rules may fail to capture the main intent of the code, especially in the case of more complex code.

4 WHEN AND HOW MODEL INTERPOLATION WORKS FOR TEXT AND CODE

After the contrastive learning stage, we obtain two models, SAGELITE-TEXT and SAGELITE-CODE. The final model, SAGELITE, is then produced by averaging the weights of these two models. To better understand the effectiveness of SAGELITE, we conduct ablation studies to systematically explore its key factors. We consider the following baselines throughout our analyses: NAIVE-MIX, which trains on mixed text and code data during both pretraining and contrastive learning; TEXT-ONLY, trained exclusively on text data at both stages; CODE-ONLY, trained only on code data at both stages. Additionally, SAGELITE-TEXT begins with a pretrained mixed model and is pre-finetuned on text contrastive learning data, while SAGELITE-CODE continues training the mixed model on code contrastive learning data.

4.1 SAGELITE VS. NAIVE DATA MIX

In Figure 1a, we compare SAGELITE to a model trained using naive data mixing during both the pre-training and pre-finetuning stages. The results indicate that SAGELITE consistently outperforms the naive data mixing approach across multiple benchmarks. We hypothesize that this is because training data from different domains may require tailored learning configurations, and naive data mixing fails to optimally accommodate these domain-specific needs. To further demonstrate that naive data mixing is suboptimal, we apply a constant learning rate to both SAGELITE-TEXT and SAGELITE-CODE and construct SAGELITE by simply averaging their model weights.

To gain a more intuitive understanding of why cross-domain model interpolation is effective, we visualize the retrieval accuracy landscape in Figure 1. Following the approach in Wortsman et al. (2022a); Garipov et al. (2018), we construct an orthonormal basis for the plane spanned by the models, with the x and y axes representing movement within this parameter space along the respective

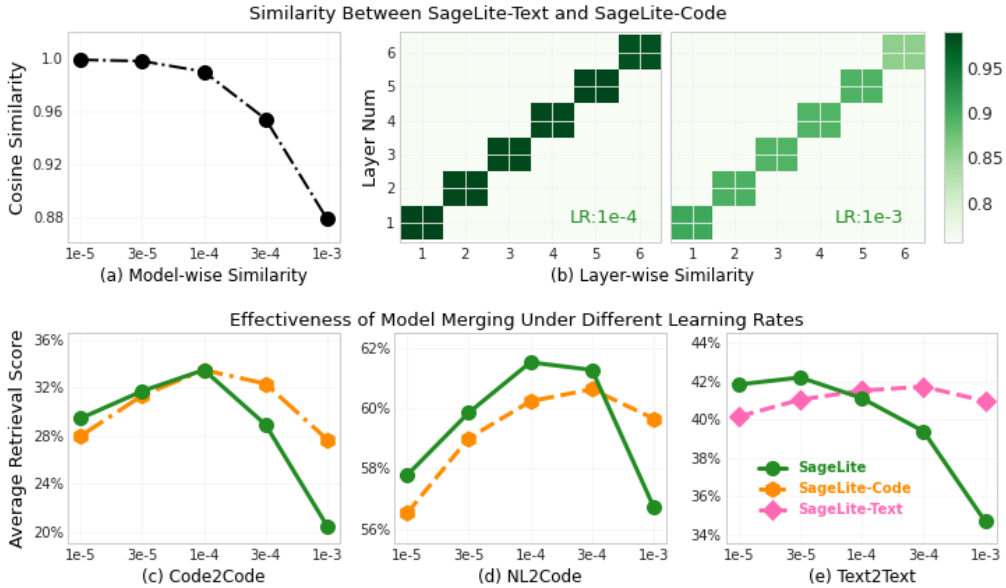


Figure 2: **(a) – (b):** Cosine similarity between models SAGELITE-TEXT and SAGELITE-CODE trained with domain-specific contrastive pre-finetuning at different learning rates. Contrastive learning maintains high similarity between the models over a wide range of learning rates. **(c) – (e):** With not too large learning rate, SAGELITE-TEXT and SAGELITE-CODE remain close and thereby the merged model SAGELITE consistently outperforms the individual domain-specific models on retrieval tasks in each domain.

directions. As illustrated, optimal performance lies between SAGELITE-TEXT and SAGELITE, with SAGELITE closely approaching this optimal region, while NAIVE-MIX remains further away from it.

