

# CoRet: Improved Retriever for Code Editing

**Fabio Fehr\***  
EPFL & Idiap Research Institute  
fabio.fehr@epfl.ch

**Prabhu Teja Sivaprasad**  
AWS AI Labs  
prbuteja@amazon.de

**Luca Franceschi**  
AWS AI Labs  
franuluc@amazon.de

**Giovanni Zappella**  
AWS AI Labs  
zappella@amazon.de

## Abstract

In this paper, we introduce CoRet, a dense retrieval model designed for code-editing tasks that integrates code semantics, repository structure, and call graph dependencies. The model focuses on retrieving relevant portions of a code repository based on natural language queries such as requests to implement new features or fix bugs. These retrieved code chunks can then be presented to a user or to a second code-editing model or agent. To train CoRet, we propose a loss function explicitly designed for repository-level retrieval. On SWE-bench and Long Code Arena’s bug localisation datasets, we show that our model substantially improves retrieval recall by at least 15 percentage points over existing models, and ablate the design choices to show their importance in achieving these results.

## 1 Introduction

Code editing is an important task that often requires developers to make changes to code repositories based on explicit natural language descriptions such as a GitHub pull request about a bug, a new feature, or about an error in the code. Successful code editing requires correct navigation and retrieval of relevant sections of the repository. This process demands a repository-level understanding of the code functionality (semantics), organisation (repository hierarchy), and relationships between various entities in the codebase (*e.g.* runtime dependencies). Retrieving multiple relevant code chunks is especially difficult in large, real-world repositories (Jimenez et al., 2024). Retrieval is important for both coding agents and humans and forms an important first step in the overall code editing process.

We find that existing pretrained encoder models perform poorly in repository-level retrieval for code-editing. We conjecture the reason for this

is threefold: the representations do not align the problem statements to segments of code; the structure from the repository hierarchy is lost; and the runtime dependencies within a repository are not captured. Models like CodeSage (Zhang et al., 2024a) capture docstring-code relationships effectively. Yet, we find that this capacity does not readily transfer to retrieval for code-editing tasks. Similarly, models like CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), and UniX-coder (Guo et al., 2022) focus on code semantics but miss repository-level structures, limiting their generalisation to this task.

Incorporating additional context has been long known to improve the quality of retrieval (Khattab and Zaharia, 2020; Zhu et al., 2024). The nature of code presents challenges in determining how to integrate context and what type of context to use. For tasks like code completion and summarisation, using chunks from other files within the same repository as context improves performance (Bansal et al., 2021; Ding et al., 2024). Bansal et al. (2023) argue that the call graph dependency structure is required to understand the semantics of code and consider embedding code subroutines together. We follow this direction.

Our work introduces a dense retrieval model for code-editing tasks. We show that optimising directly the likelihood of the model to retrieve correct code sections brings significant advantages over standard contrastive losses. Moreover, we show how to effectively incorporate file hierarchy and call graph context. These contributions lead to improvements of at least 15 percentage points recall over the baseline methods on SWE-bench and Long Code Arena.

## 2 Problem setup

Representing a code repository in structurally-informed and semantically succinct units is imper-

\*Work done during an internship at AWS, Berlin.

ative for retrieval performance (Shrivastava et al., 2023; Zhang et al., 2023b; Liao et al., 2024). An obvious atom of a repository is a file. However, files may be particularly long and may lack semantic coherence. Therefore, we further break down each file into its constituent functions, classes and methods of classes, as illustrated in Figure 1 (bottom). We refer to each such atom as a *chunk*. Further details on the chunking procedure are available in Appendix B.

The objective of code retrieval is to return all parts of the codebase that are pertinent to the current input. Let  $\mathcal{C}$  represent a code repo as a set of atoms:  $\mathcal{C} = \{c_1, \dots, c_M\}$ . We represent with  $q$  a natural text input that describes an issue in the repository. Our dataset  $\mathcal{D}$  consists of  $N$  triplets of issue descriptions, associated codebase, and the ground truth atoms *i.e.*  $\mathcal{D} = \{(q_i, \mathcal{C}_i, \mathcal{C}_i^*)\}_{i=1}^N$  where  $\mathcal{C}_i^* \subseteq \mathcal{C}_i$ . We refer to each triplet as an example, or instance. A retriever is a function  $f : (q, c_i) \rightarrow [0, 1]$  for  $c_i \in \mathcal{C}$  that assigns a numerical score to each of the atoms in  $\mathcal{C}$ . A ranking can be induced on the atoms by sorting the scores. We parameterize  $f(q, c_i; \theta_q, \theta_c) = \text{sim}(Q(q; \theta_q), C(c_i; \theta_c))$ , where  $Q$  and  $C$  are embedding networks for  $q$  and  $c_i$  respectively, and ‘sim’ is the cosine similarity between them. The ideal retriever induces a ranking that puts all the relevant atoms before the irrelevant ones. In practice, we extract the top- $k$  most similar atoms to the query, denoted by  $\mathcal{C}_F$ , where  $k \ll |\mathcal{C}|$  *i.e.*  $\mathcal{C}_F = \arg \text{top } k_{c \in \mathcal{C}} [f(q, c)]$ . We leave the dependence of  $\mathcal{C}_F$  on  $k$  implicit. For brevity, we represent  $Q(q; \theta_q)$  with  $\mathbf{q}$  and  $C(c_i; \theta_c)$  with  $\mathbf{c}_i$ . During inference, given a (test) codebase and a query in natural language, the top- $k$  most similar atoms retrieved are from the set  $\mathcal{C} = \{c_1, \dots, c_M\}$  where  $M$  is the number of atoms (chunks) in this codebase.

### 3 Proposed method

Here we describe our method to train encoder models for the specific task of code retrieval.

#### 3.1 Training

The goal of training the retrieval model  $f$  is to learn an embedding space where the query and the relevant code chunks have higher similarity than irrelevant code chunks. Each instance  $i \in [N]$ , contains a single query  $q_i$  and multiple ground truth code chunks  $c_j^*$ . These form positive pairs  $(q_i, c_j^*)$ . The remaining pairs  $(q_i, c_k)$ , for  $c_k \in \mathcal{C} \setminus \mathcal{C}^*$ , are

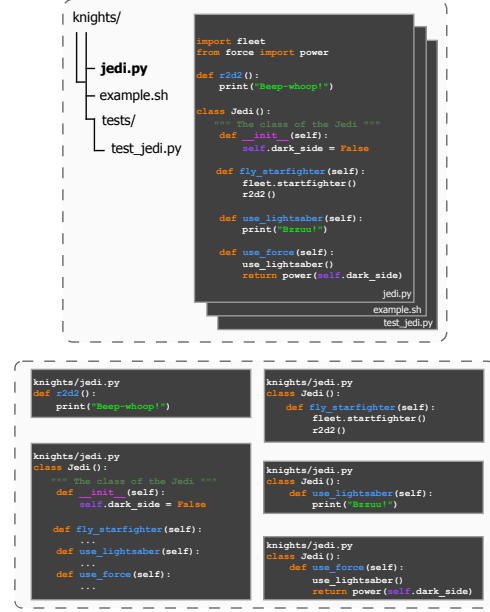


Figure 1: **Top:** Code repository before processing. **Bottom:** Code chunks after filtering and chunking.

