

MAP4CS: A Multi-dimensional Data Pruning Framework for Efficient Code Retriever Fine-tuning

YUXUAN CHEN, School of Software Engineering, Sun Yat-sen University, China

MINGWEI LIU*, School of Software Engineering, Sun Yat-sen University, China

GUANGSHENG OU, School of Software Engineering, Sun Yat-sen University, China

ZEKAI ZHANG, School of Software Engineering, Sun Yat-sen University, China

ZIKE LI, School of Software Engineering, Sun Yat-sen University, China

YANLIN WANG, School of Software Engineering, Sun Yat-sen University, China

PEILIN ZHENG, School of Software Engineering, Sun Yat-sen University, China

Retrieval-Augmented Generation (RAG) has become a cornerstone in software engineering for enhancing Large Language Models (LLMs) with domain-specific knowledge. However, adapting retrievers to evolving code repositories remains challenging due to the noise and redundancy inherent in massive code corpora. Standard fine-tuning on the full corpus is computationally expensive and often leads to sub-optimal performance due to negative transfer from low-quality samples. Conversely, simple random sampling fails to guarantee data representativeness.

To address these challenges, we propose MAP4CS (Multi-dimensional Awareness Pruning for Code Search), an adaptive data pruning framework. MAP4CS identifies a small, high-quality core subset by integrating syntactic structure, semantic diversity, and distributional representation, followed by a rigorous rule-based filtering pipeline. Extensive experiments on two large-scale datasets demonstrate that MAP4CS consistently outperforms random sampling baselines using only 5% of the training data. Remarkably, it achieves performance comparable to, or even superior to, fine-tuning on the full dataset, validating the “less is more” hypothesis in data-centric AI. Furthermore, linguistic analysis reveals an adaptive optimization mechanism: MAP4CS automatically functions as a de-duplicator for redundant corpora and a denoiser for chaotic ones, constructing a training corpus that is both lexically diverse and information-dense.

CCS Concepts: • **Software and its engineering** → **General programming languages**; • **Computing methodologies** → *Natural language processing*; • **Information systems** → *Information retrieval*.

Additional Key Words and Phrases: Code Search, Retrieval-Augmented Generation, Data Pruning, Large Language Models

*Corresponding author.

Authors’ Contact Information: Yuxuan Chen, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, chenyx677@mail2.sysu.edu.cn; Mingwei Liu, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, liumw26@mail.sysu.edu.cn; Guangsheng Ou, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, ougsh3@mail2.sysu.edu.cn; Zekai Zhang, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, zhangzk27@mail2.sysu.edu.cn; Zike Li, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, lizk8@mail2.sysu.edu.cn; Yanlin Wang, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, yanlin-wang@outlook.com; Peilin Zheng, School of Software Engineering, Sun Yat-sen University, Zhuhai, China, zhengplin@mail.sysu.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

Manuscript submitted to ACM

1

ACM Reference Format:

Yuxuan Chen, Mingwei Liu, Guangsheng Ou, Zekai Zhang, Zike Li, Yanlin Wang, and Peilin Zheng. 2026. MAP4CS: A Multi-dimensional Data Pruning Framework for Efficient Code Retriever Fine-tuning. *ACM Trans. Softw. Eng. Methodol.* XX, X, Article XXX (June 2026), 24 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

The advent of Large Language Models (LLMs) has fundamentally reshaped the landscape of software engineering, empowering automated tools for code completion, summarization, and synthesis [63, 64]. Despite their impressive capabilities, off-the-shelf LLMs often struggle with domain-specific software repositories, particularly when dealing with private enterprise libraries, evolving APIs, or legacy systems that were not part of their pre-training data [60]. To mitigate limitations such as hallucinations and knowledge cutoffs, Retrieval-Augmented Generation (RAG) has emerged as a pivotal paradigm [30, 43]. By retrieving relevant code snippets from an external knowledge base to serve as context, RAG significantly enhances the accuracy and reliability of generated code [30, 60]. Consequently, the performance of the neural code retriever becomes the bottleneck of the entire pipeline [24].

Regrettably, generic code retrievers often exhibit sub-optimal performance in these specialized software domains due to a lack of exposure to relevant proprietary knowledge during pre-training. To adapt general-purpose retrievers to these domains, fine-tuning on domain-specific corpora is essential [5]. This process typically involves extracting pairs of “Natural Language Query” and “Code Snippet”, denoted as (Q, C) , from the target repositories [23]. The model is then fine-tuned on these sample pairs to align the embedding space of the retriever with the specific coding standards and business logic of the target domain [13].

A common practice in the industry is to fine-tune models on the full available corpus to maximize knowledge coverage [27, 35]. However, blindly utilizing all data presents a critical trade-off between efficiency and effectiveness. **First, the “Noise Toxicity” Problem:** Real-world code repositories are fraught with noise, such as redundant boilerplate code, low-quality auto-generated comments, and deprecated functions [1, 28]. Training on such data can lead to *negative transfer*, where the retriever overfits to superficial lexical overlaps rather than learning robust semantic representations [6]. **Second, the Computational Barrier:** The sheer scale of industrial repositories makes full-corpus fine-tuning computationally prohibitive and environmentally costly [49], creating a significant barrier for frequent updates in continuous integration (CI) environments.

To reduce training costs, a straightforward alternative is Random Sampling [38, 48]. While this decreases fine-tuning time, this content-agnostic strategy is flawed for code search tasks. It often leads to the Loss of Long-Tail Knowledge, discarding crucial but infrequent domain-specific APIs [20], while simultaneously suffering from the Retention of High-Entropy Noise, preserving misaligned pairs that confuse the model’s decision boundaries [59]. Thus, neither full-data training nor random sampling offers a viable path for efficient and effective domain adaptation.

This dilemma poses a central research question: *Is it possible to intelligently identify a small, high-quality subset of data that trains a better code retriever than the full corpus?* While data pruning has shown promise in general NLP, directly applying these text-centric methods to code retrieval is insufficient. As we will discuss in Section 2, generic filtering often ignores the unique structural dependencies of code, while traditional software engineering approaches tend to focus on defensive filtering to remove noise, rather than selecting samples with high syntactic and semantic value.

To address these challenges, we propose **MAP4CS (Multi-dimensional Awareness Pruning for Code Search)**, an adaptive data pruning framework designed specifically for efficient retriever fine-tuning. Unlike rigid filtering rules

Creates an UnicastSubject with an internal buffer capacity hint 16. @param <T> the value type @return an UnicastSubject instance		Query	
<pre>UnicastSubject create() { return new UnicastSubject<T>(bufferSize(), true); }</pre>	Ground Truth	<pre>UnicastSubject create(int capacityHint) { return new UnicastSubject<T>(capacityHint, true); }</pre>	Random Top1 Result
	$Sim_{MAP4CS} \approx 0.86$		$Sim_{MAP4CS} \approx 0.69$
	$Sim_{Random} \approx 0.63$		$Sim_{Random} \approx 0.64$

Fig. 1. **Motivation Example 1 (Semantic Gap):** The baseline model is misled by the explicit keyword “capacity hint” in a distractor method. In contrast, training with diverse semantic strata helps identify the Ground Truth which handles this logic implicitly.

or general-purpose scorers, MAP4CS treats data selection as a multi-objective optimization problem within local semantic contexts. It operates through a rigorous coarse-to-fine pipeline: First, it structures the chaotic code corpus using Stream-Based Semantic Clustering to localize diversity and isolate noisy regions. Second, it evaluates samples via a Multi-Metric Scoring mechanism, integrating *Syntactic Structural Diversity*, *Semantic Alignment Consistency*, and *Distributional Representativeness*. Finally, a cascade filtering and sampling strategy selects the core subset. This multi-dimensional awareness allows MAP4CS to adaptively function as a “de-duplicator” for redundant corpora, a “denoiser” for chaotic datasets, and a “structure-preserver” for complex logic.