4.2 MODELS ARE CLOSE

Effective model interpolation relies on certain preconditions — merging two models with the same architecture but that are too distant in parameter space can severely degrade the resulting model's performance. From this standpoint, we expect that models trained via domain-specific contrastive learning at the pre-finetuning stage should maintain sufficient similarity for model interpolation to be effective. We explore this hypothesis in Figure 2. Notably, models trained independently on text and code data *i.e.,* , SAGELITE-TEXT and SAGELITE-CODE, at the prefinetuning stage remain closely aligned across different learning rates. Specifically, the cosine similarity between these models remains above 0.96, as long as the learning rate is below 10^{-4}. Within this range, model interpolation is highly effective, as evidented by SAGELITE consistently outperforms SAGELITE-TEXT and SAGELITE-CODE as in Figures 2 (c)–(e).

As the learning rate increases, SAGELITE-TEXT and SAGELITE-CODE begin to diverge, with a monotonically decreasing similarity, as shown in Figure 2(a). As hypothesized, this divergence results in a significant drop in the performance of the interpolated model, *i.e.,* , SAGELITE underperforms compared to SAGELITE-TEXT and SAGELITE-CODE in their respective domains. However, such large learning rates are not viable options, as they also lead to degraded performance in the individual models trained via domain-specific contrastive learning, as indicated by the monotonic performance decline of both SAGELITE-TEXT and SAGELITE-CODE when the learning rate exceeds 10^{-4}.

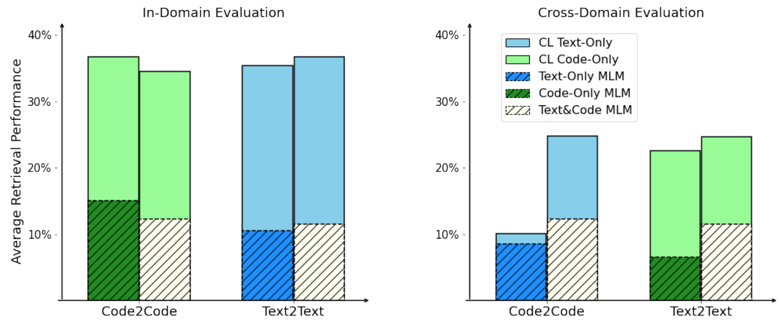


Figure 3: Compared to domain-specific pretraining, training on the mixed text and code data causes in-domain performance drop at both pretraining and the subsequent domain-specific contrastive learning stages, but it leads to better transfer learning performance (*right*).

4.3 Trade-offs Between In-Domain Accuracy and Enhanced Transfer Learning

Next, we analyze how SageLite compares to domain-specific models, which are trained exclusively on domain-specific data during both the pretraining and pre-finetuning stages. As shown in Figure 3, pretraining on mixed data leads to a decline in in-domain performance, both during pretraining and the subsequent contrastive learning stage. Figure 3 (left) demonstrates that pretraining on text-only or code-only data (*text-only* MLM and *code-only* MLM) consistently outperforms pretraining on mixed text and code data (*text&code* MLM) when evaluated on Text2Text and Code2Code retrieval tasks. This performance gap continues during the contrastive learning stage, where SageLite-Text and SageLite-Code underperform relative to domain-specific models trained with contrastive learning on top of domain-specific pretrained models.

On the other hand, Figure 3 demonstrates that the shared embeddings learned from pretraining on mixed data can significantly enhance transfer learning performance during the contrastive learning stage. Specifically, applying text-only contrastive learning to the shared embedding model outperforms its counterpart where contrastive learning is performed on a model pretrained exclusively on text data. This strong transfer learning capability can be particularly valuable in domains where contrastive learning data is scarce or challenging to acquire at scale but pretraining data is more readily available, offering a practical advantage for improving model performance.

Another notable observation from Figure 3.1 (right) is that applying text-only contrastive learning to a text-only pretrained model results in significantly lower transfer learning performance on code-related tasks, compared to the performance achieved by applying code-only contrastive learning on top of code-only pretrained models. One potential explanation for this discrepancy is the presence of text within code data during both the pretraining and contrastive learning stages, in the form of code comments, documentation, or the text summary used in the positve pairs for contrastive learning. This overlap may provide a stronger alignment between code and text representations in code-specific training.