negative pairs. We optimise the following loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_i \frac{1}{|\mathcal{C}_i^*|} \sum_{c^* \in \mathcal{C}_i^*} \log \frac{\exp\left(\frac{\mathbf{q}_i \cdot \mathbf{c}^*}{\tau}\right)}{\Gamma(\mathbf{q}, \mathcal{C}_i)}, \quad (1)$$

where  $\theta = (\theta_q, \theta_c)$  are the parameters of the model and  $\Gamma(\mathbf{q}, \mathcal{C}) = \sum_{c \in \mathcal{C}} \exp\left(\frac{\mathbf{q} \cdot \mathbf{c}}{\tau}\right)$  is the normalising factor. This is the mean likelihood of retrieving the code chunk per model, and is akin to the standard cross-entropy loss for multi-class classification.

Since the normalising factor involves a summation over all the chunks in a code repository (which can be in the order of 10000 chunks), we implement an approximation by considering only a random subset of instances:

$$\tilde{\Gamma}(\mathbf{q}, \mathcal{C}^*) = \exp\left(\frac{\mathbf{q}_i \cdot \mathbf{c}^*}{\tau}\right) + \sum_{c \in \mathcal{B}} \exp\left(\frac{\mathbf{q}_i \cdot \mathbf{c}}{\tau}\right) \quad (2)$$

where  $\mathcal{B} \subset \mathcal{C}$  is random sample of within-instance negatives and  $\tau$  is a temperature parameter. Following Zhang et al. (2024a), we set  $\tau = 0.05$  throughout. Prior works (see Appendix A) primarily use contrastive losses for feature learning, whereas we apply a standard log-likelihood loss in this setting.

#### 3.2 Call graph context for code-editing

Context has been shown to significantly improve the quality of retrieval (Lewis et al., 2020b; Günther et al., 2024; Borgeaud et al., 2022). A code repository has a natural relationship between its chunks

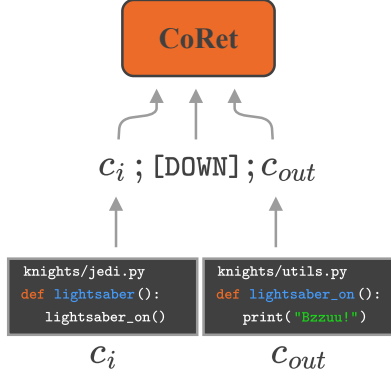


Figure 2: Given a function  $c_i$  and its downstream neighbour  $c_{out}$ , we concatenate the strings as  $c_i; [\text{DOWN}]; c_{out}$ , including the special separation token, and fine-tune the model to obtain **CoRet**.

endowed by the call graph (Ahn et al., 2009; Bansal et al., 2023). The neighbours in the call graph are the code entities that are invoked by or invoke the current chunk of interest. We propose to enrich each chunk  $c_i$  in  $\mathcal{C}$  with its call graph neighbours  $\mathcal{N}(c_i)$ . To incorporate this information, we modify the chunk embedder  $C$  to accept that information as  $C(c_i, \mathcal{N}(c_i); \theta_c)$ . Several implementations for this  $C$  are possible: we retain the same network as without the call graph information, but add the call graph chunks in the input itself *i.e.*  $C([c_i; \mathcal{N}(c_i)]; \theta_c)$ . We also use only downstream neighbours by introducing a new token `[DOWN]` that denotes this relationship. For instance, a chunk with one incoming and one outgoing edge is represented as  $c_i; [\text{DOWN}]; c_{out}$ , where ‘;’ denotes string concatenation. See Figure 2 for an example of how the call graph context is fed to the model. We follow BERT (Devlin et al., 2019) in adding token segment type embeddings *i.e.* trainable embeddings that signify  $c_i$  and  $\mathcal{N}(c_i)$ .

## 4 Experiments

### 4.1 Dataset

For training, we consider repository-level code-editing problems which contain a language problem statement  $q$  and a code repository  $\mathcal{C}$  from SWE-bench (Jimenez et al., 2024). The ground truth pull requests are parsed to obtain the ground truth chunks  $\mathcal{C}^*$  which correspond to the edited code chunks. We evaluate our trained models on SWE-bench Verified, and Long Code Arena (LCA) bug localisation (Bogomolov et al., 2024). We provide

further dataset statistics in Appendix D.

### 4.2 Metrics

We measure the performance of our model using standard retrieval metrics:  $\text{recall}@k$  and mean reciprocal rank (MRR).  $\text{Recall}@k$  measures how many of the ground truth chunks in  $\mathcal{C}^*$  are retrieved when  $k$  most similar chunks are retrieved. MRR measures the minimum  $k$  needed to retrieve at least one correct code chunk; see Appendix C for formal definitions. Additionally, when multiple chunks have to be retrieved, as in the case of LCA, we show the performance metric  $\text{Perfect-Recall}@k$  which is a binary value for each instance if all correct chunks were retrieved at  $k$ . This is better suited to measure improvements as partial retrievals are not useful for subsequent code editing.

$$\text{Perf-Recall}@k = \frac{1}{N} \sum_{i=1}^N \begin{cases} 1 & \text{if } \mathcal{C}_{F_i} \cap \mathcal{C}_i^* = \mathcal{C}_i^*, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

We prioritize recall in our choice of metrics because retrieving all the necessary code chunks is necessary to solve a task, while the impact of retrieving unnecessary chunks (*i.e.* low precision) is not clear.

### 4.3 Implementation

We implement CoRet using CodeSage Small (S) (Zhang et al., 2024a) as our pre-trained backbone with the following modification: we use mean pooling over all chunk tokens instead of using the standard `[CLS]` token. This modification resulted in a moderate performance boost in early experiments. We also tie the weights for both  $C$  and  $Q$  models, meaning  $\theta_c = \theta_q$ , which we initialise with the publicly available CodeSage S weights. We found that weight tying performs consistently better in our experiments than letting  $C$  and  $Q$  vary independently during training. Further model and implementation details are in Appendix E and Appendix G.

