

On the Challenges and Opportunities of Learned Sparse Retrieval for Code

Simon Lupart*

University of Amsterdam
Amsterdam, The Netherlands

Maxime Louis

Naver Labs Europe
Meylan, France

Thibault Formal

Naver Labs Europe
Meylan, France

Hervé Déjean

Naver Labs Europe
Meylan, France

Stéphane Clinchant

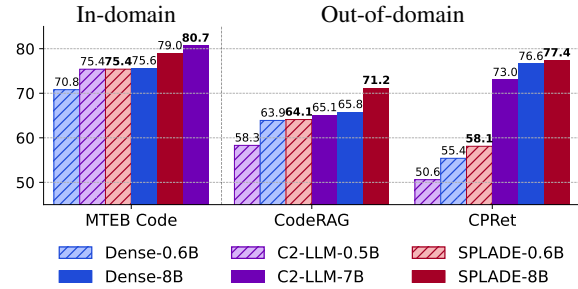
Naver Labs Europe
Meylan, France

Abstract

Retrieval over large codebases is a key component of modern LLM-based software engineering systems. Existing approaches predominantly rely on dense embedding models, while learned sparse retrieval (LSR) remains largely unexplored for code. However, applying sparse retrieval to code is challenging due to subword fragmentation, the semantic gap between natural-language queries and code, the diversity of programming languages and retrieval tasks, and the length of code documents, which can harm sparsity and latency. We introduce SPLADE-Code, the first large-scale family of learned sparse retrieval models specialized for code retrieval (600M–8B parameters). Despite a lightweight one-stage training pipeline, SPLADE-Code achieves state-of-the-art performance among retrievers under 1B parameters (75.4 on MTEB Code) and competitive results at larger scales (79.0 with 8B). We show that learned expansion tokens are critical to bridge lexical and semantic matching, and provide a latency analysis showing that LSR enables sub-millisecond retrieval on a 1M-passage collection with little effectiveness loss.

1 Introduction

Large language models (LLMs) have recently achieved strong performance on a wide range of coding tasks (Jiang et al., 2024; Hui et al., 2024; Khanfir et al., 2022; Li et al., 2023; OLMo Team et al., 2025). Beyond standalone code generation, they are increasingly integrated into agentic frameworks (Anthropic, 2026; Mistral AI, 2026), where they interact with tools, repositories, and documentation to support software development workflows (Santos et al., 2025; Yang et al., 2025). In these environments, retrieval plays a central role: models must explore large codebases, reuse existing components, and access external libraries or



Example Code and LSR Representation:

```
def f(n):  
    if n <= 1:  
        return n  
    return f(n-1) + f(n-2)  
  
x = int(input())  
print(f(x))
```

Token	Weight
Fibonacci	2.45
fib	2.22
recursive	2.12
input	1.99
f	1.90
Python	1.51

Figure 1: (Top) Performance in nDCG@10 of SPLADE-Code and dense models on code retrieval benchmarks.¹(Bottom) Example from CodeFeedback-ST. Only a few vocabulary dimensions receive non-zero weights, covering both *semantic* terms (e.g., Fibonacci, Python, recursive) and *lexical* tokens (e.g., input, f).

issue trackers such as GitHub (Wang et al., 2025). Consequently, accurate and efficient code retrieval has become a fundamental building block of modern AI-driven software engineering systems. To the best of our knowledge, existing neural code retrieval approaches predominantly rely on *dense* embedding models.

Opportunities of LSR for code. Learned sparse retrieval (LSR) models, which represent queries and documents as sparse high-dimensional bag-of-words vectors, may offer a compelling alternative for code retrieval. First, they often exhibit stronger generalization properties, being less sensitive to domain shifts and retrieval tasks (Formal et al., 2022; Lassance et al., 2024b). Second, they rely on

*Corresponding author: s.c.lupart@uva.nl | Work done during internship at Naver Labs Europe.

¹at the time of the submission, C2-LLM is the top 1 open source model on MTEB Code for 0.5B and 7B.

inverted indexes, which provide low latency (Tonello et al., 2018; Bruch et al., 2024) and easy integration into existing pipelines, while also yielding more interpretable matching signals (Nguyen et al., 2023). Finally, they offer a complementary representation to dense, making them useful in hybrid systems (Lin and Lin, 2023; Shen et al., 2023).

Challenges of LSR. However, building sparse retrieval models for code comes with its own challenges. (1) Code retrieval is heterogeneous: queries and documents may consist of natural language, code, or both, and code may come from different programming languages. As a result, the task requires strong text-to-code and code-to-code semantic alignment, akin to multilingual retrieval where semantics across languages has been challenging for LSR models (Formal et al., 2026; Nguyen et al., 2026). (2) Second, most LSR models build sparse representations by projecting embeddings onto the fixed vocabulary of their backbone LLM (Formal et al., 2021), exposing them to fragmented subword splits that can affect their representations. These issues are particularly prominent on code (Li et al., 2025c). (3) Finally, LSR models are sensitive to the long input lengths. Since token activations accumulate across positions, longer inputs typically produce denser representations, increasing both index size and query latency. While this is manageable for short natural-language documents (Lassance and Clinchant, 2022), code retrieval often involves long files or multi-function code snippets: even a classical BM25 incurs non-negligible latency (50 ms on a 1M collection in our setting). Without careful control of sparsity, LSR models may suffer from degraded efficiency at scale, making them impractical.

In this paper, we show how to tackle those challenges, achieving competitive results in effectiveness and efficiency for code retrieval. Our main contributions are as follows:

- We introduce SPLADE-Code, the first large-scale family of learned sparse retrieval models specialized for code, ranging from 600M to 8B parameters². SPLADE-Code models can be trained with a lightweight one-stage training pipeline while still achieving state-of-the-art performance among retrievers under 1B parameters (75.4 on MTEB Code with 0.6B) and strong results at larger scales (79.0 with 8B). In line

²Models weights are available at [naver/splade-code-0.6B](https://naver.com/splade-code-0.6B), [naver/splade-code-8B](https://naver.com/splade-code-8B)

with the typical behavior of sparse retrieval models, we observe strong generalization to out-of-domain settings, as shown in Figure 1 (top).

- We show that high performance critically relies on expansion tokens (i.e., tokens not present in the input), and that learned sparse representations combine strong exact lexical matching with robust semantic abstraction: example shown in Figure 1 (bottom).
- We study the search latency of LSR models for code, to evaluate their practicality in real-world settings. We achieve below 1 ms per query on the CodeSearchNet 1M passage collection using optimized LSR inverted-index retrieval, with only marginal ranking loss.

2 Related Work

Code retrieval tasks. Code retrieval has recently attracted a lot of attention, leading to the development of several benchmarks and tasks (Husain et al., 2019; Suresh et al., 2025). Applications cover Text-to-Code, Code-to-Text, Code-to-Code and Hybrid retrieval settings, including tasks such as bug fixing, code summarization, code completion, code translation, and programming contest problem solving, all framed as retrieval tasks (Li et al., 2025b). Current benchmarks also span an increasing number of programming languages, e.g. CPRet with more than 20 languages (Deng et al., 2025), and domains, e.g. StackOverflow, GitHub Functions, Database, Code Instruction, Libraries (Li et al., 2025b; Wang et al., 2025). This requires retrieval models to be robust to domains and tasks, but also to have strong semantic capabilities.

Code dense embedding. A few recent works trained dense embedding models for code. One of the first, CodeXEmbed (Liu et al., 2025), proposed a multi-stage training LoRA using English text and code retrieval data, creating a specialized embedding model. CodeR-Pile (Li et al., 2025a) proposed to synthesize a large-scale training dataset with a code-specialized LLM, adding more than 47 tasks and 20 programming languages, thus improving performance on benchmarks. More recently C2LLM (Qin et al., 2025) proposed a different pooling attention to reduce the information bottleneck in the EOS token. All these models have resource-intensive multi-stage training pipelines. Nevertheless, alternative representations remain

largely unexplored for code retrieval; to the best of our knowledge, our work is the first to investigate learned sparse retrieval in this setting.

Learned sparse retrieval refers to a family of neural retrieval methods, that learns to assign sparse vocabulary-based term weights to queries and documents, enabling semantic search with the efficiency and interpretability of inverted-index retrieval (Formal et al., 2021; Mallia et al., 2021; Nguyen et al., 2023; Kong et al., 2023). A seminal work has been the SPLADE models, showing excellent generalization properties (Formal et al., 2021, 2022; Lassance et al., 2024b; Déjean et al., 2023). SPLADE models, being based on BERT architectures have been extended with LLMs and decoder architectures (Soares et al., 2023; Doshi et al., 2024; Qiao et al., 2025; Xu et al., 2025; Zeng et al., 2025; Ma et al., 2025), and also generalized to sparse auto-encoders (Formal et al., 2026). It has also been applied to cross-lingual and conversational search (Nguyen et al., 2026; Lupart et al., 2025). Because code retrieval requires modeling both deep semantics and long contexts, it naturally aligns with SPLADE’s ability to capture sparse rich semantic signals.