5 Compare against the other models

To assess the versatility of SageLite, we compare it against both code and text embedding models, providing a comprehensive evaluation across different modalities. In order to sharpen the focus of our analysis, we prioritize retrieval tasks due to their increasing importance. Our objective is to evaluate the embedding models in practical scenarios where collecting supervised fine-tuning data is expensive or unfeasible. Thus, our evaluation emphasizes *zero-shot* performance.

For code-related retrieval tasks, we assess SageLite on both *Code2Code* and *NL2Code* semantic search. In Code2Code retrieval, the goal is to find relevant code snippets given a code fragment as a query. We utilize the Code2Code search set from (Zhang et al., 2024), which expands the original benchmark from (Guo et al., 2022) to cover nine programming languages instead of just three. For *NL2Code* semantic search, where natural language queries are used to retrieve relevant code, we rely

Code2Code Search

Model	Model Size	Embed Dim	Python	Java	JS	TS	C#	C	Ruby	PHP	GO	Avg
			Evaluation Languages									
GraphCodeBERT	125M	768	19.2	10.8	7.4	8.7	5.5	8.5	19.7	15.7	9.7	11.7
UnixCoder	125M	1024	30.8	16.5	21.3	22.0	6.2	15.6	32.3	31.9	13.9	21.2
OpenAI-Text-3-Small	NA	1536	25.2	12.6	8.0	9.4	5.5	15.9	30.7	23.3	11.2	15.8
OpenAI-Text-3-Large	NA	1536	40.6	25.3	20.1	22.0	11.8	31.9	42.5	41.8	21.8	28.7
CodeSage-Small	130M	1024	36.3	24.0	26.6	29.9	11.8	22.8	29.1	34.6	19.6	26.1
CodeSage-Base	356M	1024	47.5	22.8	28.7	32.0	13.4	31.0	44.7	51.1	25.2	33.0
CodeSage-Large	1.3B	2048	46.7	33.1	37.2	41.2	16.8	32.9	54.1	52.1	32.5	38.5
SageLite-X	850M	1536	**67.79**	**53.80**	**55.92**	**62.37**	**39.73**	**51.67**	**63.19**	**71.66**	**48.90**	**57.23**

NL2Code Search

Model Name	Model Size	Embed Dim	CoSQA Python	AdvTest Python	CSN Python	CSN Java	CSN JS	CSN PhP	CSN Go	CSN Ruby
GraphCodeBERT	125M	768	16.2	5.6	10.4	8.6	7.3	8.1	12.5	20.8
UnixCoder	125M	768	42.1	27.3	42.2	43.9	40.5	35.2	61.4	55.2
OpenAI-Text-3-Small	NA	1536	52.5	34.1	62.6	65.9	60.3	54.9	82.0	67.6
OpenAI-Text-3-Large	NA	3072	55.2	46.8	70.8	**72.9**	68.1	59.6	**87.6**	75.2
CodeSage-Small	130M	1024	49.9	41.3	64.4	63.2	60.0	54.7	77.7	63.2
CodeSage-Base	356M	1024	48.5	49.1	68.0	68.0	67.0	58.2	83.2	68.0
CodeSage-Large	1.3B	2048	47.5	**52.7**	70.8	70.2	69.5	61.3	83.7	71.9
SageLite-X	850M	1536	**61.0**	**54.62**	**73.9**	72.65	**69.81**	**63.91**	84.87	**76.61**

Table 1: MRR score (%) of NL2Code search in zero-shot setting.

MTEB Retrieval (Text2Text)

	Model Size	*Embed Dim*	*MS MARCO*	*Trec-Covid*	*NFCorpus*	*NQ*	*HotpotQA*	*FiQA*	*ArguAna*	*Touche-2000*	*CQADupStack*	*Quora*	*DBPedia*	*Scidocs*	*Fever*	*Climate-Fever*	*Scifact*
OpenAI-Text-3-Small	NA	1536	37.02	77.9	38.33	52.86	63.63	44.91	55.49	24.28	42.58	88.83	39.97	20.80	79.42	26.86	73.37
OpenAI-Text-3-Large	NA	3072	40.24	79.56	42.07	61.27	71.58	55.00	58.05	23.35	47.54	89.05	44.76	23.11	87.94	30.27	77.77
Gecko	1B	768	32.58	82.62	40.33	61.28	71.33	59.24	62.18	25.86	48.82	88.18	47.12	20.35	86.96	33.21	75.42
SageLite-X	850M	1536	35.9	74.0	33.6	51.2	53.9	46.7	60.8	23.6	38.1	87.2	36.7	20.24	87.2	24.0	68.7