To evaluate MAP4CS, we conduct extensive experiments on two large-scale datasets, CSN-Java and Query4Code, using both code-specialized (UniXcoder [15]) and general-purpose (GTE_{base} [61]), and foundation-model-based (Qwen3-Embedding [62]) dense encoders, alongside BM25 as a lexical baseline. The results demonstrate that using only 5% of the training data, MAP4CS consistently outperforms random sampling baselines across all dense encoders and, remarkably, achieves comparable or even superior performance to fine-tuning on the full dataset. For instance, on the CSN-Java dataset, MAP4CS improves the retrieval MRR by 31.9% over random sampling with UniXcoder and by 45.5% with Qwen3-Embedding, highlighting the “less is more” phenomenon in data-centric AI.

In summary, this paper makes the following contributions:

- We introduce **MAP4CS**, a multi-dimensional pruning framework tailored for code search that selects high-value samples by integrating syntactic structure, semantic alignment, and distributional density.
- We propose an adaptive selection mechanism that effectively resolves the data scale-quality trade-off by filtering high-entropy noise while maximizing sample diversity.
- Extensive experiments demonstrate that MAP4CS reduces training costs by over 90% while achieving performance comparable to or superior to full-dataset fine-tuning, validating the “less is more” hypothesis.

2 Background and Motivation

In this section, we analyze the limitations of existing data selection methodologies when applied to code retrieval and present motivational examples that necessitate a multi-dimensional pruning approach.

2.1 The Landscape of Data Pruning

In the general domain, sophisticated selection strategies have matured into two dominant paradigms: *metric-based heuristics* and *learning-based optimization*. The former relies on statistical signals—such as perplexity or error margins—to filter data, as demonstrated by the “Textbooks Are All You Need” hypothesis [14] and LIMA [65]. The latter leverages

training dynamics, employing techniques like Influence Functions [29], Core-set Selection [38], or tracking Forgetting Events [51] to mathematically quantify the contribution of each sample. Even in instruction tuning, frameworks like AlpaGasus [4] utilize teacher models to score sample complexity.

However, a direct transposition of these text-centric or image-centric algorithms to the domain of code retrieval is fundamentally insufficient. Existing approaches generally fall into two categories, neither of which fully addresses the unique duality of source code [3] (i.e., natural language intent vs. programming logic).

General Data Filtering Ignorance of Structure. General data selection methods typically treat data as flat sequences or continuous vector spaces [65]. When applied to code, metrics like language model perplexity focus solely on the probability of token sequences [14]. This approach ignores the rigid syntactic structures (e.g., Abstract Syntax Trees) and execution logic that define programming languages [1, 16]. A code snippet might be statistically predictable (low perplexity) yet syntactically trivial or functionally irrelevant to the retrieval query. By failing to account for structural integrity, these methods risk selecting “fluent” but logically empty boilerplate code, leading to sub-optimal representation learning [47].

Defensive Nature of Code Cleaning. Within the software engineering community, data management efforts have predominantly focused on sanitization rather than valuation. Standard practices involve filtering explicit noise, such as removing near-duplicates (Deduplication) [1] or discarding snippets with parse errors [28]. While these “cleaning” steps are necessary prerequisites, they are essentially defensive: they eliminate the “worst” data but lack the intelligence to actively identify the “best” data. They do not assess whether a sample contributes to bridging the **semantic gap** between natural language queries and code implementations, nor do they consider the distributional redundancy that persists even after deduplication.

Consequently, current methodologies leave us in a compromise: we must choose between general pruning methods that misunderstand code structure, or traditional cleaning methods that lack selection intelligence. This gap underscores the need for a specialized framework tailored to the multi-dimensional nature of code.

2.2 Motivation

To illustrate the limitations of naive data selection strategies and the necessity of structural and semantic awareness, we analyze two typical failure modes in code retrieval where models trained on randomly sampled data fail to align user intent with code implementation.

(1) Implicit Semantics vs. Lexical Overlap. As shown in Figure 1, queries often rely on implicit domain knowledge rather than explicit keywords. For a query requesting an “internal buffer capacity hint”, the baseline model trained on randomly sampled data is distracted by a superficial keyword match in an incorrect method (`capacityHint`), missing the ground truth which handles capacity implicitly. This failure highlights the necessity of **Semantic Alignment Consistency**—we select “hard positives” during training to teach the model to look beyond surface-level lexical overlaps.

(2) Structural Logic Alignment. Figure 2 demonstrates the challenge of distinguishing specific execution logic from generic boilerplate. The query implies a specific sequence of actions (“Signal value AND onComplete”). Lacking structural guidance, the baseline overfits to shared domain terms (e.g., “Complete”, “Queue”) in a complex utility method. This underscores the need for **Syntactic Structural Diversity**. By incorporating AST-based pruning, we ensure the training data covers diverse control flows, enabling the model to align natural language action sequences with the actual code execution logic.

Signals the given value and an onComplete if the downstream is ready to receive the final value. @param n the value to emit		Query
<pre>protected final void complete(R n) { // ... (omitted backpressure checks) for (;) { long r = get(); // ... (omitted state checks) if ((r & REQUEST_MASK) != 0) { lazySet(COMplete_MASK + 1); // [KEY EVIDENCE]: Explicit Sequence downstream.onNext(n); downstream.onComplete(); return; } // ... (omitted CAS logic) } }</pre>	Ground Truth $Sim_{MAP4CS} \approx 0.48$ $Sim_{Random} \approx 0.42$	<pre>public static <T> void postComplete(...) { // 1. Only handles completion, no emission if (queue.isEmpty()) { actual.onComplete(); return; } // 2. Complex drain logic (Not fast-path) if (postCompleteDrain(...)) { return; } for (;) { // ... (omitted complex loop state management) } }</pre> Random Top1 Result $Sim_{MAP4CS} \approx 0.39$ $Sim_{Random} \approx 0.54$

Fig. 2. **Motivation Example 2 (Structural Alignment):** The query implies a specific sequential logic (“onNext” then “onComplete”). Without structural awareness, the baseline overfits to domain terms in complex boilerplate, missing the correct control flow found by AST-aware pruning.

These failure modes highlight that effective data pruning must transcend simple statistical sampling; MAP4CS addresses this by explicitly optimizing for both semantic alignment and structural diversity, ensuring the retention of high-value samples that are critical for generalization but often discarded by naive baselines.

3 Approach

In this section, we formulate the data pruning problem and present **MAP4CS** (Multi-dimensional Awareness Pruning for Code Search). As illustrated in Figure 3, MAP4CS operates through a rigorous, coarse-to-fine pipeline: (1) **Stream-Based Semantic Clustering**, which partitions the streaming corpus into local contexts; (2) **Multi-Metric Sample Scoring**, which evaluates samples based on structure, semantics, and distribution; and (3) **Data Filtering and Sampling**, which executes the final data distillation under a strict budget.

3.1 Problem Formulation

Let $\mathcal{D} = \{(q_i, c_i)\}_{i=1}^N$ denote a large-scale, potentially noisy code corpus containing N training pairs, where q_i represents its corresponding natural language query and c_i represents a code snippet. The goal of code retrieval is to learn a scoring function $f_\theta(q, c)$ that maximizes the similarity between relevant pairs.

The data pruning problem can be formalized as finding an optimal subset $\mathcal{S} \subset \mathcal{D}$ that satisfies a strict budget constraint $|\mathcal{S}| \leq B$ (where $B \ll N$, e.g., $B = 0.05N$), such that the model trained on $\mathcal{S}$ achieves maximal performance. Formally, we aim to solve:

$$\mathcal{S}^* = \arg \max_{\mathcal{S} \subset \mathcal{D}, |\mathcal{S}| \leq B} \mathbb{E}_{(q,c) \sim \mathcal{D}_{val}} [\mathcal{M}(f_{\theta_S}(q, c))] \quad (1)$$

where θ_S denotes the parameters of the model fine-tuned on subset $\mathcal{S}$, and $\mathcal{M}(\cdot)$ is the evaluation metric (e.g., MRR) on a validation set.