In addition, LSR enables the use of traditional inverted index structures to efficiently compute sparse dot-product scores (Tonello et al., 2018). Similar to dense retrieval, additional approximate nearest neighbors methods have been developed (Bruch et al., 2024), including posting list pruning (Lassance et al., 2023) and sequential pruning (Lassance et al., 2024a) that could be studied for the long contexts of code retrieval tasks.

3 Method

3.1 Learned Sparse Retrieval

Learned sparse retrieval (LSR) is a family of models that aim to create high-dimensional sparse representations, formally:

$$\mathbf{u} = (u_1, \dots, u_N) \in \mathbb{R}^N, \quad N \in \mathbb{N}^+ \quad (1)$$

such that $\|\mathbf{u}\|_0 \ll N$. These sparse representations are used to compute similarity scores using a dot product $s = \mathbf{u} \cdot \mathbf{v}$ between query and document representations.

Most high-performing LSR models are based on SPLADE (Formal et al., 2021). In SPLADE, sparse representations are obtained by projecting contextualized token representations onto the vocabulary

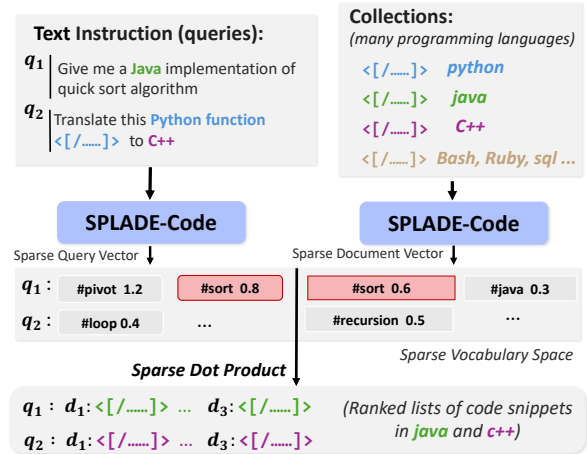


Figure 2: SPLADE-Code is a bi-encoder model for retrieval across programming languages (e.g. Text2Code, Code2Code). When queries and documents are in different programming languages, SPLADE-Code learns some shared dimensions in the representation space.

$\mathcal{V}$ of the language model (LM), using its embedding matrix. Hence, $N = |\mathcal{V}|$: the LM vocabulary fixes the dimensionality. For an input sequence of length n , sparse vectors are also aggregated across the sequence length with a max pooling:

$$\mathbf{u}_t = \max_{i \in \{1, \dots, n\}} \log(1 + \text{ReLU}(\mathbf{z}_{i,t})) \quad (2)$$

where $\mathbf{z}_{i,t}$ denotes the logit associated with token t at position i , after the projection to the vocabulary space. This max pooling operation selects, for each vocabulary term, the most salient activation across the sequence, yielding a sparse bag-of-words-like representation. The $\log(1 + \text{ReLU}(\cdot))$ activation ensures non-negative features and induces sparsity.

3.2 SPLADE for Code Retrieval

SPLADE-Code. As illustrated in Figure 2, SPLADE-Code is an LSR model for code retrieval. Unlike standard text retrieval, code retrieval requires the model to understand both textual query descriptions and structured source code. The model must thus capture semantic intent (e.g., sorting, parsing, recursion, database queries) while remaining robust to syntactic variation and programming language differences.

Training objective for code. To train the models for the code retrieval task, we use a Kullback-Leibler divergence (KLD) distillation objective (Kullback and Leibler, 1951; Lin et al., 2020). This objective is widely adopted for LSR and dense text retrieval, to distil relevance scores

from cross-encoder teacher models to retrieval models (Hofstätter et al., 2020; Formal et al., 2022). Formally, let q be a query and $\{d_1, \dots, d_{K+1}\}$ a candidate set of one positive document and K mined negatives in any programming language. The KLD loss is computed between the distributions of similarity scores obtained from the teacher and the student models, on the same set of (q, d) pairs:

$$\mathcal{L}_{\text{KLD}} = \mathcal{D}_{\text{KL}}(\mathbf{S}_T \parallel \mathbf{S}_S) \quad (3)$$

where $\mathbf{S}_S$ and $\mathbf{S}_T$ are the softmax of scores distribution of student and teacher models. This objective encourages the student model to match the teacher’s relative ranking over the candidate documents, rather than relying solely on hard relevance labels. Because queries and documents may appear in different programming languages or as textual descriptions, the objective encourages the model to project all inputs into shared sparse dimensions.

In addition, following SPLADE, and to encourage sparse activations, we also apply a FLOPs penalty within each batch. This regularizer penalizes terms that are frequently activated across the batch, which empirically leads to sparser posting lists and lower retrieval cost (Paria et al., 2020).

3.3 Model Configuration

Hyper-parameter details. We train our model on CoIR (see Table 1) with LoRA (Hu et al., 2022), a batch size of 256 with 7 negatives per query, and a max length of 512. We use a temperature of 300 in the KLD loss, with a learning rate of $1e-4$. The negatives were mined from retrieval made with Qwen3-Embedding-0.6B, and scored with Qwen3-Reranker-4B to be then distilled (Zhang et al., 2025). We select the negative between ranks 50 and 100. More details are provided in Appendix A.

Bi-directional attention. Since standard decoder-only models use causal (uni-directional) self-attention, directly applying them restricts each token’s representation to left context only, which is suboptimal for lexical matching (Xu et al., 2025). To address this, we enable bidirectional self-attention and pre-train them on MS MARCO passages (Nguyen et al., 2016) with Masked Next Token Prediction, following prior work on LSR (Qiao et al., 2025; Zeng et al., 2025).

Model merging is done with weighted spherical merging (Yang et al., 2024) from three checkpoints: (1) the base model after the first epoch, (2) one

Benchmark	Summary
CoIR (Code IR) (Li et al., 2025b)	10 datasets, 14 programming langs Train: 561-905K, Total: 2.2M Test queries: 180–53K
(in-domain)	Collection: 816–1M passages
MTEB-Code (Muennighoff et al., 2023) (build upon CoIR)	12 datasets: 10 CoIR + 2 new 14 programming langs Test queries: 180–53K
(in-domain)	Collection: 816–1M passages
CodeRAG-Bench (Wang et al., 2025)	6 datasets, 1 lang (Python) Test queries: 164–1K
(out-of-domain)	Collection: 237–40.9K passages
CPRet (Deng et al., 2025)	4 datasets, 20 programming langs Test queries: 168–10K
(out-of-domain)	Collection: 10K–41.6K passages

Table 1: Benchmark overview. Sizes are reported as min–max numbers of samples per dataset (See Appendix B for statistics and programming langs details).

model after a second epoch, (3) one model trained with a max context length of 1024 for one epoch. Small models (0.6B) also contain a fourth checkpoint trained with full finetuning (no LoRA).

4 Experiments

4.1 Benchmarks and Evaluation

We evaluate our models on various code retrieval benchmarks, containing different programming languages, tasks and characteristics. For *in-domain* evaluation, we rely on:

- CoIR (Li et al., 2025b): the biggest code retrieval benchmark, containing ten datasets in 14 code languages. It features 4 main tasks, Code2Code, Text2Code, Code2Text and Hybrids with train/test splits. All merged training datasets contain 2.2M training samples.
- MTEB-code (Muennighoff et al., 2023): built upon CoIR, with two new datasets: CodeEditSearch Retrieval and new variation of CodeSearchNet. We consider it in-domain since it is mainly composed of CoIR datasets.

In addition, we use two additional benchmarks for *out-of-domain* evaluation, namely:

- CodeRAG (Wang et al., 2025): a retrieval-augmented code generation benchmark, containing retrieval annotation on 8 coding tasks: basic programming, open-domain, and repository-level problems.
- CPRet (Deng et al., 2025): a benchmark containing 4 retrieval datasets focusing on problem-

Model	CoIR	MTEB Code	Code RAG	CPRet
Dense Retrieval				
Dense-0.6B	71.4	70.8	63.9	55.4
Dense-1.7B	73.4	73.4	<u>67.3</u>	65.8
Dense-8B	<u>76.0</u>	<u>75.6</u>	65.8	<u>76.6</u>
Lexical Retrieval (no expansion)				
BM25	45.8	47.9	57.7	18.6
SPLADE-lex.-8B	46.2	45.8	56.7	29.4
Learned Sparse Retrieval				
SPLADE-Code-0.6B	72.6	73.5	63.7	56.4
/vs Dense	(+1.2)	(+2.7)	(-0.2)	(+1.0)
SPLADE-Code-1.7B	74.3	75.1	64.9	62.5
/vs Dense	(+0.9)	(+1.7)	(-2.4)	(-3.3)
SPLADE-Code-8B	76.7	77.8	68.5	76.8
/vs Dense	(+0.7)	(+2.2)	(+2.7)	(+0.2)

Table 2: Controlled experiments comparing learned sparse retrieval, dense models, and lexical baselines using nDCG@10. All models are trained on the same data without checkpoint merging to ensure a fair comparison.

centric competitive programming with more than 20 programming languages.