Table 2: MTEB.

on three established benchmarks: *CoSQA* (Huang et al., 2021), *AdvTest* (Lu et al., 2021), and *CSN* (Guo et al., 2021). Additionally, for *Text Retrieval*, we employ tasks from the MTEB benchmark suite (Muennighoff et al., 2022).

We summarized our model performance in Table 1 to Table 2. As we can see SageLite achieve very strong performance in code, and comparable performance on text. We believe the perforamnce gap regarding the some text embedding models can be reduced through careful synthetic data curation.

5.1 More about Generalization

Given synthetic data becomes increasingly prevalent, we propose two purely LLM generated synthetic benchmark datasets to assess the generalization of a subset of selected embedding models. The multi-stage benchmark data generation pipeline began by brainstorming diverse retrieval tasks, then generated corresponding queries, positive, and negative documents. After the initial task generation, we refined and deduplicated the tasks to maximize coverage and diversity. Our goal was to assess model performance on LLM-generated datasets as

Data Generation Pipeline The synthetic text and code benchmark datasets were constructed using pipelines designed to evaluate nuanced retrieval performance. The text pipeline started with generating diverse retrieval tasks, covering a broad range of potential information retrieval scenarios. The code pipeline similarly produced a variety of developer-focused retrieval scenarios. These foundational tasks were chosen to simulate realistic search contexts, promoting diversity in query types and document structures. Next, examples were created to challenge retrieval models' sensitivity to fine-grained similarities. The code dataset paired queries with relevant snippets and deceptive negatives—incorrect but plausible code—to mimic real-world debugging and search complexities. The use of misleading negatives was intentional, designed to evaluate the models' ability to distinguish between closely related but semantically distinct examples.

Dataset Name	# Queries	Min/Median/Max Query Token Counts	# Corpus Documents	Min/Median/Max Corpus Docs Token Counts
Synthetic Text Benchmark	1296	4/9/18	14106	87/270/1217
Synthetic Code Benchmark	324	3/9/19	1691	6/105/875

Table 3: Entirely synthetic evaluation benchmark statistics.

Model	Model Size	Embed Dim	Text Synthetic	Code Synthetic
e5-base	109M	768	43.3	42.1
gte-base-en-v1.5	137M	1024	66.8	52.89
multilingual-e5-large	560M	1024	48.5	43.2
e5-mistral-7b-instruct	7B	4096	82.9	69.3
SFR-Embedding-Mistral	7B	4096	**83.9**	**68.7**
CodeSage-Large	1.3B	2048	32.2	48.6
SageLite-X	850M	1536	73.7	61.6

Table 4: Evaluation results on the text and code synthetic data benchmarks.

For text, we generated queries with short positive document summaries before additional stages that expanded summaries into detailed documents. We also generated hard negatives, ensuring the final dataset could effectively test the models' robustness in distinguishing subtle contextual nuances—a key requirement for evaluating contrastive learning models in both text and code domains. The dataset distributions can be found in 3, and full prompts used to construct the synthetic text benchmark and synthetic code benchmark can be found in the Appendix C.2.

Evaluation We conducted evaluation on our benchmark datasets through using the generated queries as the search queries, and all positive and negative generated documents as the search corpus. Full evaluation results can be found in 4. Both encoder-based and decoder-based embedding models were considered. Notably, E5-mistral-7b-instruct Wang et al. (2024) and SFR-Embedding-Mistral Meng et al. (2023) achieved superior performance across both text and code tasks. This outcome is expected, given their generative nature and fine-tuning on synthetic data collected in a manner similar to our benchmark, enabling them to handle synthetic data effectively. Additionally, their significantly larger model sizes confer an advantage. Following these models, SageLite-X demonstrated the next highest performance across both text and code benchmarks, indicating its strong capability in handling diverse data types, including synthetic datasets, which are becoming increasingly prevalent.