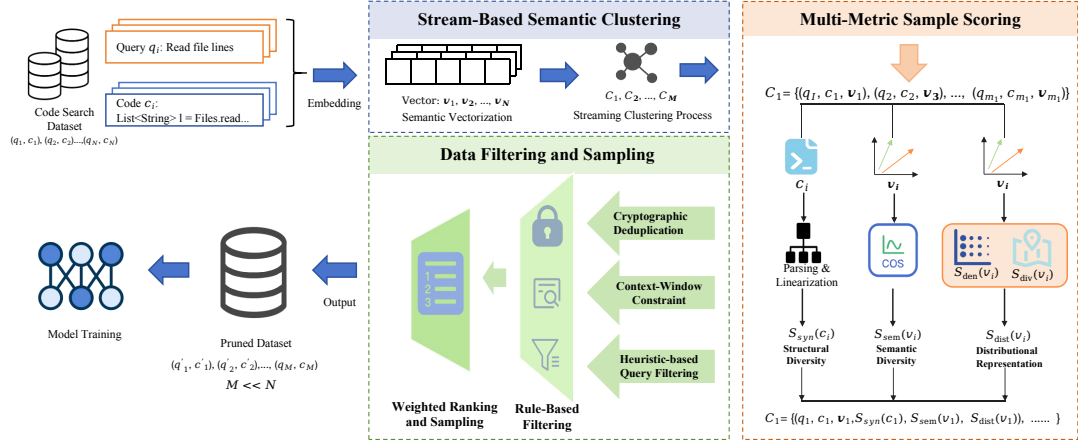


Fig. 3. Overview of MAP4CS

3.2 Stream-Based Semantic Clustering

The original code corpus usually exhibits a long-tailed distribution, with mainstream functions (such as standard I/O operations) often obscuring the logic specific to certain domains. To preserve the intrinsic semantic diversity of the data, we first structure the data into local semantic contexts.

3.2.1 Clustering Algorithm Selection. Standard clustering algorithms pose significant challenges for massive code corpora. **(1) Unknown K :** K -Means requires a predefined number of clusters (K), which is impossible to estimate accurately for evolving, streaming code data. **(2) Noise Sensitivity:** Density-based methods (e.g., HDBSCAN), while effective for shape detection, often classify distinct long-tail functionalities as noise (outliers) in high-dimensional spaces, resulting in data loss. **(3) Efficiency:** With $O(N^2)$ complexity, hierarchical methods are computationally prohibitive for datasets with millions of samples.

To address these limitations, we employ the **Single-Pass Clustering** algorithm [41]. It processes data in a streaming fashion with $O(N)$ complexity and dynamically discovers cluster structures without a fixed K . The core mechanism involves dynamic centroid updates to adapt to concept drift in the data stream. The implementation of Single-Pass Clustering is shown in Algorithm 1.

3.2.2 Clustering Process. The implementation of our clustering phase involves three key steps: vectorization, streaming clustering, and threshold determination.

Semantic Vectorization. Before formal clustering, we utilize a pre-trained encoder (e.g., UniXcoder) to map each code-query pair (q_i, c_i) into a dense semantic vector v_i . This step ensures that subsequent operations are performed in a high-dimensional semantic space capturing deep functional relationships.

Streaming Clustering Process. The core mechanism involves dynamic centroid updates to adapt to the streaming nature of the data. As detailed in Algorithm 1, for each incoming sample v_i , we calculate its cosine similarity with existing cluster centroids C . If the maximum similarity exceeds a threshold τ , the sample is assigned to the best-matching cluster, and the centroid is updated via a weighted moving average (Line 9). Otherwise, a new cluster is initialized.

Algorithm 1 Single-Pass Clustering with Dynamic Centroids**Require:** Embeddings $\mathcal{E} = \{\mathbf{v}_1, \dots, \mathbf{v}_N\}$, Threshold τ **Ensure:** Cluster Labels $\mathcal{L}$, Centroids C

```

1: Initialize  $C \leftarrow \emptyset$ , Counts  $\mathcal{N} \leftarrow \emptyset$ 
2: for  $i = 1$  to  $N$  do
3:    $\mathbf{v}_i \leftarrow \mathcal{E}[i]$ 
4:   if  $C$  is empty then
5:     Create new cluster  $k_{new} \leftarrow 0$ 
6:      $C[k_{new}] \leftarrow \mathbf{v}_i$ ;  $\mathcal{N}[k_{new}] \leftarrow 1$ 
7:      $\mathcal{L}[i] \leftarrow k_{new}$ 
8:     continue
9:   end if
10:  Calculate similarities to all centroids
11:   $k^* \leftarrow \arg \max_j (\cos(\mathbf{v}_i, \mathbf{c}_j))$  ▷ Find nearest centroid
12:   $sim_{max} \leftarrow \max_j (\cos(\mathbf{v}_i, \mathbf{c}_j))$ 
13:  if  $sim_{max} \geq \tau$  then
14:     $\mathcal{L}[i] \leftarrow k^*$ 
15:     $n \leftarrow \mathcal{N}[k^*]$ 
16:     $\mathbf{c}_{k^*} \leftarrow \frac{\mathbf{c}_{k^*} \cdot n + \mathbf{v}_i}{n+1}$  ▷ Dynamic Update
17:     $\mathcal{N}[k^*] \leftarrow n + 1$ 
18:  else
19:    Create new cluster  $k_{new} \leftarrow |C|$ 
20:     $C[k_{new}] \leftarrow \mathbf{v}_i$ ;  $\mathcal{N}[k_{new}] \leftarrow 1$ 
21:     $\mathcal{L}[i] \leftarrow k_{new}$ 
22:  end if
23: end for
24: return  $\mathcal{L}$ 

```

Threshold Determination. Based on preliminary experiments, we set the similarity threshold at $\tau = 0.55$. This threshold strikes a balance between *intra-cluster cohesion* and *semantic coverage*: it ensures that elements within each cluster are semantically relevant (high cohesion) while preventing the formation of excessive singleton clusters (over-fragmentation) or overly broad categories.

3.3 Multi-Metric Sample Scoring

Within each local semantic cluster, simply selecting the "most similar" samples leads to homogeneity. MAP4CS evaluates samples across three independence metrics to ensure diversity and representativeness: Structural Diversity (S_{syn}), Semantic Consistency (S_{sem}), and Distributional Representation (S_{dist}).

3.3.1 Structural Diversity (S_{syn}). Neural retrieval models often tend to overfit surface-level lexical features rather than learning deep programming logic structures[47]. To enable the model to perceive different syntactic structures and enhance its generalization ability, the training data need include diverse syntactic implementations of the same functionality.

We introduce an Abstract Syntax Tree (AST)-based method to quantify structural diversity. The process consists of two stages:

(1) Parsing & Linearization. To extract structural features from source code, we utilize Tree-sitter, a high-performance incremental parsing generator capable of building concrete syntax trees for various programming languages. It is particularly robust against syntax errors often found in noisy corpora.

For a given code snippet c , we first generate its AST using Tree-sitter[39]. To reduce the computational complexity of similarity comparisons, we perform a Depth-First Search (DFS) traversal to convert the tree structure into a linearized string sequence in S-expression format[2]. Take the following Python function as an example.

```
def add(a, b): return a + b
```

After parsing and linearization, its structural representation becomes:

```
(function_definition (identifier) (parameters (identifier) (identifier)) (
  return_statement (binary_operator (identifier) (identifier))))
```

This representation abstracts away specific variable names (e.g., 'a', 'b'), retaining only 'identifier' placeholders and syntactic tags. This allows for a focus on logical structure rather than naming conventions in computation.