We report the main retrieval metrics, using nDCG@10 (Järvelin and Kekäläinen, 2000) following common practice on the used benchmarks. This evaluates the top ranking of each retrieved ranked list using the relevance annotation from evaluated datasets. For the LSR retrieval we used the *seismic* library, which contains an efficient implementation of inverted index (Bruch et al., 2024), and report search latency (excluding query encoding).

Baselines. We compare SPLADE-Code against dense retrievers trained with the same pipeline on Qwen3-0.6B, 1.7B, and 8B, as well as top models from the MTEB Code leaderboard. These include C2-LLM (Qin et al., 2025), CodeXEmbed (Liu et al., 2025) and CodeR, aka bge-code-v1 (Li et al., 2025a), trained on CoIR and synthetic Qwen2.5Coder-32B data. We further compare SPLADE-Code to other LSR variants, including SPLADE-lexical, which uses token weighting without expansion, BM25, alternative LLM backbones (QwenCoder, Llama), and SPLARE, which learns sparse features via sparse autoencoders.

4.2 Main Results

LSR and dense in controlled experiments. Table 2 presents the performance of LSR and dense models on four code retrieval benchmarks under controlled experimental settings. All models are

trained on the CoIR training split using comparable hyperparameters and backbone architectures to ensure a fair comparison (cf Appendix A). Evaluation spans 22 datasets and covers more than 20 programming languages across a variety of subtasks.

Dense models achieve high performance overall, with nDCG@10 scores typically in the 0.70-0.80 range, with consistent improvements as the backbone model size increases, providing strong baselines. Yet, our results show that SPLADE-Code consistently matches or exceeds dense baselines across the four benchmarks. Compared to dense, SPLADE-Code-8B outperforms them in both in-domain and out-of-domain evaluation. These improvements also remain stable across different model scales.

In contrast, purely lexical approaches such as SPLADE-lexical perform poorly. These models lack the semantic expansion required to capture the meaning of code-related queries and documents. Similarly, BM25 struggles in this setting for the same reason. This behavior is expected, as code retrieval tasks are highly semantic, particularly for Text-to-Code and Code-to-Code retrieval across different programming languages.

In-domain performance comparison with top models. Table 3 compares SPLADE-Code with state-of-the-art dense retrievers. In this setting, we apply checkpoint merging to improve the LSR models, which yields gains of approximately 1-2 nDCG@10 points compared to the original SPLADE-Code models reported in Table 2. We also focus on the comparison with C2-LLM, which represents the closest experimental setting to ours (similar sizes, and both trained on CoIR using checkpoint merging), in contrast to CodeR-all, which was trained on 5.2M training samples (vs. 2.2M for all other models).

For smaller models, SPLADE-Code-0.6B shows state-of-the-art performance among systems below one billion parameters. In particular, it achieves comparable results to C2-LLM-0.5B, and ranks first on CoIR within this model size category. Now at larger scales, SPLADE-Code-8B achieves the best performance on three datasets (CodeSearchNet, CodeTrans-Contest, and Stack-Overflow-QA) among the systems considered, suggesting that certain code retrieval tasks benefit from LSR over dense retrieval. Compared with C2-LLM-7B and CodeR-all, SPLADE-Code-8B obtains particularly lower performance on CosQA and CodeTrans-

Model	Apps	CosQA	T2SQL	CSN [†]	CSN -COIR	CSN -CCR	Edit Ret. [†]	CodeTrans -CT	SO -DL	QA	Feedback -ST	-MT	Avg COIR	Avg MTEB
BM25	4.7	18.7	24.9	63.5	70.3	59.3	53.3	47.7	34.4	70.2	68.1	59.1	45.8	47.9
<i>SotA Dense Retrieval</i>														
Jina-v2-code	16.3	41.0	44.2	—	84.0	82.7	—	86.6	30.5	89.4	69.0	52.1	59.6	—
Dense-0.6B	51.8	33.3	71.1	77.5	85.8	92.0	58.7	86.2	32.6	91.5	80.2	89.4	71.4	70.8
Dense-8B	82.1	32.2	72.2	78.9	88.9	95.2	68.6	93.1	28.2	95.8	80.6	91.6	76.0	75.6
CodeR-all-1.5B	98.1	<u>46.7</u>	64.4	90.5	89.5	98.3	70.8	94.4	<u>46.1</u>	95.4	<u>90.6</u>	94.4	81.8	81.6
CodeR-coir-1.5B	81.5	46.6	64.2	—	89.1	97.8	—	94.4	46.3	95.1	89.6	93.2	79.8	—
CodeXEmb-2B	76.9	40.5	<u>78.4</u>	—	87.9	97.7	—	90.3	38.6	94.5	86.4	65.5	75.7	—
CodeXEmb-7B	85.4	42.5	78.9	—	<u>89.7</u>	<u>98.0</u>	—	<u>94.5</u>	40.5	<u>96.3</u>	87.5	68.8	78.2	—
C2-LLM-0.5B	61.0	38.3	74.1	89.2	86.7	96.3	71.4	84.3	34.0	89.4	88.6	92.3	74.5	75.4
C2-LLM-7B	<u>86.7</u>	39.8	75.8	91.1	89.8	97.9	81.5	92.5	34.1	94.9	90.7	<u>94.3</u>	79.7	<u>80.7</u>
<i>Learned Sparse Retrieval</i>														
SPLADE-Code-0.6B	66.5	35.4	74.3	90.5	86.2	91.4	68.4	90.9	31.4	93.3	84.7	92.0	74.6	75.4
/vs C2-LLM-0.5B	(+5.5)	(-2.9)	(+0.2)	(+1.3)	(-0.5)	(-4.9)	(-3.0)	(+6.6)	(-2.6)	(+3.9)	(-3.9)	(-0.3)	(+0.1)	(+0.0)
SPLADE-Code-8B	<u>86.7</u>	32.5	75.7	92.1	88.9	94.6	<u>77.1</u>	94.6	28.1	96.5	87.6	93.9	77.9	79.0
/vs C2-LLM-7B	(+0.0)	(-7.3)	(-0.1)	(+1.0)	(-0.9)	(-3.3)	(-4.4)	(+2.1)	(-6.0)	(+1.6)	(-3.1)	(-0.4)	(-1.8)	(-1.7)

Table 3: MTEB-Code retrieval results (nDCG@10). All results are reported with pruning (500, 1000). Datasets included in MTEB-Code but not in CoIR are marked with [†].

DL. This difference may be partially explained by dataset characteristics, as CosQA has been shown to exhibit strong annotation biases (Gong et al., 2024). Dense models may more easily adapt to such biases, whereas LSR models rely on vocabulary-based sparse representations, which may make fitting these dataset-specific patterns more challenging.

Generalization to out-of-domain benchmarks.

Table 4 reports a detailed comparison with top models on the out-of-domain CodeRAG Bench and CPRet benchmarks. In this setting, SPLADE-Code achieves higher performance than most models. In particular, it surpasses C2-LLM by 5.8 and 6.1 nDCG@10 points on CodeRAG Bench for both the 0.6B and 8B configurations, and by 7.5 and 4.4 points on CPRet, respectively. Datasets such as DS-1000, ODEX, T2C, and C2C illustrate the limitations of lexical-only retrieval. On these datasets, BM25 achieves nDCG@10 values typically below 10, where in contrast, SPLADE-Code has +30 nDCG points gains. This behavior suggests that LSR reduces lexical mismatch and captures semantic meaning between queries and code.