6 Conclusion

In conclusion, SageLite demonstrates a significant advancement in the field of code embeddings by effectively leveraging a combination of text and code data, followed by rigorous unsupervised contrastive learning. Through this approach, the model not only excels in capturing shared representations across text and code but also achieves fine-grained semantic discrimination crucial for domain-specific generalization. Our results suggest that this comprehensive framework for integrating knowledge from diverse sources can open new avenues for scalable, adaptable, and efficient embedding model.

References

Marie anne Lachaux, Baptiste Roziere, Marc Szafraniec, and Guillaume Lample. DOBF: A deobfuscation pre-training objective for programming languages. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=3ez9BSHTNT.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1xMH1BtvB`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

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

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020. doi: 10.18653/v1/2020.findings-emnlp.139. URL `https://aclanthology.org/2020.findings-emnlp.139`.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcode{bert}: Pre-training code representations with data flow. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=jLoC4ez43PZ`.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*, 2022.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. Cosqa: 20,000+ web queries for code search and question answering. *arXiv preprint arXiv:2105.13239*, 2021.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019. URL `https://arxiv.org/abs/1909.09436`.

Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems*, 35:29262–29277, 2022.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Xue Jiang, Zhuoran Zheng, Chen Lyu, Liang Li, and Lei Lyu. Treebert: A tree-based pre-trained model for programming language. In *Uncertainty in Artificial Intelligence*, pp. 54–63. PMLR, 2021.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Learning and evaluating contextual embedding of source code. In *International conference on machine learning*, pp. 5110–5121. PMLR, 2020.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*, 2024.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Xianming Li and Jing Li. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*, 2023.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=6lE4dQXaUcb`.

Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. Sfr-embedding-mistral: Enhance text retrieval with transfer learning. *arXiv preprint arXiv:2401.00368*, 2023.

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

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A Raffel. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36, 2024a.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. *arXiv preprint arXiv:2402.09906*, 2024b.

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

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*, 2021.

OpenAI. Gpt-4 technical report, 2023.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023. URL `https://arxiv.org/abs/2306.01116`.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. *Salesforce AI Research Blog*, 2024.

Xin Wang, Yasheng Wang, Fei Mi, Pingyi Zhou, Yao Wan, Xiao Liu, Li Li, Hao Wu, Jin Liu, and Xin Jiang. Syncobert: Syntax-guided multi-modal contrastive pre-training for code representation. *arXiv preprint arXiv:2108.04556*, 2021a.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, 2021b. doi: 10.18653/v1/2021.emnlp-main.685. URL `https://aclanthology.org/2021.emnlp-main.685`.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

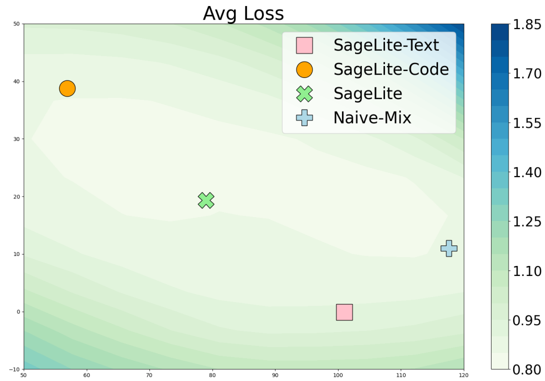


Figure 4: The average discrimination loss achieved by SAGELITE and its domain-specific components, SAGELITE-TEXT and SAGELITE-CODE. The results show that models achieving optimal performance consistently lie between SAGELITE-TEXT and SAGELITE-CODE, validating our strategy of obtaining a versatile model through interpolation.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. PMLR, 2022a.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7959–7971, 2022b.

Dejiao Zhang, Shang-Wen Li, Wei Xiao, Henghui Zhu, Ramesh Nallapati, Andrew O Arnold, and Bing Xiang. Pairwise supervised contrastive learning of sentence representations. *arXiv preprint arXiv:2109.05424*, 2021.

Dejiao Zhang, Wasi Uddin Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. CODE REPRESENTATION LEARNING AT SCALE. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=vfzRRjumpX`.