### 4.4 Results

**Existing models are sub-optimal when used for retrieval in code editing.** We compare our model to standard methods such as BM25 (Trotman et al., 2014), several text-code encoder models like CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), UniXcoder (Guo et al., 2022), and CodeSage (Zhang et al., 2024a). In Figure 3, we present each baseline model retrieval performance

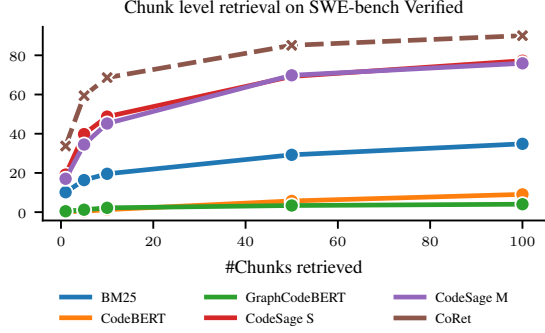


Figure 3: Recall@ $k$  for the baseline models and our proposed method on instances of SWE-bench Verified. CodeSage-family models substantially outperform other baselines. CoRet, described in Section 3, consistently outperforms baselines across all  $k$ . The dashed line corresponds to the trained CoRet model, whereas the solid lines correspond to the untrained baselines.

	SWE Verified			LCA		
Model	@5	@20	MRR	@5	@20	MRR
CodeSage S	0.34	0.51	0.35	0.26	0.34	0.28
CoRet - CG	0.52	0.69	0.52	<b>0.32</b>	0.41	0.45
CoRet - CG + file	<b>0.54</b>	0.69	0.52	0.29	0.38	0.44
CoRet	<b>0.54</b>	<b>0.71</b>	<b>0.53</b>	<b>0.32</b>	<b>0.47</b>	<b>0.47</b>

Table 1: Perfect recall at chunk level on code-editing retrieval tasks.

on SWE-bench Verified in solid lines. Among the models considered, CodeSage S and CodeSage M perform the best with the caveat that CodeSage M’s performance comes with a much higher computational resource requirements. For this reason, we pick CodeSage S model as our pretrained backbone for further finetuning.

**Representation learning focused on retrieval improves performance.** In Figure 3, the dashed line reports the performance of CoRet after training. We report the perfect chunk recall in Table 1. It is evident that CoRet improves upon the best baseline CodeSage S significantly. On SWE-bench Verified, recall@5 improves by 52.9% compared to CodeSage S for recall@5 and by about 35% for recall@20.

**Call graph context improves multi-chunk retrieval.** Next, we ablate on the call graph context to assess the contribution of including this additional information during training and inference. Table 1 reports perfect chunk recall on our two evaluation datasets. CoRet - CG indicates our method without the call graph context described in Section 3.2. We present results of additional baselines in Figure 4

	wo/filepath			w/filepath		
Model	@5	@20	MRR	@5	@20	MRR
BM25	0.15	0.21	0.14	0.16	0.22	0.16
CodeSage S	0.40	<b>0.58</b>	0.33	0.40	0.57	0.35
CoRet	<b>0.42</b>	<b>0.58</b>	<b>0.42</b>	<b>0.53</b>	<b>0.70</b>	<b>0.53</b>

Table 2: Chunk-level accuracy on SWE-bench Verified, comparing CoRet without (wo/filepath) and with (w/filepath) file path input. CoRet is fine-tuned with file paths, highlighting performance drop without them. CoRet performance improves with file path context.

in the Appendix and provide a summary here. We further perform an ablation (CoRet - CG + file) where we include as additional context a number of chunks randomly sampled from the same file as the target chunk, and we leave the rest of the pipeline unchanged. We observe a substantial decrease in performances throughout, further validating our design choice.

**Choice of negative samples in Equation (2) influences the performance.** Traditional methods for representation learning using contrastive losses (Chopra et al., 2005; Chen et al., 2020; Karpukhin et al., 2020) form negative samples by using positives from elements across the batch. We term this *across-instance negatives*. In Equation (2), we form the negatives from entirely within a problem’s repository and not across problems, like in Sohn (2016a), and we term this *in-instance negatives*. For our experiments we randomly draw up to a maximum of 1024 negative samples to compute the term  $\tilde{I}$  in our loss (Equation (2)) and consider the influence of the number of negatives in Figure 5 in the Appendix. We find improvements for all  $k$  when we include in-instance negatives. Additionally, the impact of number of negatives is also evident; the larger the number of negatives, the better the performance. Increasing the number of negatives from 8  $\rightarrow$  1024 improves the recall@20 by almost 10 points. This provides further evidence that Equation (1) with the approximation from Equation (2) mimics the goal of a retriever: it explicitly models which code chunks are relevant and which are not.

**File hierarchy is an important feature for retrieval.** We prefix the file path to each chunk as a part of the chunking strategy. We find that it is an important feature. In Table 2, we show the retrieval performance drops for CoRet when file paths are removed from the chunk representation for infer-



ence, as the models are originally trained with the file path present. The performance difference is minimal for both BM25 and CodeSage S whereas CoRet learns to rely on the file name through training. Additional evidence from an attention matrix is presented in Appendix I.

## 5 Conclusions

In this work, we present an explicit study of repository-level code-editing retrieval. We propose a method for training models specifically designed for this task and demonstrate that existing retrieval models are suboptimal in this setting. We identify that this problem differs from the traditional contrastive representation learning problem and propose a loss function that substantially improves the performance of retrieval models compared to using standard contrastive losses. Further, incorporating code context from neighbours in the call graph gives an additional boost in performance. We speculate that further improvement may come with better strategies to select relevant neighbours, *e.g.* by leveraging topological properties of the call graph (Tsourakakis et al. (2017); Chiang et al. (2014); Cesa-Bianchi et al. (2012)). We hope this work highlights the importance of retrieval for code editing and inspires further research to advance models and techniques in this domain.

## Limitations

Our work can be extended in several ways. For instance, our experiments are restricted to Python, and are based on two datasets - SWE-bench and LCA. At the time of the submission they were the only datasets that allowed for repository level code retrieval problems. Recently, a multi-language dataset called SWE-PolyBench (Rashid et al., 2025) has been released and we plan to include it in our future work on this topic.

Furthermore, SWE-bench Verified requires to modify, for the large part, one file in each test case. Thus, file recall can be very high. LCA features edits in multiple files, and thus is a better repository to benchmark. The data preparation and chunking step, while extensible, can be expensive to implement for languages other than Python.

Our investigation has been limited to encoder models for their feature prediction abilities. Several modern LLMs have been modified to output feature embeddings (BehnamGhader et al., 2024; Tao et al., 2024). It is also likely that these LLMs

trained on large corpus can provide better baseline performance, but training them requires more resources.