More broadly, the strong out-of-domain performance aligns with prior findings on SPLADE models, which have shown robust generalization across domains (Lassance et al., 2024b). Because sparse representations are grounded in the backbone vocabulary, they can remain effective even when queries and documents differ from the training distribution, as the model can still rely on lexical and shared semantic terms within the vocabulary space. Finally, SPLADE-Code remains competi-

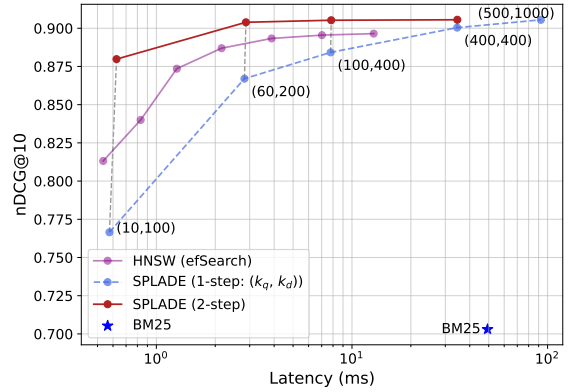


Figure 3: Effectiveness–efficiency trade-off on the CodeSearchNet 1M passage collection. The figure reports retrieval latency and effectiveness of SPLADE-Code-8B compared to the Dense-8B model. (k_q, k_d) indicates the pruning used on queries and documents for LSR.

tive with CodeR-all, which was trained on a much larger dataset. Since some evaluation datasets overlap with CodeR-all training data, its evaluation is not strictly out-of-domain. Despite this advantage, SPLADE-Code achieves comparable performance with a substantially lighter training setup.

4.3 Analysis

Effectiveness–Efficiency trade-off. Figure 3 shows the effectiveness–efficiency trade-off of SPLADE-Code, reporting retrieval latency in milliseconds per query against retrieval effectiveness (average latency on the 53k test queries). Overall, SPLADE-Code achieves efficient retrieval through pruning of the sparse representations. We evaluate a two-step SPLADE retrieval strategy, where an ini-

Model	CodeRAG-Bench							CPRet				
	Human Eval	MBPP	DS-1K	ODEX	Repo Eval	SWE Ben-L	Avg	T2C	C2C	P2Du	S2Fu	Avg
BM25	100.0	98.6	5.2	6.7	93.2	43.0	57.7	0.9	7.4	12.4	53.8	18.6
<i>SotA Dense Retrieval</i>												
Dense-0.6B	100.0	99.1	29.0	16.9	92.1	46.4	63.9	32.3	43.7	54.2	91.4	55.4
Dense-8B	100.0	98.9	30.3	26.4	89.0	50.4	65.8	64.3	74.2	72.5	<u>95.3</u>	76.6
CodeR-all-1.5B	100.0	<u>99.2</u>	<u>40.8</u>	36.1	93.1	67.4	72.8	75.1	85.4	58.4	94.7	78.3
CodeR-coir-1.5B	100.0	98.9	37.2	32.5	92.2	<u>64.6</u>	70.9	—	—	—	—	—
CodeXEmbed-2B	100.0	97.4	25.4	23.9	88.7	52.4	64.6	39.6	68.0	45.3	86.4	59.8
SFR-mistral-7B	100.0	99.0	19.3	<u>37.1</u>	83.8	62.7	67.0	22.2	50.9	31.9	69.4	43.6
C2-LLM-0.5B	99.8	98.6	18.1	21.0	67.6	44.6	58.3	38.1	50.6	31.3	82.4	50.6
C2-LLM-7B	100.0	99.1	34.7	31.2	72.3	53.2	65.1	67.8	75.6	54.6	94.0	73.0
<i>Learned Sparse Retrieval</i>												
SPLADE-Code-0.6B	100.0	<u>99.2</u>	21.1	29.1	95.8	39.5	64.1	42.4	47.6	49.7	92.6	58.1
/vs C2-LLM-0.5B	(+0.2)	(+0.6)	(+3.0)	(+8.1)	(+28.2)	(-5.1)	(+5.8)	(+4.3)	(-3.0)	(+18.4)	(+10.2)	(+7.5)
SPLADE-Code-8B	100.0	99.4	42.0	38.8	94.2	52.6	<u>71.2</u>	70.9	76.4	66.3	96.1	<u>77.4</u>
/vs C2-LLM-7B	(+0.0)	(+0.3)	(+7.3)	(+7.6)	(+21.9)	(-0.6)	(+6.1)	(+3.1)	(+0.8)	(+11.7)	(+2.1)	(+4.4)

Table 4: Out-of-domain retrieval performance on CodeRAG and CPRet (nDCG@10). All results are reported with pruning (1000, 1000).

tial stage applies aggressive pruning on query and document representations (Lassance et al., 2024a), e.g. (10, 100), to quickly retrieve a candidate set, followed by a second stage that refines the scores with a less aggressive pruning configuration, in practice (500, 1000). Because both stages rely on the same sparse representations, this approach preserves most of the retrieval effectiveness while substantially reducing latency. In contrast, BM25 exhibits high latency (49 ms per query³), due to the long average lengths of queries and documents in CodeSearchNet (594 and 156 average words respectively), which results in a large number of postings to process in the inverted index. For comparison, dense retrieval uses the HNSW algorithm (Malkov and Yashunin, 2020) implemented in FAISS (Douze et al., 2025), and all experiments are conducted on the same single AMD EPYC 7313 processor, while LSR uses the *Seismic* inverted index library.

Effect of the projection space. Table 5 compares LSR models built on different backbone LLMs, which define the projection space. Overall, performance is relatively stable across 7-8B backbones. In particular, generalist models such as Qwen3-8B, Llama3-8B, and code-specialized backbones like Qwen2.5Coder-7B yield nearly identical results across benchmarks. Comparing with smaller models, on Qwen2.5Coder-0.5B and Qwen2.5Coder-1.5B, results are also similar to those for models trained on Qwen3-0.6B and

Model Variant	CoIR	MTEB Code	Code RAG	CPRet
<i>SPLADE-Code w/ Generalist LLM</i>				
Qwen3-8B	76.7	77.8	68.5	76.8
Llama3-8B	<u>76.6</u>	<u>77.6</u>	<u>68.6</u>	72.4
<i>SPLADE-Code w/ Code-Specialized LLM</i>				
Qwen2.5Coder-0.5B	70.4	71.5	62.4	46.6
Qwen2.5Coder-1.5B	71.9	72.9	63.9	50.4
Qwen2.5Coder-7B	76.4	<u>77.6</u>	71.3	74.9
<i>SPLARE-Code w/ Sparse Auto-Encoder (Formal et al., 2026)</i>				
with general-domain SAE	76.0	77.1	62.3	67.8
with code-specific SAE	74.6	76.0	69.4	72.2

Table 5: Ablation comparison of LSR models with different LLM backbones. Results are reported without checkpoint merging for fair comparison.

Qwen3-1.7B. SPLADE-Code seems to be robust to the projection vocabulary. Finally, replacing vocabulary-based projection with learned sparse latent features, as in SPLARE (Formal et al., 2026), produces comparable results. This may be because the matching space remains largely grounded in English natural language, which acts as a shared pivot between queries and documents. As a result, differences in the projection vocabulary appear to have limited impact on retrieval effectiveness.

Training pipeline variants. Table 6 compares several training variants. Both the base setup and the instruction-tuned variant achieve similar performance, indicating that SPLADE-Code works well with or without instructions in this setting.

In contrast, intermediate English fine-tuning leads to lower effectiveness on both CoIR and

³using pyserini (Lin et al., 2021) BM25 inverted index.

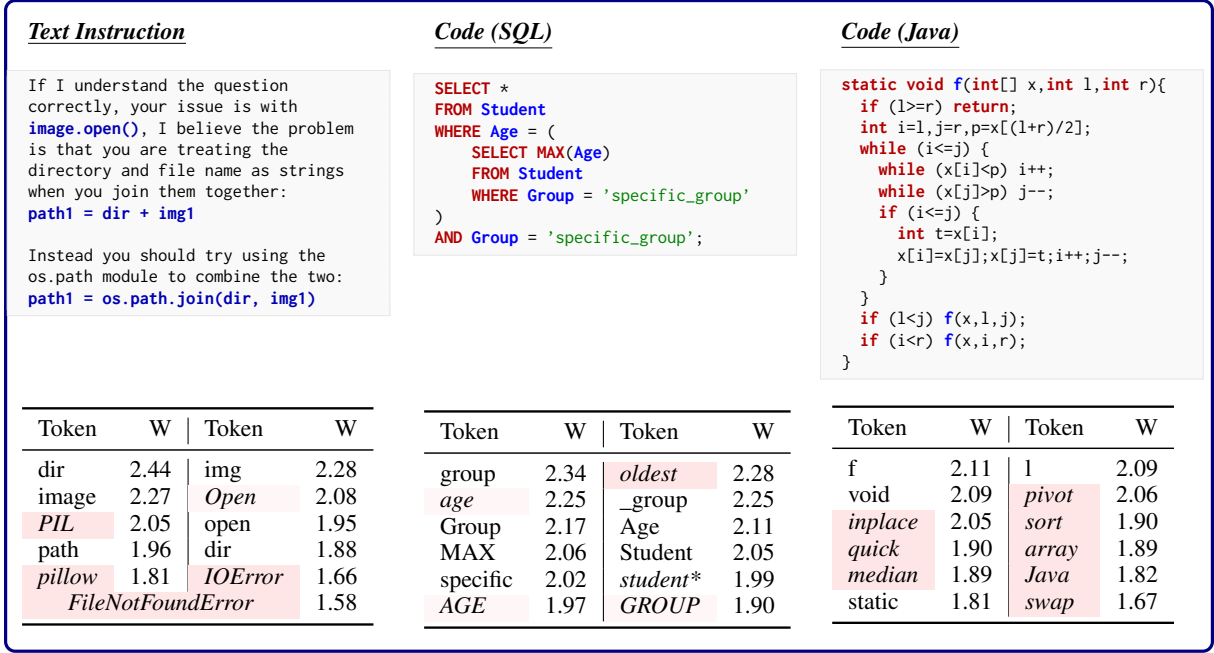


Figure 4: Examples of sparse vocabulary activations produced by SPLADE-Code-8B for different inputs. Highest-weighted **expansion tokens** in the representations capturing query intent, or algorithmic concepts highlighted.