(2) Similarity Calculation & Scoring. Once the linearized AST sequences are obtained, we quantify the structural similarity using sequence matching. Specifically, for a sample c_i in cluster C_k , its structural diversity score $S_{syn}(c_i)$ is defined as the complement of the average sequence similarity to other samples in the cluster:

$$S_{syn}(c_i) = 1 - \frac{1}{|C_k| - 1} \sum_{\substack{c_j \in C_k \\ j \neq i}} \text{SeqSim}(\text{AST}(c_i), \text{AST}(c_j)) \quad (2)$$

Here, $\text{SeqSim}(\cdot)$ computes the ratio of the Longest Common Subsequence (LCS) [21] between two linearized AST strings, implemented via the `difflib.SequenceMatcher` algorithm. Where we define $S_{syn}(c_i) = 0$ (or 1) if $|C_k| = 1$. A higher S_{syn} indicates that the sample possesses a unique syntactic structure within its local cluster, thereby offering richer training signals.

3.3.2 Semantic Diversity (S_{sem}). Code Search tasks typically involve complex semantic matching scenarios, ranging from intuitive keyword matching to deep intent understanding. Merely retaining high-similarity samples might ensure alignment accuracy but could lead the model to learn only simple pattern matching (Easy Positives), ignoring deeper or ambiguous semantic associations. To enable the retriever to comprehensively learn semantic hierarchical features and improve generalization across queries of varying difficulty, the training data should retain a complete semantic spectrum from strong to weak alignment. By covering samples from high, medium, and low similarity intervals, the model is forced to learn robust decision boundaries rather than memorizing high-frequency co-occurrences.

We measure the cosine similarity between the code embedding $\mathbf{v}_c$ and query embedding $\mathbf{v}_q$.

$$S_{sem}(c_i, q_i) = \cos(\mathbf{v}_{c_i}, \mathbf{v}_{q_i}) \quad (3)$$

Driven by this rationale, we employ a Stratified Sampling strategy. Instead of strictly maximizing similarity, we sample proportionally from high, medium, and low similarity tiers. This ensures the dataset includes standard positives for basic learning, hard positives for robustness, and edge cases for diversity.

3.3.3 Distributional Representation (S_{dist}). In the vector space, data representativeness is a multifaceted concept: it requires covering both the core regions of the feature space (representativeness) and the boundary regions (diversity).

Focusing solely on cluster centers leads to mode collapse, while focusing purely on outliers introduces noise. To address this trade-off, we introduce a composite metric, denoted as $S_{dist}(\mathbf{v}_i)$, which operates directly on the dense semantic vectors $\mathbf{v}_i$ generated in the semantic clustering stage. We define the distributional score as a tuple of two complementary sub-metrics:

$$S_{dist}(\mathbf{v}_i) = (S_{den}(\mathbf{v}_i), S_{div}(\mathbf{v}_i)) \quad (4)$$

where S_{den} and S_{div} are calculated as follows:

1. Representative Density (S_{den}). Based on Kernel Density Estimation (KDE), we calculate a density score S_{den} for a sample $\mathbf{v}_i$ within a cluster C_k (size n) using a Gaussian kernel K_h :

$$S_{den}(\mathbf{v}_i) = \frac{1}{n} \sum_{j=1}^n K_h(\|\mathbf{v}_i - \mathbf{v}_j\|) \quad (5)$$

High-density samples are selected as they represent the centroid or "mainstream" patterns of the cluster.

2. Coverage Diversity (S_{div}). To cover semantic boundaries, we calculate the diversity score S_{div} based on the Euclidean distance to the nearest neighbor:

$$S_{div}(\mathbf{v}_i) = \min_{j \neq i} \|\mathbf{v}_i - \mathbf{v}_j\|_2 \quad (6)$$

We select samples with the highest S_{div} scores (i.e., those furthest from neighbors) to maximize the semantic breadth covered by the pruned dataset.

3.4 Data Filtering and Sampling

After evaluating samples across multiple dimensions, simply aggregating the top-scoring candidates may still result in a dataset plagued by redundancy or engineering incompatibilities. To address this, MAP4CS employs a rigorous "Filter-then-Rank" cascade pipeline, combining deterministic hard rules with adaptive soft scoring.

3.4.1 Step 1: Rule-Based Filtering. Before performing expensive ranking operations, it is computationally efficient to discard samples that are fundamentally invalid or redundant. This step ensures that the final dataset meets basic engineering standards for efficient LLM training.

We implement a three-stage sequential filter:

- (1) **Cryptographic Deduplication:** To eliminate exact redundancy, we compute the **SHA-256 hash** of the normalized code content. Duplicate instances share the same hash and are removed, ensuring each code sample is unique.
- (2) **Context-Window Constraint:** Training retrievers on excessively long sequences leads to truncation and semantic loss. We utilize a pre-trained tokenizer to filter out code snippets exceeding the context window (specifically, $L > 1024$ tokens).
- (3) **Heuristic-based Query Filtering:** Inspired by the syntactic filter in NLQF [50], we directly apply its heuristic rules to cleanse the dataset. Noisy queries containing excessive non-ASCII characters, URL links, or specific metadata tags create noisy labels that hinder retrieval performance. Therefore, we filter out these elements as well as sequences that are too short to convey meaningful intent to ensure semantic clarity.

3.4.2 Step 2: Weighted Ranking and Sampling. Theoretical analysis suggests that different quality dimensions contribute differently to the training signal. Structural and semantic consistencies are foundational for correctness, while distributional metrics provide auxiliary regularization. A uniform summation would ignore these varying importances.

For the survivors of the hard filtering, we first perform Min-Max Normalization on their raw scores to map them into a unified range $[0, 1]$. We then compute a composite score S_{total} using a weighted fusion strategy:

$$S_{total}(i) = w_{syn} \cdot \hat{S}_{syn}(i) + w_{sem} \cdot \hat{S}_{sem}(i) + w_{den} \cdot \hat{S}_{den}(i) + w_{div} \cdot \hat{S}_{div}(i) \quad (7)$$

Based on empirical tuning and domain knowledge, we assign weights as follows: $w_{syn} = w_{sem} = 1/3$ to prioritize foundational data correctness, and $w_{den} = w_{div} = 1/6$ to balance distributional representativeness. Finally, we rank all samples by S_{total} and select the top $\mathcal{K}$ instances (e.g., top 5%) to form the final pruning dataset $\mathcal{S}^*$.

4 Evaluation

To comprehensively evaluate the effectiveness and efficiency of MAP4CS, we design experiments to answer the following three research questions:

- **RQ1 (Effectiveness Analysis):** How does MAP4CS perform on downstream code retrieval tasks?
- **RQ2 (Ablation Study):** What is the contribution of each component in our framework?
- **RQ3 (Parameter Analysis):** How does the pruning ratio impact model performance?

4.1 Experimental Setup

4.1.1 Evaluation Benchmarks. To assess the generalizability of the fine-tuned models, we evaluate them on four diverse benchmarks using the COIR framework. We categorize these benchmarks into two groups to test different capabilities: **CodeSearchNet** and **CoSQA** serve as in-domain datasets to evaluate standard code search performance. Conversely, **APPS** and **Synthetic Text2SQL**, which involve retrieving corresponding code or SQL solutions based on problem descriptions, are treated as external retrieval datasets to assess the model’s out-of-domain (OOD) generalization.

CodeSearchNet (CSN) [23]: A widely used dataset in the field of code search, covering six programming languages: Python, Java, Go, PHP, Ruby, and JavaScript. Derived from open-source projects on GitHub, the data consists of (Docstring, Code) pairs, where the query is the function’s docstring and the code is a complete function body.