## References

- Shin-Young Ahn, Sungwon Kang, Jongmoon Baik, and Ho-Jin Choi. 2009. [A weighted call graph approach for finding relevant components in source code](#). 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, pages 539–544.
- Aakash Bansal, Zachary Eberhart, Zachary Karas, Yu Huang, and Collin McMillan. 2023. [Function call graph context encoding for neural source code summarization](#). *IEEE Transactions on Software Engineering*, 49:4268–4281.
- Aakash Bansal, Sakib Haque, and Collin McMillan. 2021. [Project-level encoding for neural source code summarization of subroutines](#). In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 253–264.
- Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. [LLM2vec: Large language models are secretly powerful text encoders](#). In *First Conference on Language Modeling*.
- Egor Bogomolov, Aleksandra Eliseeva, Timur Galimzyanov, Evgeniy Glukhov, Anton Shapkin, Maria Tigina, Yaroslav Golubev, Alexander Kovrigin, Arie van Deursen, Maliheh Izadi, and Timofey Bryksin. 2024. [Long code arena: a set of benchmarks for long-context code models](#). *Preprint*, arXiv:2406.11612.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. 2022. [Improving language models by retrieving from trillions of tokens](#). In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.
- Nicolo Cesa-Bianchi, Claudio Gentile, Fabio Vitale, and Giovanni Zappella. 2012. A correlation clustering approach to link classification in signed networks. In *Conference on Learning Theory*, pages 34–1. JMLR Workshop and Conference Proceedings.
- Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

- Kai-Yang Chiang, Cho-Jui Hsieh, Nagarajan Natarajan, Inderjit S Dhillon, and Ambuj Tewari. 2014. Prediction and clustering in signed networks: a local to global perspective. *The Journal of Machine Learning Research*, 15(1):1177–1213.
- S. Chopra, R. Hadsell, and Y. LeCun. 2005. [Learning a similarity metric discriminatively, with application to face verification](#). In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: Pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Yanguibo Ding, Zijian Wang, Wasi Ahmad, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2024. [CoCoMIC: Code completion by jointly modeling in-file and cross-file context](#). In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 3433–3445, Torino, Italia. ELRA and ICCL.
- Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. [CodeBERT: A pre-trained model for programming and natural languages](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.
- Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan, Benjamin Van Durme, and Jamie Callan. 2021. [Complete lexical retrieval model with semantic residual embeddings](#). In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part I*, pages 146–160, Berlin, Heidelberg. Springer-Verlag.
- Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. [UniXcoder: Unified cross-modal pre-training for code representation](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.
- 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. 2021. [Graphcode{bert}: Pre-training code representations with data flow](#). In *International Conference on Learning Representations*.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. [Deepseek-coder: When the large language model meets programming - the rise of code intelligence](#). *ArXiv*, abs/2401.14196.
- Michael Günther, Isabelle Mohr, Daniel James Williams, Bo Wang, and Han Xiao. 2024. [Late chunking: Contextual chunk embeddings using long-context embedding models](#). *Preprint*, arXiv:2409.04701.
- Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. [CoSQA: 20,000+ web queries for code search and question answering](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5690–5700, Online. Association for Computational Linguistics.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. [Code-searchnet challenge: Evaluating the state of semantic code search](#). *CoRR*, abs/1909.09436.
- Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. [Unsupervised dense information retrieval with contrastive learning](#). *Trans. Mach. Learn. Res.*, 2022.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. [SWE-bench: Can language models resolve real-world github issues?](#) In *The Twelfth International Conference on Learning Representations*.
- Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. [Pre-trained contextual embedding of source code](#).
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. [Dense passage retrieval for open-domain question answering](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.
- Omar Khattab and Matei Zaharia. 2020. [Colbert: Efficient and effective passage search via contextualized late interaction over bert](#). In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20*, page 39–48, New York, NY, USA. Association for Computing Machinery.
- Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020a. [Pre-training via paraphrasing](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 18470–18481. Curran Associates, Inc.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020c. [Retrieval-augmented generation for knowledge-intensive nlp tasks](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.
- Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and Yongbin Li. 2024a. [Evocodebench: An evolving code generation benchmark with domain-specific evaluations](#). In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. [Starcoder: may the source be with you!](#) *Preprint*, arXiv:2305.06161.
- Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Yichun Yin, Hao Zhang, Yong Liu, Yasheng Wang, and Ruiming Tang. 2024b. [Coir: A comprehensive benchmark for code information retrieval models](#). *Preprint*, arXiv:2407.02883.
- Dianshu Liao, Shidong Pan, Xiaoyu Sun, Xiaoxue Ren, Qing Huang, Zhenchang Xing, Huan Jin, and Qinying Li. 2024. [A3-codgen: A repository-level code generation framework for code reuse with local-aware, global-aware, and third-party-library-aware](#). *Preprint*, arXiv:2312.05772.
- Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. [Repubench: Benchmarking repository-level code auto-completion systems](#). In *The Twelfth International Conference on Learning Representations*.
- Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. 2025. [CodexGraph: Bridging large language models and code repositories via code graph databases](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 142–160, Albuquerque, New Mexico. Association for Computational Linguistics.
- 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. 2021. [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)*.
- Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. [Sparse, dense, and attentional representations for text retrieval](#). *Transactions of the Association for Computational Linguistics*, 9:329–345.
- Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2024. [How to understand whole software repository?](#) *ArXiv*, abs/2406.01422.
- Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. 2024. Repograph: Enhancing ai software engineering with repository-level code graph. *arXiv preprint arXiv:2410.14684*.
- Dragomir R. Radev, Hong Qi, Harris Wu, and Weiguo Fan. 2002. [Evaluating web-based question answering systems](#). In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain. European Language Resources Association (ELRA).
- Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. 2025. [Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents](#). *Preprint*, arXiv:2504.08703.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-bert: Sentence embeddings using siamese bert-networks](#). In *Conference on Empirical Methods in Natural Language Processing*.
- Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. 2021. [Contrastive learning with](#)



- hard negative samples. In *International Conference on Learning Representations*.
- Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. 2024. [Specrover: Code intent extraction via llms](#). *Preprint*, arXiv:2408.02232.
- Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. 2023. [Re-pofusion: Training code models to understand your repository](#). *Preprint*, arXiv:2306.10998.
- Kihyuk Sohn. 2016a. [Improved deep metric learning with multi-class n-pair loss objective](#). In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Kihyuk Sohn. 2016b. [Improved deep metric learning with multi-class n-pair loss objective](#). In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2025. [CoRNStack: High-quality contrastive data for better code retrieval and reranking](#). In *The Thirteenth International Conference on Learning Representations*.
- Chongyang Tao, Tao Shen, Shen Gao, Junshuo Zhang, Zhen Li, Zhengwei Tao, and Shuai Ma. 2024. LLMs are also effective embedding models: An in-depth overview. *arXiv preprint arXiv:2412.12591*.
- Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. [Improvements to bm25 and language models examined](#). *Proceedings of the 19th Australasian Document Computing Symposium*.
- Charalampos E Tsourakakis, Michael Mitzenmacher, Kasper Green Larsen, Jarosław Błasiok, Ben Lawson, Preetum Nakkiran, and Vasileios Nakos. 2017. Predicting positive and negative links with noisy queries: Theory & practice. *arXiv preprint arXiv:1709.07308*.
- Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. [Representation learning with contrastive predictive coding](#). *ArXiv*, abs/1807.03748.
- Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2024. [Coderag-bench: Can retrieval augment code generation?](#) *Preprint*, arXiv:2406.14497.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint*.
- Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. [Approximate nearest neighbor negative contrastive learning for dense text retrieval](#). In *International Conference on Learning Representations*.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024a. [Qwen2 technical report](#). *Preprint*, arXiv:2407.10671.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024b. [Swe-agent: Agent-computer interfaces enable automated software engineering](#). *Preprint*, arXiv:2405.15793.
- Dejiao Zhang, Wasi Uddin Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. 2024a. [CODE REPRESENTATION LEARNING AT SCALE](#). In *The Twelfth International Conference on Learning Representations*.
- Fengji Zhang, B. Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023a. [Repocoder: Repository-level code completion through iterative retrieval and generation](#). In *Conference on Empirical Methods in Natural Language Processing*.
- Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023b. [Repocoder: Repository-level code completion through iterative retrieval and generation](#). In *The 2023 Conference on Empirical Methods in Natural Language Processing*.
- Shuai Zhang, Wang Lijie, Xinyan Xiao, and Hua Wu. 2022. [Syntax-guided contrastive learning for pre-trained language model](#). In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2430–2440, Dublin, Ireland. Association for Computational Linguistics.
- Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024b. [Autocoderover: Autonomous program improvement](#). *Preprint*, arXiv:2404.05427.
- Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. 2024. [Large language models for information retrieval: A survey](#). *Preprint*, arXiv:2308.07107.