Model Variant	CoIR	MTEB Code
SPLADE-Code-0.6B		
(Base)	72.6	73.5
(w/ Instructions)	73.0	73.4
(English→Code)	67.1	68.5
(Contrastive)	68.3	67.4

Table 6: Ablations on SPLADE-Code-0.6B. The default setup uses KL distillation without instructions or intermediate English fine-tuning. Results are reported without checkpoint merging for fair comparison.

MTEB Code. This differs from dense retrieval, where an intermediate text retrieval stage is often beneficial. A plausible explanation is that LSR already retains lexical matching capabilities through its sparse representations, reducing the need for an English intermediate stage before training on code.

Interpretability of the LSR representations. A key advantage of SPLADE-style LSR is interpretability: the representations are highly sparse, with only a few hundred active dimensions in a vocabulary-sized space, and each dimension can be directly mapped to a token. Figure 4 illustrates this behavior with three examples. Example 1 shows semantically meaningful expansion around a Python image-processing issue. Example 2 shows that the model emphasizes semantic concepts over generic SQL syntax. Example 3 shows that the model can

recover high-level algorithmic intent from code even when it is not explicitly stated, supporting matching between natural language descriptions and code across programming languages.

More broadly, the activated terms remain semantically coherent, and we do not observe obvious concept drift toward noisy or non-semantic tokens (Polyakov et al., 2026; Nguyen et al., 2024). Among the top-25 activated terms, roughly 35% come from the input itself and 65% are expansion terms, highlighting the central role of expansion in the learned sparse representation. For clarity in the examples provided, we also omitted a few lexical tokens with different capitalization or subword tokens from the top weights.

5 Conclusion

We introduced SPLADE-Code, the first learned sparse retrieval models specialized for code. Across multiple in-domain and out-of-domain benchmarks, SPLADE-Code achieves strong effectiveness with a lightweight single-stage training pipeline, showing that LSR is a viable alternative to dense retrieval for code search. Our analysis further shows that expansion terms are central to this performance, enabling sparse representations to bridge lexical matching and semantic abstraction while remaining interpretable. We also showed that SPLADE-Code can provide favorable

effectiveness–efficiency trade-offs, reaching below 1 ms latency with only limited effectiveness loss, and is interpretable.

Future work could integrate LSR more tightly into real-world agentic software engineering systems, where retrieval must jointly satisfy effectiveness, latency, and interpretability constraints.

Limitations

Our study has several limitations. First, while we evaluate SPLADE-Code across multiple benchmarks and both in-domain and out-of-domain settings, these datasets may not fully reflect the distribution of real developer queries (e.g., repository-specific naming conventions, evolving APIs, or interactive debugging contexts). Additional evaluation on more complex and realistic datasets, query logs and repository-level tasks would strengthen external validity. Then, we primarily study retriever-level effectiveness. In practical LLM-based software engineering systems, retrieval interacts with reranking, tool use, and generation. Measuring end-to-end impacts (e.g., downstream task success, faithfulness, and debugging outcomes) remains an important direction. Finally, a further challenge in comparing latency is that dense retrieval is often optimized for multi-CPU or GPU-based similarity search, while sparse inverted-index methods are typically evaluated as single-core systems designed to handle large query volumes in parallel, so the resulting measurements are not directly equivalent.

References

- Anthropic. 2026. Claude code. <https://claude.com/product/claude-code>.
- Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient inverted indexes for approximate retrieval over learned sparse representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 152–162.
- Hervé Déjean, Stéphane Clinchant, Carlos Lassance, Simon Lupart, and Thibault Formal. 2023. [Benchmarking middle-trained language models for neural search](#). In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’23, page 1848–1852, New York, NY, USA. Association for Computing Machinery.
- Han Deng, Yuan Meng, Shixiang Tang, Wanli Ouyang, and Xinzhu Ma. 2025. CPRet: A dataset, benchmark, and model for retrieval in competitive programming. *arXiv preprint arXiv:2505.12925*.
- Meet Doshi, Vishwajeet Kumar, Rudra Murthy, Vignesh P, and Jaydeep Sen. 2024. [Mistral-SPLADE: LLMs for better learned sparse retrieval](#). *Preprint*, arXiv:2408.11119.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The Faiss library. *IEEE Transactions on Big Data*.
- Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. [From distillation to hard negative sampling: Making sparse neural ir models more effective](#). In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’22, page 2353–2359, New York, NY, USA. Association for Computing Machinery.
- Thibault Formal, Maxime Louis, Hervé Déjean, and Stéphane Clinchant. 2026. [Learning retrieval models with sparse autoencoders](#). In *The Fourteenth International Conference on Learning Representations*.
- Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. [SPLADE: Sparse lexical and expansion model for first stage ranking](#). In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’21, page 2288–2292, New York, NY, USA. Association for Computing Machinery.
- Jing Gong, Yanghui Wu, Linxi Liang, Yanlin Wang, Jiachi Chen, Mingwei Liu, and Zibin Zheng. 2024. CoSQA+: Pioneering the multi-choice code search benchmark with test-driven agents. *arXiv preprint arXiv:2406.11589*.
- Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *International Conference on Learning Representations*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

- Kalervo Järvelin and Jaana Kekäläinen. 2000. [IR evaluation methods for retrieving highly relevant documents](#). In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 41–48, New York, NY, USA. ACM.
- Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*.
- Ahmed Khanfir, Matthieu Jimenez, Mike Papadakis, and Yves Le Traon. 2022. CodeBERT-NT: code naturalness via codebert. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pages 936–947. IEEE.
- Weize Kong, Jeffrey M. Dudek, Cheng Li, Mingyang Zhang, and Michael Bendersky. 2023. [SparseEmbed: Learning sparse lexical representations with contextual embeddings for retrieval](#). In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2399–2403, New York, NY, USA. Association for Computing Machinery.
- Solomon Kullback and R. A. Leibler. 1951. [On information and sufficiency](#). *Annals of Mathematical Statistics*, 22:79–86.
- Carlos Lassance and Stéphane Clinchant. 2022. [An efficiency study for SPLADE models](#). In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2220–2226, New York, NY, USA. Association for Computing Machinery.
- Carlos Lassance, Hervé Déjean, Stéphane Clinchant, and Nicola Tonellotto. 2024a. [Two-step SPLADE: Simple, efficient and effective approximation of splade](#). In *Advances in Information Retrieval: 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24–28, 2024, Proceedings, Part II*, page 349–363, Berlin, Heidelberg. Springer-Verlag.
- Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024b. [SPLADE-v3: New baselines for SPLADE](#). *arXiv preprint arXiv:2403.06789*.
- Carlos Lassance, Simon Lupart, Hervé Déjean, Stéphane Clinchant, and Nicola Tonellotto. 2023. [A static pruning study on sparse neural retrievers](#). In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1771–1775, New York, NY, USA. Association for Computing Machinery.
- Chaofan Li, Jianlyu Chen, Yingxia Shao, Defu Lian, and Zheng Liu. 2025a. [Towards a generalist code embedding model based on massive data synthesis](#). In *The Thirty-ninth Annual 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, and 1 others. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.
- Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Hao Zhang, Xinyi Dai, Yasheng Wang, and Ruiming Tang. 2025b. [CoIR: A comprehensive benchmark for code information retrieval models](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22074–22091, Vienna, Austria. Association for Computational Linguistics.
- Yinxi Li, Yuntian Deng, and Pengyu Nie. 2025c. TokDrift: When LLM speaks in subwords but code speaks in grammar. *arXiv preprint arXiv:2510.14972*.
- Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362.
- Sheng-Chieh Lin and Jimmy Lin. 2023. [A dense representation framework for lexical and semantic matching](#). *ACM Trans. Inf. Syst.*, 41(4).
- Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv preprint arXiv:2010.11386*.
- Ye Liu, Rui Meng, Shafiq Joty, silvio savarese, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2025. CodeXEmbed: A generalist embedding model family for multilingual and multi-task code retrieval. In *Second Conference on Language Modeling*.
- Simon Lupart, Mohammad Aliannejadi, and Evangelos Kanoulas. 2025. [DiSCo: LLM knowledge distillation for efficient sparse retrieval in conversational search](#). In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2025, Padua, Italy, July 13-18, 2025*, pages 9–19. ACM.
- Guangyuan Ma, Yongliang Ma, Xuanrui Gou, Zhenpeng Su, Ming Zhou, and Songlin Hu. 2025. [LightRetriever: A LLM-based hybrid retrieval architecture with 1000x faster query inference](#). *Preprint*, arXiv:2505.12260.
- Yu A. Malkov and D. A. Yashunin. 2020. [Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs](#). *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836.
- Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. [Learning passage impacts for inverted indexes](#). In *Proceedings of the 44th International ACM SIGIR Conference on Research and*