CoSQA [22]: A dataset containing 20,604 annotated pairs of natural language queries and codes. The queries originate from real user search logs on Microsoft Bing, while the codes are Python functions from the CodeSearchNet dataset. Each pair is annotated by at least 3 human annotators, providing high-quality supervision for query-code matching.

APPS [19]: A challenging benchmark for code generation, measuring the ability of models to take arbitrary natural language specifications and generate satisfactory Python code. It includes 10,000 problems ranging from simple one-line solutions to substantial algorithmic challenges.

Synthetic Text2SQL [37]: A rich dataset of high-quality synthetic Text-to-SQL samples. It includes approximately 106k records covering 100 distinct domains and a wide range of SQL complexity levels, from simple retrieval to complex joins and aggregations.

4.1.2 Models. We conduct experiments using three representative pre-trained models: **UniXcoder** [15], an encoder-only architecture widely adopted in the code search domain; **GTE_{base}** [61], a contemporaneous model that has achieved state-of-the-art (SOTA) results across multiple general retrieval benchmarks; and **Qwen3-Embedding** [62], a foundation-model-based dense encoder. (Additionally, a widely adopted sparse lexical baseline, **BM25** [46], is included for baseline comparison in our evaluation).

UniXcoder: A unified cross-modal pre-trained model for programming languages. To enhance semantic representation, UniXcoder incorporates multi-modal contents such as code comments and ASTs, where the ASTs are flattened via a one-to-one mapping. The model is pre-trained using multi-modal contrastive learning and cross-modal generation tasks to effectively align representations among programming languages and natural language.

GTE_{base}: A general-purpose text embedding model based on the BERT architecture [8], trained via multi-stage contrastive learning [7]. Despite treating code merely as unstructured text, GTE achieves superior performance on code retrieval benchmarks by leveraging robust semantic representations learned from massive text pairs.

Qwen3-Embedding [62]: A state-of-the-art foundation-model-based dense encoder built upon the Qwen3 language model family. In this work, we employ the 0.6B variant, which features a 32k context length and supports instruction-aware text embedding. Notably, Qwen3-Embedding is extensively trained on over 100 languages, including various programming languages, providing robust cross-lingual and code retrieval capabilities.

4.1.3 Implementation. We implement all experiments using PyTorch and the Sentence-Transformers framework. To ensure both reproducibility and fair comparison, our hyperparameter settings are determined through a hybrid approach: aligning with established protocols in prior code retrieval studies [15, 47] and conducting preliminary empirical tuning on the validation set to ensure optimal convergence. Specifically, models are fine-tuned for 6 epochs with a batch size of 4. We utilize the AdamW optimizer with an initial learning rate of $2e-5$ and a linear warmup of 10a configuration widely validated for stabilizing transformer fine-tuning. To optimize semantic alignment, we employ the Multiple Negatives Ranking Loss (MNRL), which effectively leverages in-batch negatives to enhance retrieval distinctiveness. The maximum sequence length is set to 512 tokens to accommodate typical code contexts. For reproducibility, we fix the random seed at 42 across all runs. Our source code and detailed hyperparameters are available at <https://anonymous.4open.science/r/MAP4CS-097A>.

4.2 RQ1: Effectiveness Analysis

In this section, we first quantitatively evaluate the retrieval performance of MAP4CS against both random sampling and full-dataset fine-tuning across multiple benchmarks. Beyond standard performance metrics, we qualitatively investigate the underlying mechanisms of our approach by visualizing the semantic distribution shifts and analyzing the linguistic diversity of the pruned data.

4.2.1 Design. To investigate the effectiveness of MAP4CS, we fine-tune three representative models: UniXcoder, GTE_{base}, and Qwen3-Embedding, on two distinct datasets: CSN-Java and Query4Code. CSN-Java is a subset from the CodeSearchNet corpus. Query4Code is an LLM-annotated dataset specifically constructed to simulate real-world code retrieval scenarios by generating user-style queries rather than using noisy or descriptive docstrings.

We compare our method against two baselines, Random Sampling and Full Data fine-tuning. To contextualize the neural retrieval results, we also report BM25 as a non-parametric lexical baseline. To verify the effectiveness, MAP4CS and Random Sampling use the same data budget, 5%.

In order to evaluate the effectiveness of code search, we use Mean Reciprocal Rank (MRR) [53] as an evaluation metric. MRR is widely adopted in information retrieval (IR) [17] and code search research [47, 57] to assess ranking quality. It measures the mean of the reciprocal ranks of the first relevant code snippet across all queries.

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{rank}_i} \quad (8)$$

Table 1. Performance comparison on downstream code retrieval tasks.

Training Dataset	Method	In-Domain		Out-of-Distribution		Avg.
		CodeSearchNet	CoSQA	APPS	Text2SQL	
Lexical Baseline						
None	BM25	0.360	0.086	0.009	0.025	0.120
UniXcoder						
None	Zero-Shot	0.431	0.122	0.007	0.124	0.171
CSN-Java	Full Data (100%)	0.365	0.092	0.010	0.030	0.124
	Random Sampling (5%)	0.467	0.106	0.019	0.070	0.166
	MAP4CS (5%)	0.613	0.150	0.036	0.076	0.219
Query4Code	Full Data (100%)	0.577	0.140	0.018	0.033	0.192
	Random Sampling (5%)	0.542	0.156	0.010	0.032	0.185
	MAP4CS (5%)	0.573	0.160	0.028	0.037	0.200
GTE _{base}						
None	Zero-Shot	0.200	0.180	0.018	0.127	0.131
CSN-Java	Full Data (100%)	0.126	0.018	0.007	0.060	0.053
	Random Sampling (5%)	0.273	0.066	0.019	0.095	0.113
	MAP4CS (5%)	0.438	0.132	0.028	0.116	0.178
Query4Code	Full Data (100%)	0.447	0.134	0.018	0.043	0.160
	Random Sampling (5%)	0.374	0.146	0.012	0.042	0.144
	MAP4CS (5%)	0.432	0.140	0.027	0.044	0.161
Qwen3-Embedding						
None	Zero-Shot	0.398	0.076	0.079	0.125	0.170
CSN-Java	Full Data (100%)	0.160	0.006	0.002	0.013	0.045
	Random Sampling (5%)	0.260	0.014	0.020	0.058	0.088
	MAP4CS (5%)	0.337	0.066	0.036	0.073	0.128
Query4Code	Full Data (100%)	0.190	0.030	0.004	0.023	0.062
	Random Sampling (5%)	0.210	0.022	0.009	0.034	0.069
	MAP4CS (5%)	0.298	0.040	0.011	0.059	0.102

We follow the same experimental configuration detailed in Section 4.1.3, fine-tuning UniXcoder, GTE_{base}, and Qwen3-Embedding on both datasets with the strict 5% data budget.

4.2.2 Results. Table 1 presents the retrieval performance across four downstream tasks using different fine-tuning strategies. The results provide compelling evidence for the effectiveness of our proposed framework compared to both random sampling and full-dataset training.

Comparison with BM25. BM25 achieves an average score of 0.120. Although it remains competitive on CodeSearchNet (0.360), its performance drops sharply on CoSQA (0.086) and the OOD tasks APPS (0.009) and Text2SQL (0.025). This gap indicates that lexical matching alone is insufficient for capturing user intent and cross-domain semantic alignment in code retrieval. In contrast, dense retrievers generally provide stronger semantic matching, and MAP4CS further improves their fine-tuning efficiency by selecting high-quality training pairs.