A APPENDIX

B DATA, MODEL, AND HYPER-PARAMETERS DETAILS

C EVALUATION

C.1 BASELINE MODELS

Code Embedding Models We compare our models against four general-purpose code representation learning encoders. Both CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2021) are trained with standard MLM on six programming languages using CodeSearchNet (Husain et al., 2019)[1], while the replaced token detection objective (Clark et al., 2020) and data flow prediction objectives are adopted as auxiliary objectives, respectively. UnixCoder (Guo et al., 2022) is trained via three language modeling and two contrastive learning objectives using the same dataset. More recently, StarEncoder (Li et al., 2023) is trained with MLM and next sentence prediction (Devlin et al., 2019) on 86 programming languages from The Stack (Kocetkov et al., 2022). Zhang et al. (2024) is the current state-of-the-art code embedding models on Code2Code and NL2Code search performance among open-sourced embedding models.

[1]The dataset includes 2.3M functions paired with natural language documents.

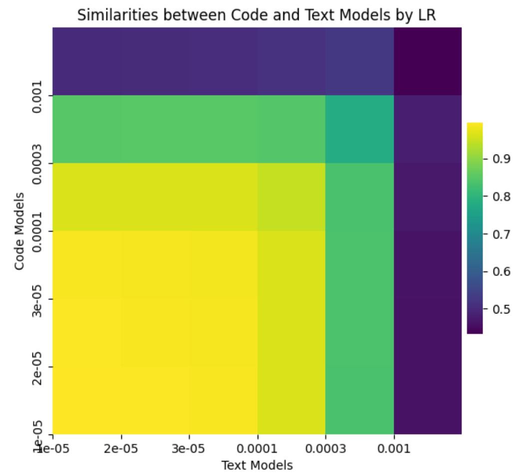


Figure 5: This figure shows the similarity between text and text models for various learning rates.

C.2 Synthetic Benchmark Prompts and Examples

Text Retrieval Prompts: Stages 2-4

Stage 2 Prompt. Text retrieval task: `task`. Create one text retrieval example using the format provided. The JSON object must contain the following keys:

- **user_query:** A string, a random user search query specified by the retrieval task.
- **summary:** A string, a short summary of a relevant document for the user_query. Try not to repeat exact phrases from the query in this positive, accurate summary. The summary should be between 20-50 words.

Stage 3 Prompt. Given the following query and short summary, produce a document of 128-1024 words. The document should contain the information presented in the short summary, but expand the document to include more context and related information. Do not copy exact phrases or answers in the document; instead, try to be creative in how you present the information. The document should seem independently written, and not reference the summary nor the query. Query: query Summary: summary Please output your completed document in ¡document¿ tags.

Stage 4 Prompt. Given the following query and short summary, produce a document of 128-1024 words each that does not contain the answer to the query. The document should contain related information presented in the short summary, but expand the document to include more context and related information. Try to copy exact phrases or answers from the summary and query, and be creative in how you present the information. The document should a be misleading answer to the user query; it must be incorrect, or be on a related but different subject. The document should not reference the summary nor the query, nor reference the fact that it is wrong or misleading. Query: query Summary: summary Please output your completed document in ¡document¿ tags.

Code Retrieval Prompt: Stage 2

Given the following code retrieval task: task. Create one code retrieval example using the format provided. The JSON object must contain the following keys:

- **user_query:** A string, a random developer search query specified by the retrieval task. The query should require a code response of 20-100 lines of code.
- **positive_code:** A string, a correct chunk of code or relevant document for the developer query with no explanation. The code should be 20-100 lines long.
- **incorrect_code:** A list of 4 strings, all incorrect code responses to the developer query. These can include buggy implementations or code with errors, or code that responds to another similar query. These responses must not be accurate responses to the developer search query. Each negative code snippet should be 20-100 lines long.