- Development in Information Retrieval*, SIGIR '21, page 1723–1727, New York, NY, USA. Association for Computing Machinery.
- Mistral AI. 2026. Mistral vibe. <https://mistral.ai/fr/products/vibe>.
- Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. **MTEB: Massive text embedding benchmark**. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.
- Thong Nguyen, Mariya Hendriksen, Andrew Yates, and Maarten de Rijke. 2024. Multimodal learned sparse retrieval with probabilistic expansion control. In *European Conference on Information Retrieval*, pages 448–464. Springer.
- Thong Nguyen, Yibin Lei, Jia-Huei Ju, Eugene Yang, and Andrew Yates. 2026. **MILCO: Learned sparse retrieval across languages via a multilingual connector**. In *The Fourteenth International Conference on Learning Representations*.
- Thong Nguyen, Sean MacAvaney, and Andrew Yates. 2023. A unified framework for learned sparse retrieval. In *European Conference on Information Retrieval*, pages 101–116. Springer.
- Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. **MS MARCO: A human generated machine reading comprehension dataset**. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain, December 9, 2016, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- OLMo Team, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, and 1 others. 2025. OLMo 3. *arXiv preprint arXiv:2512.13961*.
- Biswajit Paria, Chih-Kuan Yeh, Ian En-Hsu Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. 2020. **Minimizing flops to learn efficient sparse representations**. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Gregory Polyakov, Harris Scells, and Carsten Eickhoff. 2026. **Understanding wacky weights: A dissection of splade’s learned term importance**. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 3028–3038. ACM.
- Jingfen Qiao, Thong Nguyen, Evangelos Kanoulas, and Andrew Yates. 2025. **Leveraging decoder architectures for learned sparse retrieval**. In *Knowledge-Enhanced Information Retrieval: Second International Workshop, KEIR 2025, Lucca, Italy, April 10, 2025, Revised Selected Papers*, page 19–35, Berlin, Heidelberg. Springer-Verlag.
- Jin Qin, Zihan Liao, Ziyin Zhang, Hang Yu, Peng Di, and Rui Wang. 2025. C2LLM technical report: A new frontier in code retrieval via adaptive cross-attention pooling. *arXiv preprint arXiv:2512.21332*.
- Hélio Victor F Santos, Vitor Costa, João Eduardo Montandon, and Marco Tulio Valente. 2025. Decoding the configuration of ai coding agents: Insights from claude code projects. *arXiv preprint arXiv:2511.09268*.
- Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Guodong Long, Kai Zhang, and Daxin Jiang. 2023. **Unifler: A unified retriever for large-scale retrieval**. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '23*, page 4787–4799, New York, NY, USA. Association for Computing Machinery.
- Livio Soares, Daniel Gillick, Jeremy Cole, and Tom Kwiatkowski. 2023. **NAIL: Lexical retrieval indices with efficient non-autoregressive decoders**. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2574–2589, Singapore. Association for Computational Linguistics.
- 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*.
- Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient query processing for scalable web search. *Foundations and Trends® in Information Retrieval*, 12(4-5):319–500.
- Zora Zhiruo Wang, Akari Asai, Frank F Xu, Yiqing Xie, Graham Neubig, Daniel Fried, and 1 others. 2025. CodeRAG-Bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214.
- Zhichao Xu, Aosong Feng, Yijun Tian, Haibo Ding, and Lin Lee Cheong. 2025. **CSPLADE: Learned sparse retrieval with causal language models**. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 99–114, Mumbai, India. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.
- Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024. Model merging in LLMs, MLLMs, and beyond: Methods, theories, applications, and opportunities. *ACM Computing Surveys*.

Jian Yang, Xianglong Liu, Weifeng Lv, Ken Deng, Shawn Guo, Lin Jing, Yizhi Li, Shark Liu, Xianzhen Luo, Yuyu Luo, and 1 others. 2025. From code foundation models to agents and applications: A comprehensive survey and practical guide to code intelligence. *arXiv preprint arXiv:2511.18538*.

Hansi Zeng, Julian Killingback, and Hamed Zamani. 2025. Scaling sparse and dense retrieval in decoder-only llms. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2679–2684.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

A Hyper-parameters

Training settings. Table 7 summarizes the main hyper-parameters we used in our experiments. We train our model on the training set of CoIR, containing of ten datasets merged together for a total of 2.2M training samples, for one epoch. Each training sample consists of one query, one positive, and n negatives. In practice, we use batch size of 256, achieved with gradient accumulation, and 7 or 8 negatives per query depending on the parameter size of the models. We used LoRA (Hu et al., 2022) rank of 64. The negatives are mined from retrieval made with Qwen3 Embedding (0.6B), and scored with Qwen3 Reranker (4B) to be then distilled (Zhang et al., 2025). We select the negative in-between the rank 50 to 100 following previous training of embedding models. All models are trained with a max length of 512. Since queries and documents are very long, we observed that models tend to produce high similarity scores, we thus use a high temperature in the KLD loss to mitigate and have more stable training, we used temperature of 300. The learning rate is set to $1e-4$. Model merging is done with weighted spherical merging (Yang et al., 2024) from three checkpoints: (1) the same base model after the first epoch, (2) and after a second epoch of training, (3) the base model trained with a max context length of 1024 for one epoch. Small models (0.6B) also contain a fourth checkpoint trained with full finetuning (no LoRA). All decoder models are pretrained on MS MARCO passages (Nguyen et al., 2016) with bi-directional attention to transform from causal to bi-directional attention models.

Dense models training settings. In §4, we train dense models in controlled experiments, to enable direct comparison with LSR models. These models are trained using eos token pooling. The learning rate is $1e-5$, the batch size is 128, the warmup ratio is 0.05 and the KL temperature is set to 0.1 (after some hyper-parameter search). 8 negatives are used for each query. The rest of the parameters are identical to the sparse models.

Masked Language pretraining SPLADE models use bi-directional attention. The models are first fine-tuned quickly with bi-directional attention and a masked language modelling objective. We use the exact same setup as in (Formal et al., 2026; Zeng et al., 2025). Note that this step is not very expensive: it requires about 15 hours on a single

GPU for the 8B model.

Hardware. Training is done on 4 A100 with 80GB of memory each. Retrieval latencies are computed on a single AMD EPYC 7313 processor for fair comparison between dense and LSR.

Latency experiments. For the LSR retrieval we used the *seismic* library, which contains an efficient implementation of inverted index (Bruch et al., 2024). In particular, for the latency experiments, we use *seismic* with parameters $k = 1000$, $query_cut = 500$, $heap_factor = 2.5$, and $n_knn = 0$. For *hnsu*, we use *faiss* (Douze et al., 2025) and parameters $M = 32$, $efConstruction = 40$ and vary $efSearch$. We retrieve $top_k = 1000$ in all cases.

B Datasets Statistics

We provide additional statistics about the benchmark in Table 8, including the number of datasets for each benchmark, the list of programming languages and the main tasks within each benchmark. We also provide some dataset sizes for queries and collections.

C Additional Analysis

Failure Cases. Figure 6 shows examples of the sparse activations produced by SPLADE-Code-8B. The main takeaway is that the model often captures the right high-level concepts. For example, the quicksort implementations in Java, Perl, and Haskell activate terms such as *quick*, *sort*, *pivot*, *partition*, *array*, and *swap*. This is encouraging, as the model is not only relying on exact token overlap, but also learns useful algorithmic expansions across programming languages.