Comparison with Random Sampling. Under a strict data budget, MAP4CS consistently outperforms the random sampling baseline across all training datasets and models. Specifically, on the CSN-Java dataset with the UniXcoder model, our method improves the average score from 0.166 (Random 5%) to 0.219, achieving a relative improvement of **31.9%**. Similarly, on the GTE_{base} model, we observe a substantial gain from 0.113 to 0.178 (**+57.5%**). The same trend holds for Qwen3-Embedding: MAP4CS improves the average score from 0.088 to 0.128 on CSN-Java (**+45.5%**) and from 0.069 to 0.102 on Query4Code (**+47.8%**). This demonstrates that our multi-dimensional clustering and filtering strategy successfully identifies high-value core samples, whereas random sampling often captures redundant or unrepresentative data.

Comparison with Full Data Fine-tuning. A striking observation is that our method often matches or even exceeds the performance of fine-tuning on the full dataset (100% data). On the CSN-Java dataset, training with the full dataset yields surprisingly poor results (e.g., Avg 0.053 for GTE_{base}), performing even worse than random sampling. This suggests the original corpus contains severe noise or low-quality pairs that induce negative transfer. By effectively pruning this noise, our method achieves a significantly higher score (0.178), validating its capability as a robust data cleaner. This phenomenon is particularly clear when comparing CSN-Java and Query4Code. CSN-Java is derived from raw GitHub repositories and therefore contains more noise, including comment-code mismatches, duplicated snippets, and boilerplate implementations. Consequently, pruning CSN-Java with MAP4CS yields pronounced gains by removing harmful or redundant samples. In contrast, Query4Code is a retrieval-optimized query-code dataset with higher semantic consistency and fewer expression errors. Since its full-data baseline is already strong, there is less room for large absolute improvements. On the Query4Code dataset, where the data quality is relatively stable, our method still achieves comparable or slightly better performance than full-data fine-tuning (e.g., 0.200 vs. 0.192 on UniXcoder). This confirms that our framework can reduce training costs by over 90% without compromising retrieval accuracy. Therefore, the key value of MAP4CS is not to substantially surpass full-data fine-tuning on every already high-quality dataset, but to achieve comparable or better retrieval accuracy under an extremely low training budget. Such a result effectively validates the “**Less is More**” phenomenon, demonstrating that a small, high-quality subset can serve as a superior training signal compared to a massive but noisy corpus.

Robustness Across Models. The improvements are consistent across the code-specialized (UniXcoder) and general embedding (GTE_{base}) models, extending even to the stronger foundation-model-based Qwen3-Embedding. Notably, on the domain-specific CodeSearchNet evaluation metric, our method achieves a peak score of 0.613 (CSN-Java/UniXcoder), significantly outperforming the pre-trained baseline (0.431) and other fine-tuning strategies. Crucially, on OOD tasks, our method consistently maintains a performance advantage over baselines. For instance, with GTE_{base} trained on Query4Code, MAP4CS improves performance on APPS by **125%** compared to random sampling (0.027 vs. 0.012). For Qwen3-Embedding, MAP4CS also improves every downstream task over both random sampling and full-data fine-tuning under the same training dataset, demonstrating that the proposed pruning strategy remains useful even when the backbone encoder has stronger pre-trained semantic representations. Meanwhile, the competitive zero-shot performance of Qwen3-Embedding suggests that overly noisy or mismatched fine-tuning data can still hurt strong foundation encoders, further motivating quality-aware pruning. This suggests that by maximizing structural and semantic diversity, MAP4CS helps the model learn generalized representations rather than overfitting to specific domain shortcuts.

Finding 1: MAP4CS demonstrates superior effectiveness in code retrieval tasks by outperforming random sampling baselines and achieving comparable or better results than full-dataset fine-tuning, despite using only 5% of the

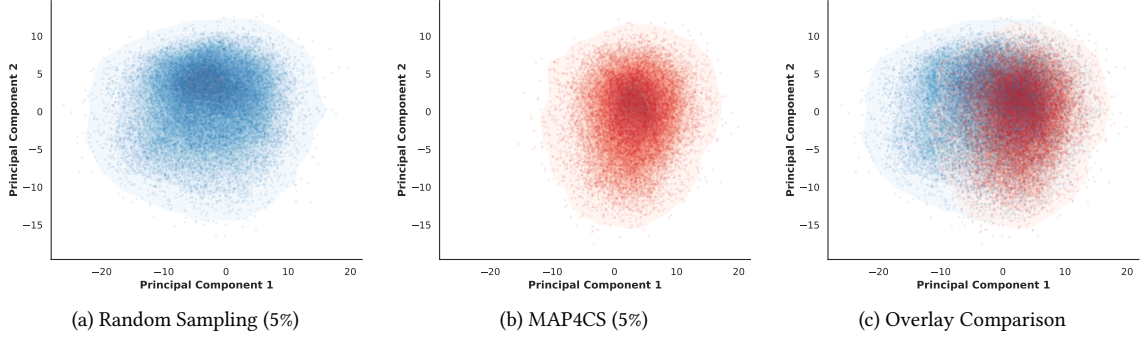


Fig. 4. Visualization of semantic distribution in the Code-Query Semantic Space. The Random Sampling baseline (Blue) exhibits a diffuse distribution with scattered outliers, indicating high-entropy noise. In contrast, MAP4CS (Red) demonstrates a more refined distribution, concentrating on high-value semantic regions.

training data. Moreover, whether in-domain or OOD, our method consistently outperforms random sampling. This highlights the framework’s capability in filtering noise and identifying high-value core samples.

4.2.3 Visualization of Semantic Distribution. To visually verify the effectiveness of MAP4CS in filtering data, we compare the semantic distribution of our pruned subset against the Random Sampling baseline. Specifically, we employ the UniXcoder encoder to extract high-dimensional embeddings from the two subsets generated by MAP4CS and Random Sampling, respectively. These embeddings, derived from the mean pooling of the last hidden state, are then projected into a 2D space using Principal Component Analysis (PCA) [44]. To ensure a fair comparison within the same semantic manifold, we concatenate the vectors from both groups before fitting the PCA model. Furthermore, to address the overplotting issue in scatter plots and to quantify distributional differences, we visualize the distributions using Kernel Density Estimation (KDE) contours, as shown in Figure 4. Simply put, KDE helps visualize data density by smoothing discrete points into continuous contours, allowing us to clearly identify the high-density “core” regions of the semantic space (consistent with Eq. 5).

To quantify the divergence between the distributions, we calculate the **Jensen-Shannon (JS) Distance** [33] and the **Overlap Coefficient** [52]. The Overlap Coefficient measures the overlapping area of two probability distributions (range [0, 1], where higher values indicate greater similarity), while the JS Distance measures the difference based on KL divergence (range [0, 1], where lower values indicate greater similarity). Analysis reveals distinct characteristics in the semantic space. In the **Code-Query Semantic Space**, MAP4CS exhibits a significantly different distribution profile compared to random sampling. The Overlap Coefficient drops to 0.5549, while the JS Distance increases to 0.4895. This divergence supports two key mechanisms:

- **Effective Denoising:** The random sampling distribution (Blue in Figure 4) is characterized by a diffuse spread, mixing high-quality data with a large volume of noisy samples where the code is mismatched or semantically ambiguous relative to the query.
- **Semantic Refinement:** MAP4CS (Red) forms a distinct, compact cluster within the semantic space. Notably, our clusters appear more compact, yet the fine-tuned model performance is superior. This suggests that unlike Random Sampling which captures a diffuse spread of data including outliers (potential noise), MAP4CS may