Sample Text Retrieval Example

Listing 1: A structured example for the text retrieval task.


```
{
  "user_query": "Impact of urban growth on endangered species in
      city outskirts",
  "positive_document": "Urbanization, the process by which rural
      areas transform into urban environments, brings with it a
      multitude of changes to the landscape. Forests, wetlands, and
      grasslands are often cleared to make way for residential
      developments, commercial buildings, and infrastructure
      projects. This transformation is particularly destructive for
      endangered species whose survival relies on the integrity of
      these natural habitats. The effects of urban growth can
      primarily be seen in two critical areas: habitat fragmentation
       and habitat degradation.

  Habitat fragmentation occurs when continuous habitats are divided
       into smaller, isolated patches by roads, buildings, and other
       urban structures. For many species, a continuous habitat is
      essential for their survival, as it allows for movements
      related to breeding, foraging, and escaping predators. When
      these habitats are broken up, it restricts the natural
      movements of wildlife, leading to decreased genetic diversity,
       inbreeding, and increased vulnerability to environmental
      changes. For example, certain amphibians, which have specific
      breeding sites often located in wetland areas, may find it
      increasingly difficult to reach these sites when their habitat
       is fragmented by new housing developments and roadways.

  In the face of expanding cities, habitat degradation also poses a
       significant challenge. Habitat degradation involves the
      reduction in quality and usability of an environment for
      wildlife, often through pollution, introduction of invasive
      species, and general disturbances from human activities. As
      urban areas expand, pollutants from vehicles, factories, and
      human settlements can seep into the surrounding land and water
       bodies. Such changes can disrupt the delicate balance of
      ecosystems, making it difficult for native species to thrive.
      Native plants and animals might be replaced by non-native or
      invasive species that can better tolerate the altered
      conditions, leading to further declines in native populations.

  Moreover, one cannot overlook the impact of human-induced light
      and noise pollution, which significantly affect wildlife
      behaviors. Many species depend on natural light cycles for
      activities like migration and reproduction. Urban lights can
      disorient birds during migration, causing fatal collisions
      with buildings. Similarly, increased noise levels from traffic
       and industrial activities can disturb communication and
      breeding of various species, especially those that rely on
      acoustic signals.\n\nAnother aspect to consider is climate
      change, which is exacerbated by urbanization. The creation of
      heat |islandswhere urban areas become significantly warmer
      than their rural surroundings due to human |activitiesalters
      local climate conditions. This can shift the distribution of
      species, further adding to the stress on those already
      endangered. Many species are specifically adapted to their
      regional climate, and changes can render their current
      habitats unsuitable, forcing them to migrate, where possible,
      to more hospitable areas or face heightened risk of extinction
      .

    ...
  "
}
```

Sample Code Retrieval Example

Listing 2: A structured example for the code retrieval task.

```
{
  "user_query": "How do I calculate the determinant of a matrix in C++?",
  "positive_document":
"#include <iostream>
#include <vector>

using namespace std;

typedef vector<vector<double>> Matrix;

double determinant(Matrix mat, int n) {
    double det = 0;
    if (n == 1)
        return mat[0][0];
    if (n == 2)
        return (mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]);

    Matrix subMatrix(n, vector<double>(n));
    for (int x = 0; x < n; x++) {
        int subi = 0;
        for (int i = 1; i < n; i++) {
            int subj = 0;
            for (int j = 0; j < n; j++) {
                if (j == x)
                    continue;
                subMatrix[subi][subj] = mat[i][j];
                subj++;
            }
            subi++;
        }
        det = det + (pow(-1, x) * mat[0][x] * determinant(subMatrix, n - 1));
    }
    return det;
}

int main() {
    Matrix mat = {{1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9}};

    cout << \"Determinant: \" << determinant(mat, 3) << endl;
    return 0;
}",
  "hard_negative_documents": [
"#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<double>> Matrix;

double computeDeterminant(Matrix mat, int n) {
    double det = 1;
    for (int i = 0; i < n; i++) {
        det *= mat[i][i];
    }
    return det;
}
```

```

int main() {
    Matrix mat = {{2, 1},
                  {5, 3}};

    cout << \"Determinant: \" << computeDeterminant(mat, 2) << endl;
    return 0;
}",
    "#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int>> Matrix;

Matrix invertMatrix(Matrix mat) {
    // Implementation for inverting matrix goes here
}

int main() {
    Matrix mat = {{4, 7},
                  {2, 6}};

    Matrix invertedMat = invertMatrix(mat);

    // Code to display inverted matrix
    return 0;
}",
    "#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int>> Matrix;

int calculateTrace(Matrix mat, int n) {
    int trace = 0;
    for (int i = 0; i < n; i++) {
        trace += mat[i][i];
    }
    return trace;
}

int main() {
    Matrix mat = {{1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9}};

    cout << \"Trace: \" << calculateTrace(mat, 3) << endl;
    return 0;
}"
  ]
}
```