However, these activations should not be read as full program understanding. In some cases, the representation captures the topic of the code but misses important details. For example, for SQL queries involving $MAX(Age)$, SPLADE-Code activates relevant terms such as *group*, *age*, *MAX*, and *oldest*, but does not clearly distinguish whether the maximum is computed over the whole table or only within a group. Similar issues can appear for small operator changes, boundary conditions, or ordering constraints. This is expected to some extent: SPLADE-Code is a first-stage retrieval model, not a symbolic executor or a program verifier. Its strength is to retrieve topically and semantically related candidates, while finer-grained correctness

	SPLADE-Code 0.6B	SPLADE-Code 1.7B	SPLADE-Code 8B
Base Model	Qwen/Qwen3-0.6B	Qwen/Qwen3-1.7B	Qwen/Qwen3-8B
Batch size	256	256	256
Negatives per query	8	7	7
LoRA Rank	64	64	64
Max Length	512	512	512
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Temperature (KLD)	300	300	300
Epochs	1	1	1
Bidirectional Attention	Yes	Yes	Yes
Warmup Ratio	0.01	0.01	0.01
Quantization	BF16	BF16	BF16
Lambda q	1×10^{-4}	1×10^{-4}	1×10^{-4}
Lambda d	1×10^{-4}	1×10^{-4}	1×10^{-4}
KLD Loss Weight	0.5	0.5	0.5
Weight Decay	0.0	0.0	0.0
Optimizer	Adam	Adam	Adam
Inference Max Length	2048	2048	2048
Pruning Inference	(500,1000)	(500,1000)	(500,1000)
Heap Factor (LSR)	2.5	2.5	2.5

Table 7: Main training configurations for the base models. For model merging we merge the same model after 1 and 2 epochs, and with a max length of 1024 (3 ckpts). For 0.6B we also merge a full finetuning model.

may need to be handled by a later reranking or verification step.

Full description of the qualitative examples (Table 4). In Example 1 (Python image loading), the model expands the representation with semantically related terms such as pillow and PIL, as well as exception types (e.g., IOError, FileNotFoundError). These activations align with common failure modes and help retrieve passages describing similar issues.

In Example 2 (SQL query), the model down-weights generic syntactic tokens (e.g., SELECT, FROM, WHERE) and instead emphasizes content-bearing terms such as oldest from MAX(Age). This suggests that the representation prioritizes semantics over syntax.

In Example 3 (Java implementation of quicksort), the input code does not explicitly mention “quicksort”, yet the model activates algorithmic and implementation-level concepts such as quick, pivot, sort, and java, along with related terms like swap, inplace, and array. Such activations shows how SPLADE-Code uses mostly English as matching language between textual instructions and code from different programming languages.

Efficiency. We provide in Figure 5 an additional comparison of the latency of SPLADE-Code-0.6B. Latencies are overall similar to the curve provided in the main body of the paper for SPLADE-Code-8B. Also note that we measure here the latency of

the retrieval only from passing through the inverted index or the vector search algorithm, we do not consider the latency from encoding the query and getting the sparse representation, since this time is similar for dense and LSR models. Also note that the added latency from two-step SPLADE is estimated here as the time to compute 1000 dot products, which we estimated to 50 microseconds for 1000 dot products between vectors of size 100K with only 400 non zero dimension.

Ablations. We provide in Table 9 and Table 10 the performance of models on individual datasets from each benchmark. This includes some models from ablation or models without checkpoint merging.

D Performance on English and Multilingual Benchmark

While the main focus of our research was to develop code retrieval model, we observed that the models were good in English and had some multilingual capacities. We provide in Table 11 and Table 12 the performance of SPLADE-Code on MTEB English and MTEB multilingual. As a comparison, LION-SP-8B (Zeng et al., 2025) achieves 48.5 on MTEB(eng, v2) and 50.0 on MTEB(Multilingual, v2). SPLADE-Code-8B is more effective than strong monolingual English retrieval models. SPLADE-EN-0.6B is the English SPLADE model we trained for the intermediate English finetuning ablation within Table 6. It achieves good performance in English.

Benchmark	#Datasets	#Lang	Task / Domain	Languages	Test Size	Collection Size
CoIR (Li et al., 2025b)	10	14	Code Retrieval Text-to-Code Code-to-Code Code-to-Text Hybrid Code (8 Coding Tasks)	Python, SQL, Go, Java, JS, C, CSS, PHP, C++, HTML, Rust, Shell, Swift, Ruby	180 – 53K	816 – 1M
CodeRAG-Bench (Wang et al., 2025)	6	1	Retrieval-Augmented Code Generation Open-Domain Basic Programming Repository-Level Retrieval (8 Coding Tasks)	Python	164 – 1K	237 – 40.9K
CPRet (Deng et al., 2025)	4	20+	Competitive Program- ming Retrieval Text-to-Code Code-to-Code Problem-to-Duplicate Simplified-to-Full (4 Coding Tasks)	Python, C++, C, Java, Perl, Groovy, R, Rust, Dart, C#, Matlab, CoffeeScript, Assembly, Haskell, Batchfile, Go, Ruby, JavaScript, Kotlin	168 – 10K	10K – 41.6K
MTEB-Code (build upon CoIR) (Muennighoff et al., 2023)	12	14	Code Retrieval Text-to-Code Code-to-Code Code-to-Text Hybrid Code (9 Coding Tasks)	Python, SQL, Go, Java, JS, C, CSS, PHP, C++, HTML, Rust, Shell, Swift, Ruby	180 – 53K	816 – 1M

Table 8: Benchmark statistics of CoIR, CodeRAG-Bench, CPRet and MTEB Code. Test and collection sizes are reported as ranges (min–max per dataset). MTEB Code is composed of CoIR with the two additional datasets: CodeEditRetrieval and a variation of CodeSearchNet.

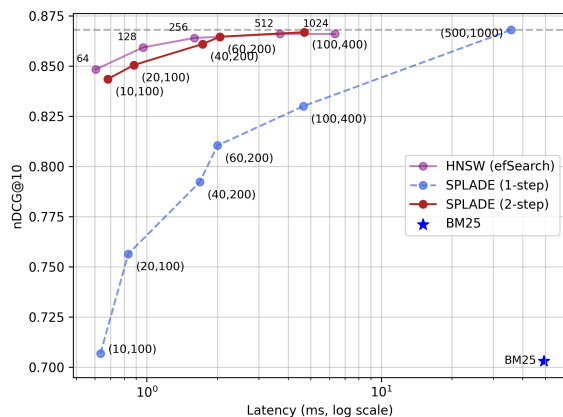


Figure 5: Latency of SPLADE-Code-0.6B for code representation with two-step SPLADE on Qwen3-0.6B. On Qwen3-0.6B the dot product for the dense is more efficient since embedding dimensions are lower, hence lower latencies with HNSW, but SPLADE still matches the Pareto frontier.

Code (Java)

```
static void quicksort(int[] x,int l,int r){
  if(l>=r)return;
  int i=l,j=r,p=x[(l+r)/2];
  while(i<=j){
    while(x[i]<p)i++;
    while(x[j]>p)j--;
    if(i<=j){
      int t=x[i]; x[i]=x[j]; x[j]=t;
      i++; j--;
    }
  }
  if(l<j)quicksort(x,l,j);
  if(i<r)quicksort(x,i,r);
}
```

Token	W	Token	W
quick	2.23	l	2.16
x	2.09	void	2.09
<i>pivot</i>	2.03	sort	1.98
<i>Java</i>	1.97	<i>array</i>	1.92
r	1.89	<i>inplace</i>	1.82
<i>swap</i>	1.80	<i>partition</i>	1.67

Code (Perl)

```
sub quicksort{
  my($x,$l,$r)=@_;
  return if $l>=$r;
  my $i=$l; my $j=$r;
  my $p=$x->[int(($l+$r)/2)];
  while($i<=$j){
    while($x->[$i]<$p){$i++;}
    while($x->[$j]>$p){$j--;}
    if($i<=$j){
      my $t=$x->[$i];
      $x->[$i]=$x->[$j]; $x->[$j]=$t;
      $i++; $j--;
    }
  }
  quicksort($x,$l,$j) if $l<$j;
  quicksort($x,$i,$r) if $i<$r;
}
```

Token	W	Token	W
<i>Perl</i>	2.17	quick	2.11
sort	2.08	perl	2.05
x	2.02	<i>inplace</i>	1.97
l	1.93	<i>pivot</i>	1.90
<i>array</i>	1.88	r	1.71
<i>swap</i>	1.62	<i>partition</i>	1.56

Code (Haskell)

```
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (p:xs) =
  quicksort [x | x <- xs, x < p]
  ++ [p]
  ++ quicksort [x | x <- xs, x >= p]
```

Token	W	Token	W
<i>pivot</i>	2.14	quick	2.11
<i>Haskell</i>	2.05	sort	1.97
xs	1.90	p	1.79
<i>list</i>	1.70	<i>partition</i>	1.64
<i>GHC</i>	1.65	<i>median</i>	1.51
<i>Prelude</i>	1.51	<i>greater</i>	1.50

Figure 6: Examples of sparse vocabulary activations produced by SPLADE-Code-8B for quicksort implementations in Java and in two languages appearing in CPRet but absent from CoIR’s language list. Highlighted tokens correspond to semantic or algorithmic expansions beyond exact surface syntax.