## Appendix A Related work

Here, we present some works related to ours.

**Benchmarks for code retrieval** The scope of standard benchmarks for semantic matching or code search between a natural language query and code such as CoSQA (Huang et al., 2021), CodeSearchNet (Husain et al., 2019), AdvTest (Lu et al., 2021), CodeXGLUE (Lu et al., 2021) are based on docstrings and corresponding code. The repository-level retrieval benchmarks such as RepoEval (Zhang et al., 2023a), Repobench (Liu et al., 2024), CodeRAG-Bench (Wang et al., 2024), EvoCodeBench (Li et al., 2024a) are typically used for code completion tasks where they do not have natural language queries or the retrieval is only for context and not editing. CoIR (Li et al., 2024b) is a recent benchmark that curated a collection of multiple datasets for various scenarios, including text-to-code, code-to-text, code-to-code retrieval, thus going beyond our target setting. Therefore, we focus our work on real-world GitHub repositories with natural language queries such as SWE-bench (Jimenez et al., 2024) and Long Code Arena for bug localisation (Bogomolov et al., 2024). CoRNStack (Suresh et al., 2025) is the another dataset that is relevant to us, however this is a concurrent work to ours and we do not experiment with it.

**Models for code-editing retrieval** Generative Large Language Models (LLMs) are showing improvements on many code related tasks (Li et al., 2023; Guo et al., 2024; Yang et al., 2024a). BehnamGhader et al. (2024) proposes LLM2Vec which allows LLMs to be used for dense retrieval. LLM2Vec comes at a cost of a much higher parameter count for the model, making it slower during inference, more costly to store embeddings, and challenging to fine-tune. Xia et al. (2024) propose a simple LLM-only framework to code-editing. Their approach prompts the model to localise, repair, and validate their solution. Most current approaches use agents which allows an LLM to interact with the repository through the use of tools such as reading, editing files, and running bash commands (Yang et al., 2024b). A notable LLM-agent AutocodeRover is provided with specific code search APIs which iteratively retrieve code context and locate bugs (Zhang et al., 2024b). Further improvements are seen by SpecRover which generates summaries and feedback messages

during agent steps (Ruan et al., 2024). To improve repository-level navigation, Ma et al. (2024) propose an agent RepoUnderstander that condenses the codebase into a knowledge graph and exploit the structure of the repository using Monte Carlo tree search. Liu et al. (2025) parses a repository into code entities and establishes relationships between them through a dataflow analysis, forming a repo-specific context graph. This is shown to improve code completion accuracy. Liu et al. (2025) integrate LLM agents with graph database interfaces extracted from code repositories. By leveraging the structural properties of graph databases and the flexibility of the graph query language, CodexGraph. RepoGraph (Ouyang et al., 2024) constructs a graph of code lines, with the nodes being code lines that capture definition-reference dependencies. These works are orthogonal ways of approaching the code retrieval problem through prompting LLMs.

**Representation learning for code** Representation learning for programming languages has benefited many downstream applications. Different techniques have been applied to learning representation such as masked language modelling (MLM) (Feng et al., 2020; Li et al., 2023), next token prediction (Kanade et al., 2020; Li et al., 2023) and contrastive learning (Guo et al., 2022; Zhang et al., 2022, 2024a). Representation learning seeks to induce meaningful (vector) embeddings of inputs, and is motivated by applications such as information retrieval via semantic search (Reimers and Gurevych, 2019; Izacard et al., 2021; Zhang et al., 2024a). Successful application of unsupervised contrastive learning leverages text-code pairs, mined from docstrings (Husain et al., 2019; Guo et al., 2022; Zhang et al., 2022, 2024a). These works involve models that semantically align the embeddings of code to its natural language description. However, they do not consider the alignment or abstraction required for retrieval for code-editing queries at the repository-level. Concurrent to our work, CoCoMic (Ding et al., 2024) shows that including relevant cross-file context based on import statements significantly improves retrieval.

**Repository-level feature learning** The standard loss for code retrieval is contrastive multi-class N-pair loss or InfoNCE (Sohn, 2016b; van den Oord et al., 2018; Chen et al., 2020). This loss maximises similarity between positive pairs while reducing similarity to all other pairs. This loss is

used in many retrieval or semantic search applications (Husain et al., 2019; Karpukhin et al., 2020; Zhang et al., 2024a). Hard-negative mining selects negative examples that differ from the anchor but have similar embeddings, making them the most challenging for the model to distinguish (Robinson et al., 2021). Supervised contrastive learning with hard-negatives has been shown to generally improve retrieval performance (Karpukhin et al., 2020; Lewis et al., 2020a,c). However, Xiong et al. (2021) argues that in-batch negatives are unlikely to be hard negatives when the mini-batch size is far smaller than the corpus size and when only a few negatives are informative. This has been shown to produce suboptimal training signals for dense passage retrieval across independent documents. A solution is to use negatives from lexical models such as BM25 during training (Karpukhin et al., 2020; Gao et al., 2021; Luan et al., 2021). Similarly, for code retrieval across independent repositories, we reduce in-batch negatives in favour of sampled hard negatives within the instance repository. Our formulation closely resembles standard maximum likelihood estimation. For each repository, the likelihood function is modeled as a categorical distribution over pairs, each consisting of the query and a repository code chunk. To reduce computational complexity, we approximate the normalisation factor by sampling a number of ‘negative’ pairs. Our training loss is then computed by averaging across all repositories. On the architecture side, we note that other kinds of fusion like feature fusion have been explored in literature in other contexts (Günther et al., 2024), however for this work, we limit ourselves to input string concatenation for its ease of implementation.