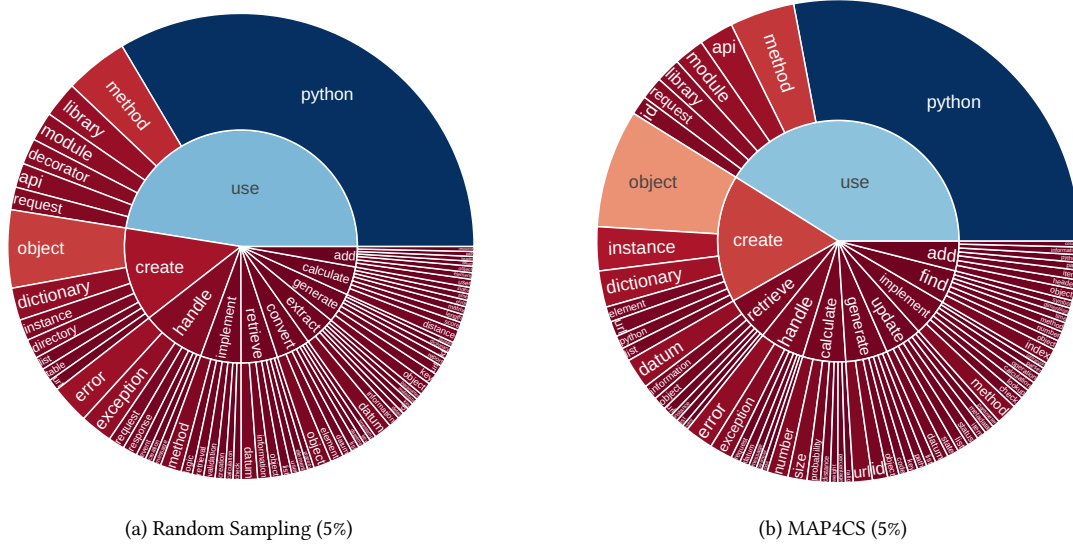


Fig. 5. Sunburst plot of the top 10 most frequent verbs, with their corresponding top 7 root nouns within the query subsets obtained via Random Sampling and MAP4CS.

effectively isolate and retain samples with high semantic density, proving that data quality—defined by semantic consistency—is more critical than mere coverage for training efficient retrievers.

Finding 2: Quantitative analysis of semantic distribution confirms that MAP4CS achieves a “coarse-to-fine” refinement. By shifting the data distribution away from noisy, mismatched regions (low overlap with random baseline) and concentrating on high-consistency semantic clusters, it constructs a significantly purer and more representative training corpus.

4.2.4 Reason Analysis. To investigate the underlying reasons for performance improvement, we analyze the linguistic characteristics of the training data on the Query4Code dataset.

First, we visualize the semantic intent distribution by depicting the most common verbs and their corresponding objects within the query subsets obtained via Random Sampling and MAP4CS (Figure 5). The analysis reveals that while there is some overlap in high-frequency verbs (e.g., “use”, “add”) between random sampling and our method, the pruned dataset exhibits a cleaner and more focused distribution of object terms, reducing long-tail noise.

In addition, we utilize quantitative metrics to measure the quality of the data. On one hand, our method increases the Type-Token Ratio (TTR) from 0.0402 (Random Sampling) to 0.0579 (MAP4CS). Since TTR represents the ratio of unique words to total words, this substantial increase indicates that MAP4CS effectively performs “de-duplication” and “quality enhancement,” filtering out repetitive boilerplate code. On the other hand, the Shannon Entropy slightly decreases from 6.269 (Random Sampling) to 6.160 (MAP4CS), a drop of approximately 1.7%. While entropy represents uncertainty, high entropy in random sampling often stems from noise and long-tail confusion. Our clustering-based approach acts as a denoising mechanism, making the data distribution more representative.

Table 2. Ablation study of different components in our framework. The best results among fine-tuned models are highlighted in **bold**.

Method	In-Domain		Out-of-Distribution		Avg.
	CodeSearchNet	CoSQA	APPS	Text2SQL	
GTE _{base} (Training Dataset: Query4Code)					
Random Sampling (5%)	0.374	0.146	0.012	0.042	0.144
Clustering Only (5%)	0.401	0.146	0.018	0.041	0.152
MAP4CS (5%)	0.432	0.140	0.027	0.044	0.161

Finding 3: Compared to random sampling, MAP4CS effectively filters out high-entropy noise while simultaneously achieving higher lexical diversity, resulting in an information-dense training set.

4.3 RQ2: Ablation Study

In this section, we dissect the contribution of each component within MAP4CS to understand the source of its effectiveness.

4.3.1 Design. To isolate the impact of different pruning stages, we conduct an ablation study using the GTE_{base} model on the Query4Code dataset. We compare the following settings (Clustering Only and MAP4CS) against the Random Sampling fine-tuning baseline. Clustering Only applies random sampling within the clusters generated by Single-pass Clustering to select 5% of the data.

4.3.2 Results. Table 2 summarizes the results of the ablation study. Clustering Only achieves an average score of 0.152, showing a clear improvement over the Random Sampling baseline (0.144). This confirms that clustering-based sampling captures the semantic skeleton and diversity of the original dataset more effectively than pure random sampling. By ensuring coverage across different semantic clusters, the model avoids overfitting to dominant patterns.

Integrating the multi-dimensional sorting and filtering mechanism (MAP4CS) further boosts the performance to 0.161, demonstrating that quality filtering is as important as diversity. The improvement from Clustering Only (0.152) to MAP4CS (0.161) highlights the value of our scoring mechanism. By filtering out low-quality samples within clusters (e.g., mismatched query-code pairs or low-density outliers), MAP4CS ensures that the model trains on the most representative data. Notably, on the challenging APPS task, our method achieves a score of 0.027, significantly outperforming both Clustering Only (0.018) and Random Sampling (0.012). This indicates that high-quality, denoised training data is crucial for the model to generalize well on complex code search tasks, rather than merely memorizing patterns.

Finding 4: While semantic clustering provides a strong foundation for data diversity, the multi-dimensional filtering and scoring mechanism is the key driver of performance. It eliminates harmful noise and retains high-quality samples, enabling MAP4CS to surpass both random and clustering-based baselines, especially on complex generalization tasks.

4.4 RQ3: Parameter Analysis

In this section, we investigate the trade-off between data scale and model performance, and further analyze the linguistic properties of the sampled subsets to explain the efficiency of our method.

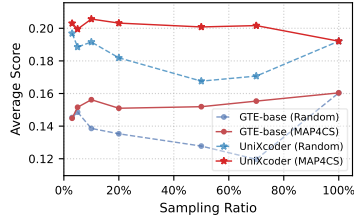


Fig. 6. Performance Trend across Sampling Ratios

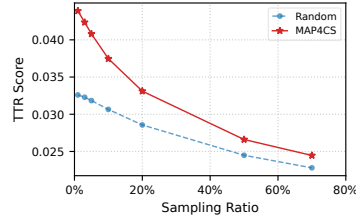


Fig. 7. TTR Trend

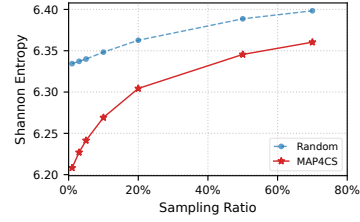


Fig. 8. Shannon Entropy Trend

4.4.1 Design. To answer RQ3, we conduct a sensitivity analysis on the Query4Code dataset. We vary the sampling ratio from 3% to 100% (specifically {3%, 5%, 10%, 20%, 50%, 70%, 100%}) and evaluate the performance of both UniXcoder and GTE_{base}. We compare our MAP4CS against the Random Sampling baseline to assess data efficiency and robustness.

4.4.2 Results. Figure 6 illustrates the performance trends. MAP4CS demonstrates exceptional efficiency in low-data scenarios. Even with as little as 3% of the training data, our pruned subset consistently outperforms the random sampling baseline. Notably, on UniXcoder, the model trained with just 3% of our pruned data surpasses the model trained with 50% randomly sampled data. This validates our framework’s ability to prioritize the most information-dense samples.