Model	Apps	CosQA	T2SQL	CSN [†]	CSN -COIR	CSN -CCR	CodeEdit Retrieval [†]	CodeTrans -Contest	StackOv -DL	CodeFeedback -ST	CodeFeedback -MT	Avg COIR	Avg MTEB	
BM25	4.7	18.7	24.9	63.5	70.3	59.3	53.3	47.7	34.4	70.2	68.1	59.1	45.8	47.9
SPLADE-lex.-0.6B	10.2	6.8	18.7	42.7	37.5	45.1	35.0	55.5	30.9	71.1	67.7	79.9	42.3	41.8
SPLADE-lex.-1.7B	12.8	7.7	17.9	46.3	39.0	48.9	35.9	55.2	29.7	73.7	68.1	82.0	43.5	43.1
SPLADE-lex.-8B	16.8	7.4	18.6	49.7	41.9	55.3	37.6	61.5	29.2	76.5	69.5	85.3	46.2	45.8
SPLADE-Code-no-merging-0.6B	67.3	33.4	69.0	90.3	87.6	94.6	65.5	89.2	23.7	93.4	77.1	90.3	72.6	73.5
SPLADE-Code-no-merging-1.7B	72.8	33.8	71.7	91.1	88.1	94.2	67.4	90.9	25.6	94.0	80.5	90.6	74.3	75.1
SPLADE-Code-no-merging-8B	87.8	34.0	71.1	92.1	89.9	96.7	74.8	93.9	23.1	96.4	81.1	91.9	76.7	77.8
SPLADE-Code-QwenCoder-0.5B	50.5	32.8	70.3	90.0	87.2	93.5	63.2	86.7	24.6	91.8	79.8	87.3	70.4	71.5
SPLADE-Code-QwenCoder-1.5B	54.3	30.5	72.7	90.6	87.6	92.2	64.9	87.7	31.4	92.6	81.7	88.6	71.9	72.9
SPLADE-Code-QwenCoder-7B	86.2	32.3	70.6	92.3	90.2	96.9	74.4	94.4	23.5	96.1	81.5	92.2	76.4	77.6
SPLARE-LlamaScope-7B-L26	78.0	32.6	75.0	90.2	88.2	95.1	75.1	93.3	27.2	95.7	83.9	90.6	76.0	77.1
SPLARE-Llama3-SAE-CoIR-7B-L24	81.0	32.3	70.6	91.6	89.0	95.7	74.1	93.9	20.9	96.7	74.6	91.7	74.6	76.0
SPLARE-Qwen3-SAE-CoIR-8B-L32	87.6	28.2	70.7	90.8	89.1	96.0	73.5	93.1	22.9	96.2	77.2	91.4	75.2	76.4
SPLADE-Code-0.6B	66.5	35.4	74.3	90.5	86.2	91.4	68.4	90.9	31.4	93.3	84.7	92.0	74.6	75.4
SPLADE-Code-1.7B	71.1	32.1	73.3	91.3	87.9	92.8	66.7	92.3	25.4	93.9	84.1	91.4	74.4	75.2
SPLADE-Code-8B	86.7	32.5	75.7	92.1	88.9	94.6	77.1	94.6	28.1	96.5	87.6	93.9	77.9	79.0

Table 9: Detailed MTEB Code retrieval results of ablations (nDCG@10), using pruning (500, 1000).

Model	CodeRAG							CPRet				
	Human Eval	MBPP	DS-1000	ODEX	Repo Eval	SWE Bench-L	Avg	T2C	C2C	P2Dup	S2Full	Avg
BM25	100.0	98.6	5.2	6.7	93.2	43.0	57.7	0.9	7.4	12.4	53.8	18.6
SPLADE-lex.-0.6B	100.0	99.0	5.2	1.7	94.1	33.5	55.6	7.1	13.0	20.1	58.1	24.6
SPLADE-lex.-1.7B	99.8	99.2	4.8	2.9	94.3	28.0	54.8	8.9	16.2	22.8	59.8	26.9
SPLADE-lex.-8B	99.8	98.4	6.4	3.2	93.9	38.9	56.7	11.6	19.5	23.1	63.4	29.4
SPLADE-Code-no-merging-0.6B	100.0	98.9	22.8	27.0	95.1	38.2	63.7	40.4	50.2	43.8	91.0	56.4
SPLADE-Code-no-merging-1.7B	100.0	98.9	29.0	28.3	93.4	39.9	64.9	46.7	57.6	52.9	92.8	62.5
SPLADE-Code-no-merging-8B	100.0	98.6	38.1	36.7	90.8	47.0	68.5	70.7	76.8	64.1	95.8	76.8
SPLADE-Code-QwenCoder-0.5B	100.0	99.1	24.0	21.0	93.3	36.8	62.4	28.1	35.5	37.2	85.4	46.6
SPLADE-Code-QwenCoder-1.5B	100.0	99.3	24.0	22.4	93.2	44.9	63.9	30.0	41.6	42.3	87.7	50.4
SPLADE-Code-QwenCoder-7B	100.0	98.6	36.5	36.0	90.2	66.7	71.3	70.5	73.4	59.9	95.6	74.9
SPLARE-LlamaScope-7B-L26	100.0	93.4	30.2	24.2	75.0	50.8	62.3	59.4	66.6	51.7	93.5	67.8
SPLARE-Llama3-SAE-CoIR-7B-L24	100.0	98.3	37.4	38.5	92.9	49.3	69.4	63.9	72.2	58.1	94.4	72.2
SPLADE-Code-0.6B	100.0	99.2	21.1	29.1	95.8	39.5	64.1	42.4	47.6	49.7	92.6	58.1
SPLADE-Code-1.7B	100.0	99.0	30.7	28.4	93.0	42.5	65.6	48.1	56.8	55.4	93.5	63.4
SPLADE-Code-8B	100.0	99.4	42.0	38.8	94.2	52.6	71.2	70.9	76.4	66.3	96.1	77.4

Table 10: Detailed out-of-domain retrieval performance on CodeRAG and CPRet for ablation models (nDCG@10). Retrieval with pruning (1000, 1000).

Dataset	SPLADE-Code-0.6B	SPLADE-Code-8B	SPLADE-EN-0.6B
Average	45.4	49.7	48.1
ArguAna	56.4	59.9	52.3
CQADupstackGamingRetrieval	56.8	59.2	56.0
CQADupstackUnixRetrieval	40.5	46.7	37.6
ClimateFEVERHardNegatives	19.0	21.7	19.6
FEVERHardNegatives	60.8	66.0	62.2
FiQA2018	36.9	47.8	36.0
HotpotQAHardNegatives	47.8	52.9	59.2
SCIDOCS	17.4	19.7	16.6
TRECCOVID	68.6	69.1	77.3
Touche2020Retrieval.v3	49.8	53.5	63.9

Table 11: Results of the SPLADE-Code models on MTEB(eng, v2) with pruning (40, 400).

Dataset	SPLADE-Code-0.6B	SPLADE-Code-8B	SPLADE-EN-0.6B
Average	48.3	54.3	47.0
AILAStatutes	28.1	32.4	27.7
ArguAna	56.4	59.9	52.3
BelebeleRetrieval	54.1	77.1	48.4
CovidRetrieval	73.3	72.7	78.9
HagridRetrieval	98.7	98.7	98.8
LEMBPasskeyRetrieval	38.8	38.8	38.8
LegalBenchCorporateLobbying	94.1	95.7	93.1
MIRACLRetrievalHardNegatives	31.5	43.2	31.5
MLQARetrieval	55.7	71.5	56.6
SCIDOCS	17.4	19.7	16.6
SpartQA	4.5	2.9	1.1
StackOverflowQA	86.9	93.5	74.7
StatcanDialogueDatasetRetrieval	28.0	37.2	22.3
TRECCOVID	68.6	69.1	77.3
TempReasonL1	0.2	0.2	0.3
TwitterHjerneRetrieval	49.7	71.9	42.5
WikipediaRetrievalMultilingual	80.8	89.1	80.8
WinoGrande	2.9	4.8	3.7

Table 12: Results of the SPLADE-Code models on MTEB(Multilingual, v2) with pruning (40, 400).