## Appendix B Chunking methodology

### B.1 Representing file & class information

We form *chunks* of code by splitting each file into its constituent classes, methods and functions as shown in Figure 1.

**File hierarchy:** These code chunks do not contain any information about their locality in the code repository. To add this information, we insert the file path at the beginning of all code chunks. This is an important bias for code-editing retrieval as the problem statements  $q$ ’s may often contain relevant file paths. For instance, on SWE-bench Verified (Jimenez et al., 2024), 26% of the problem statements contain the path of at least one ground

truth file (i.e. one file that needs to be edited) (see Table 4). Empirically we verify the efficacy of this representation by showing improved retrieval performance and visualising the attention maps of a trained retrieval model in Appendix I.

**Class representation:** Similarly, we preserve the class hierarchy within the chunks. We represent a class by its documentation string, its constructor and declaration of class methods. Each method is represented by including the class it belongs to.

## Appendix C Metric Definitions

We measure the performance of our model using standard retrieval metrics: recall@ $k$  and mean reciprocal rank (MRR). We report recall at both a file and a chunk level. We compute file recall by retrieving  $k$  chunks and take the files those chunks have been extracted from. A similar process is done to the ground truth chunks as well.

$$\text{Recall@}k = \frac{1}{N} \sum_{i=1}^N \frac{|\mathcal{C}_{F_i} \cap \mathcal{C}_i^*|}{|\mathcal{C}_i^*|} \quad (4)$$

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{rank}(\mathcal{C}_i^*, \mathcal{C}_{F_i})} \quad (5)$$

Here  $\text{rank}(\cdot, \cdot)$  refers to the rank of the first element that is common to both the arguments. This metric measures the minimum value of  $k$  needed to retrieve at least one correct code chunk, and is also known as the First Answer Reciprocal Rank (Radev et al., 2002).

## Appendix D Dataset statistics

We report relevant dataset statistics in Table 3 and Table 4. If an instance does not have at least one modified function, class or method of class, we discard it for the purpose of computing reported statistics. For all token calculations we use the SentencePiece (SP) tokenizer employed by CodeSage (Zhang et al., 2024a). We extract ground truth (GT) files and chunks using the ground truth patch for each instance, namely a file is in the ground truth if it is edited in the patch, and the same for other chunks. Overall, training instances (SWE-bench train) have slightly larger ground truth sets, comprising on average 5 chunks, than test instances (SWE-bench verified and LCA). File overlap (GT file overlap in Table 4) is overall high across datasets, suggesting that typically changes are restricted to a single or a small number of files.

Finally, more than a quarter of queries in SWE-bench and more than a third in LCA of queries contain the path of at least one file to edit (GT file in query, Table 4).

## Appendix E Baseline Model Details

We report in Table 6 various statistics about the models and methods we choose as baselines. Given the trade-offs between the performance and resource requirements, we choose CodeSage S as our base model to build CoRet upon.

## Appendix F Baseline performance

We report in Figure 4 the performance of various models and methods on SWE-bench Verified (before fine-tuning).

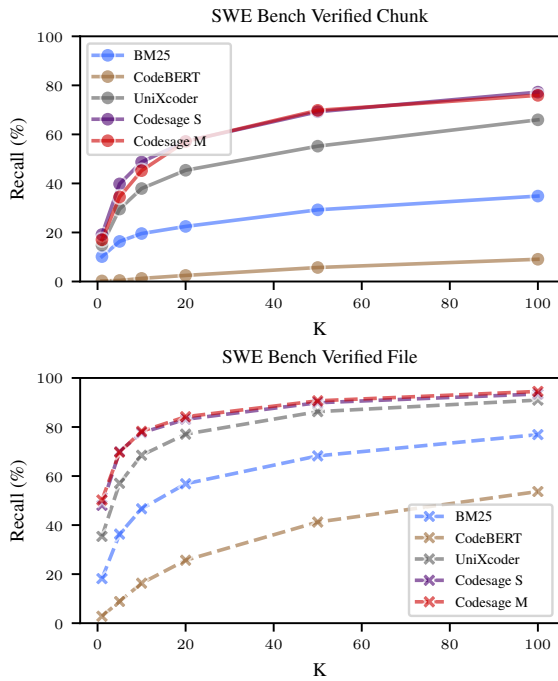


Figure 4: Performance of various models on SWE-bench Verified before fine-tuning. Top: chunk level recall; bottom: file-level recall. It is evident that the modern encoder models like the ones from the CodeSage family perform substantially better than other baselines. However, those models reach acceptable performance (say around 80% chunk recall) only when retrieving a large number of chunks ( $k \geq 50$ ). This motivates our need to train models that are specific for code retrieval.

## Appendix G Fine-tuning details

All experiments use the model CodeSage small<sup>1</sup> (Zhang et al., 2024a) for comparability and

<sup>1</sup>We use the initial version <https://huggingface.co/codesage/codesage-small> as the second version was re-

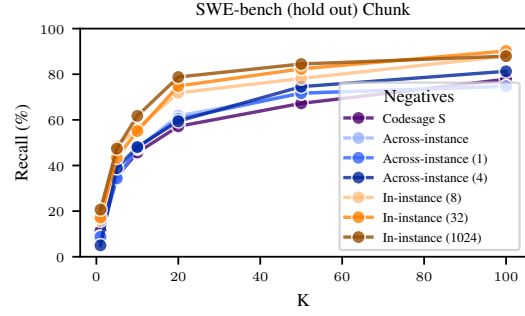


Figure 5: The influence of number of negatives  $|B|$  and their source.

reproducibility. This Transformer encoder has 6 layers with 8 attention heads per layer. The size for the word embedding vectors and model projections are 1024, feed-forward dimensions are 4096 which leads to models of approximately 130 million trainable parameters. We use bf16-mixed precision to reduce the memory footprint. The model was pre-trained on natural language and code tokenized with a Sentence Piece Tokenizer with a vocabulary of approximately 50K tokens. Each fine-tuning experiment takes approximately 24 hours to run on 8 NVIDIA A10G Tensor Core GPUs. During fine-tuning we use the hyperparameters reported in Table 7.

We choose the optimiser hyperparameters to increase stability and remove the need of initial learning rate warm up. Learning rate was selected over the grid

$$\{1e^{-3}, 5e^{-4}, 1e^{-4}, 5e^{-5}, 1e^{-5}, 5e^{-6}, 1e^{-6}\}$$

The batch size is 1 instance per GPU over 8 GPUs with gradient accumulation of 32; namely, we use an effective batch size of 256 instances. For each instance we sample 1024 (in-repository) negatives during training.

## Appendix H Choice of negative samples in Equation (2) influences the performance

In Figure 5, we show the impact of choice of negatives. Karpukhin et al. (2020) propose including hard-negative chunks from BM25 which we select from the same instance – we found marginal improvement when considering a single BM25 negative. In summary, in-instance negatives show a clear advantage, and including more negatives is advantageous for the performance. These results

leased after completion of the project.

Dataset		Instances	Files	Chunks	Chunk-level			Query
					GT	Lines	Tokens	Tokens
SWE-bench	Train	8349	263 <sub>(269)</sub>	3628 <sub>(3150)</sub>	5 <sub>(18)</sub>	20 <sub>(6)</sub>	198 <sub>(60)</sub>	631 <sub>(1974)</sub>
SWE-bench	Verified	435	420 <sub>(222)</sub>	8551 <sub>(6118)</sub>	2 <sub>(3)</sub>	19 <sub>(7)</sub>	195 <sub>(84)</sub>	539 <sub>(687)</sub>
LCA	Test	34	210 <sub>(238)</sub>	3243 <sub>(5010)</sub>	4 <sub>(5)</sub>	19 <sub>(8)</sub>	206 <sub>(142)</sub>	911 <sub>(878)</sub>

Table 3: General dataset statistics for SWE-bench dataset. **GT** is the ground truth number of chunks that are edited, **Lines** the average number of lines and **Tokens** the average number of tokens (with SP tokenizer) edited. Standard deviation are reported in brackets.

Dataset		Chunks per File	Files per GT	GT file overlap	GT file in query
SWE-bench	Train	17 <sub>(8)</sub>	1.97 <sub>(3.06)</sub>	0.83	0.28
SWE-bench	Verified	22 <sub>(9)</sub>	1.24 <sub>(1.06)</sub>	0.82	0.26
LCA	Test	16 <sub>(7)</sub>	2.4 <sub>(2.24)</sub>	0.70	0.38

Table 4: **GT file overlap**: On instances where there are multiple chunks to edit, we report the empirical probability that at least one file is in common between the chunks to edit. **GT file in query** is the empirical probability that the query contains at least one file path of the ground truth files to edit.

have been computed on a hold-out set from the SWE-bench train dataset.

## Appendix I Repository hierarchy experiment

In this experiment we show that including the file path to the chunks representation is important for the retrieval task and aids in maintaining the repository hierarchy. In Table 2, we show the retrieval performance of models tested with and without including the file path. The BM25 and not-fine-tuned CodeSage model achieve minor improvements in retrieval performance, indicating that fine-tuning is crucial for the model to make use of the added information.

We also investigate to what extent a model trained with file paths rely on them. In Table 2 in the main paper, we show the retrieval performance drop of a model trained with file paths once they are removed. We further analyse this by showing the attention plots of a model trained with file paths when given a query, visualised in Figure 6. For this, we consider a repository from the SWE-bench train subset which was not used in our training set (it is a validation example) and contains the correct file path in the query. We select the second from the last layer of the model and averaged over all attention heads. Figure 6 and Figure 7 shows that the model attends to the file path both for queries and code chunks and highlights that the model has learned to find and leverage the path information

in a natural language query.

## Appendix J License of artefacts used

SWE-bench (Jimenez et al., 2024) dataset packages multiple repositories that are based on BSD 3-Clause, MIT, Apache-2.0, Custom (based on BSD-2 and BSD-3), GPLv2 licenses. The exact details are available in the original paper in Table 12. The LCA dataset is released under Apache-2, with its constituent data point from repositories with MIT and Apache 2.0 licenses.

The code for training the models for CodeBERT and GraphCodeBERT is released under a MIT license at <https://github.com/microsoft/CodeBERT/blob/master/LICENSE> and their weights are released under with no license specified at <https://huggingface.co/microsoft/codebert-base> and <https://huggingface.co/microsoft/graphcodebert-base> respectively. UniXcoder’s weights are released under Apache 2.0 at <https://huggingface.co/microsoft/unixcoder-base>. Our primary baselines, the CodeSage family, have been released under a permissive Apache 2.0 license at <https://huggingface.co/codesage/codesage-small>.



Dataset		Calls	GT call overlap
SWE-bench	Train	1.40 <sub>(0.70)</sub>	0.46
SWE-bench	Verified	1.92 <sub>(2.22)</sub>	0.38
SWE-bench	Lite	2.37 <sub>(2.90)</sub>	0.16
LCA	Test	1.15 <sub>(0.49)</sub>	0.40

Table 5: **GT call overlap**: When there are multiple chunks to edit what is the probability of at least 1 chunk being connected by the call graph. The standard deviation is represented in the brackets.

Model	Parms	Tok	Length	Encoding
BM25	-	words	-	-
CodeBERT	125M	BPE	512	0.5min
GraphCodeBERT	125M	BPE	512	0.5min
UniXcoder	125M	BPE	512	0.5min
CodeSage S	130M	SP	1024	1.5min
CodeSage M	400M	SP	1024	5min
CodeSage L	1.3B	SP	1024	22.5min

Table 6: Statistics about baseline models and methods. Parms: number of model parameters; Tok: tokenizer with SentencePiece (SP), Byte-Pair Encoding (BPE); Length: maximum number input tokens (maximum context length); Encoding the approximate inference time in minutes per 10K chunks, on a single GPU. The model we chose for fine-tuning (CodeSage S) is highlighted. Additionally, the models have been released under permissive licenses, or their official huggingface repositories do not have a license specific.

Hyperparameters	Full	Late Fusion
Optimiser	RAdam	RAdam
Learning rate	$5e^{-4}$	$5e^{-4}$
Scheduler	cosine decay	cosine decay
Batch size	8	8
Gradient acc.	32	32
Negatives	1024	1024
Epochs	4	8

Table 7: Hyperparameters for fine-tuning

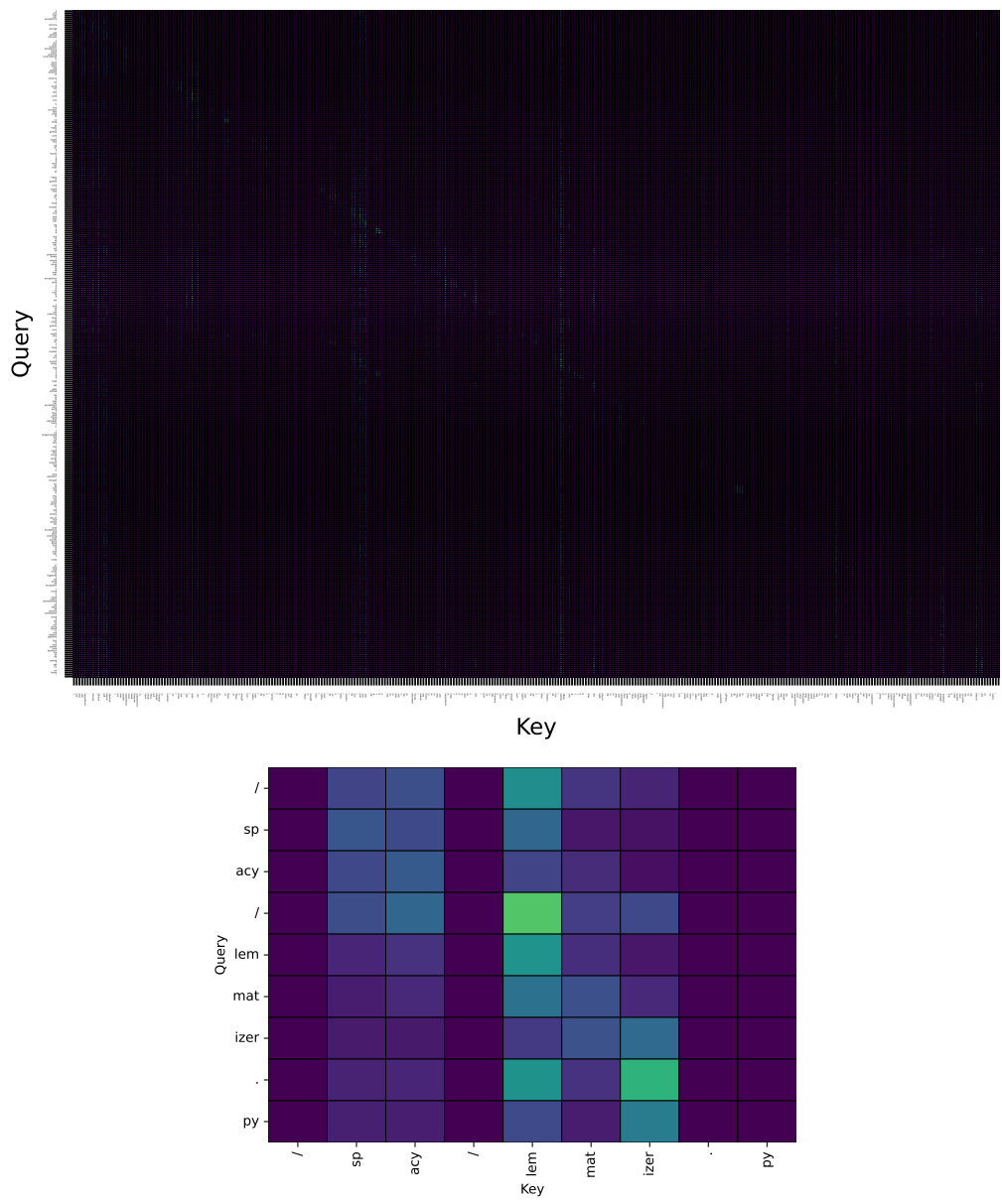


Figure 6: Attention map: query (problem statement) containing file path. Top: full average attention map (best viewed on a screen); bottom: zoom-in portion of the attention map containing the file path. Best viewed in colour on a digital display

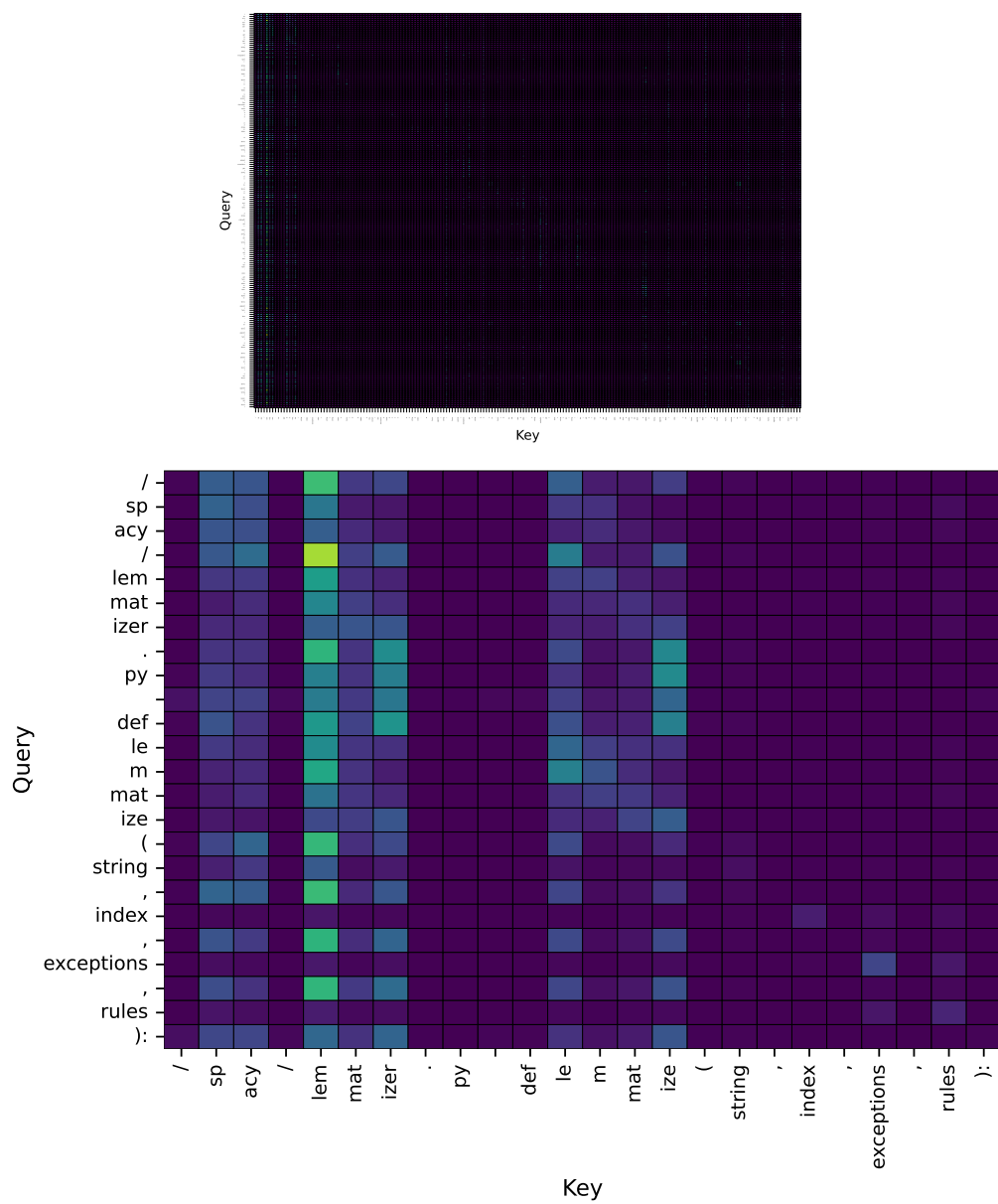


Figure 7: Attention map: code chunk containing file path. Top: full average attention map (best viewed on a screen); bottom: zoom-in portion of the attention map containing the file path. Best viewed in colour on a digital display