A counter-intuitive phenomenon observed in the random sampling baseline is the performance volatility—and even degradation—as the data ratio increases (e.g., from 5% to 70%). This suggests that blindly adding more data introduces detrimental noise. In contrast, our method maintains a stable and superior performance trajectory, confirming that our multi-dimensional filtering acts as an effective “firewall” against long-tail noise.

Finding 5: MAP4CS achieves peak performance with minimal data (~10%) and exhibits superior stability compared to Random Sampling, which may suffer from performance degradation due to noise accumulation as data scale increases.

4.4.3 Reason Analysis. To understand why our method achieves superior performance with less data, we analyze the evolution of linguistic quality metrics across different sampling ratios, as shown in Figure 7 and Figure 8.

Figure 7 reveals a significant “cold-start” advantage for our method. In the low-resource regime (Ratio < 10%), the pruned subset maintains a substantially higher TTR compared to random sampling. Although TTR naturally decays as the corpus size grows, the persistent gap highlights that MAP4CS prioritizes non-redundant samples. This implies that the model is exposed to a broader spectrum of lexical and syntactic patterns much earlier in the training process.

As shown in Figure 8, the random sampling baseline consistently exhibits slightly higher entropy values compared to our pruned subset. While higher entropy typically implies more information, in the context of raw code corpora, it often stems from high-entropy noise. The slightly lower but stable entropy of our method indicates a successful denoising effect.

By filtering out chaotic, high-entropy elements while enhancing lexical diversity, MAP4CS constructs a more regularized and information-dense training corpus, which explains its superior efficiency.

Finding 6: Linguistic analysis confirms that the efficiency of MAP4CS stems from its ability to maximize lexical diversity (High TTR) while filtering out high-entropy noise, effectively solving the trade-off between data scale and quality.

5 Threats to Validity

Internal Validity. One potential threat to internal validity is the selection of hyperparameters, which can significantly impact model performance. To mitigate this, we maintain consistent hyperparameters for all model fine-tuning experiments, ensuring consistency and fairness in our evaluations. Additionally, there is a risk of data contamination; pre-trained models like UniXcoder might have been exposed to the CodeSearchNet corpus during their initial pre-training phase, which could artificially inflate evaluation results on in-domain datasets. To mitigate this concern and verify the true generalization capability of our method, we incorporate APPS and Synthetic Text2SQL as out-of-distribution (OOD) benchmarks. These datasets represent distinct tasks with vastly different data distributions, ensuring that the observed performance gains stem from the high-quality pruned data rather than prior data memorization.

External Validity. The generalizability of our findings might be limited by the selection of retrieval models and datasets. To mitigate model-specific biases, we evaluate three diverse dense neural architectures: UniXcoder (code-specialized), GTE_{base} (general-purpose), and Qwen3-Embedding (foundation-model-based), alongside BM25 as a sparse lexical baseline. This broad model coverage minimizes the risk that our findings are biased toward a specific retrieval mechanism. To ensure robustness across data distributions, we conduct experiments on datasets with varying noise levels and construction paradigms, specifically CSN-Java (raw open-source repositories) and Query4Code (LLM-optimized queries). Furthermore, to address potential bias towards specific languages, we extend our evaluation using the CodeSearchNet and Synthetic Text2SQL benchmarks, which collectively cover seven programming languages (Go, Java, JavaScript, PHP, Python, Ruby, and SQL). This multi-lingual evaluation demonstrates that the effectiveness of MAP4CS is not confined to a single programming language but is applicable across diverse software ecosystems.

A final threat to external validity involves the gap between academic benchmarks and real-world industrial code search. Large-scale enterprise search logs are strictly confidential due to trade secrets, proprietary business logic, and sensitive development contexts, making it difficult to fully replicate industrial deployment conditions. Nevertheless, our benchmark suite partially bridges this gap: notably, CoSQA consists of real user queries sampled from Microsoft Bing search logs, effectively capturing realistic developer intents and natural-language query styles. We leave the validation of MAP4CS on anonymized industrial search logs as a direction for future work.

6 Related Work

6.1 Neural Code Retrieval

Code retrieval aims to identify the most relevant code snippet from a corpus given a natural language query. Early approaches primarily rely on keyword matching techniques, such as TF-IDF [26] and BM25 [46]. With the advent of deep learning, neural code retrieval has become the dominant paradigm [13]. Many modern approaches employ a bi-encoder architecture, where code and queries are mapped into a shared vector space [13].

Pre-trained models like CodeBERT [11], GraphCodeBERT [16], and UniXcoder [15] employ masked language modeling and cross-modal tasks to learn robust representations, significantly advancing the state-of-the-art. Building on this, researchers have introduced code-specific optimization strategies: ContraCode [25] leverages compiler-generated code variants for contrastive learning to capture functional semantics; CodeRetriever [32] enforces code-text alignment through unimodal and bimodal contrastive losses; and CodeT5 [55] adopts identifier-aware pre-training tasks, excelling in both generation and retrieval. Recently, general-purpose text embeddings like GTE [61] have also shown promise in code domains.

While these works focus on designing better model architectures or pre-training objectives, they typically train on massive, unfiltered corpora. In contrast, our work focuses on the data-centric perspective, investigating how to efficiently fine-tune these strong backbones using a small but highly representative subset of data.

6.2 Data Pruning and Selection

Data pruning aims to select a high-quality subset of training data without compromising model performance. This field has evolved significantly across different domains.

General NLP and LLM Fine-tuning. In general NLP, strategies range from heuristic rules (e.g., filtering based on length, perplexity, or stop-word ratios) [56] to sophisticated metric-based approaches. With the rise of Large Language Models (LLMs), the focus has shifted to Instruction Tuning data selection. The “Less Is More for Alignment” (LIMA) hypothesis [65] demonstrates that a handful of carefully curated examples can outperform massive datasets. Subsequent works like AlpaGasus [4] and MoDS [9] leverage powerful teacher models to score and filter data based on quality, complexity, and diversity. Other approaches leverage training dynamics, such as Influence Functions [29] or Forgetting Events [51], to identify samples that contribute most to model generalization. However, these methods typically treat data as linear text sequences, ignoring the rigid syntactic structures inherent in programming languages.

Data Selection for General Retrievers. In the broader field of information retrieval (e.g., web search, QA), data strategies have primarily focused on data augmentation or negative sampling rather than positive pruning for efficiency. For instance, techniques like ANCE [58] and RocketQA [45] mine “hard negatives” to sharpen decision boundaries. While effective, these approaches essentially *increase* the training load to improve performance. Research on pruning *positive* training samples to reduce computational costs remains sparse in the retrieval domain. Moreover, existing general selection methods do not account for the specific semantic gap between code logic and documentation. As a result, directly applying general text pruning metrics to code retrieval often yields sub-optimal results.

Code-Specific Data Cleaning. Data cleaning in software engineering has evolved from simple preprocessing to sophisticated quality assurance pipelines. *Deduplication* is a foundational step; Allamanis [1] demonstrates that code duplication artificially inflates performance metrics and causes data leakage. Consequently, modern large-scale code datasets, such as The Stack [28] and StarCoder [31], implement aggressive Near-deduplication (e.g., MinHash) and PII redaction as standard protocols.

Beyond deduplication, recent research focuses on filtering low-quality code using *dynamic* or *static* signals. On the dynamic side, execution feedback is leveraged to verify functional correctness. For instance, Shi et al. [47] and Han et al. [18] harness test case execution results to align code retrieval models with ground-truth correctness, filtering out snippets that fail to compile or pass tests. However, execution-based methods are computationally expensive and hard to scale. Alternatively, static approaches rely on heuristic rules (e.g., line length, parser checks) or learned metrics. Gunasekar et al. [14] introduce a classifier-based approach to select “textbook-quality” code, significantly boosting model performance with less data. Similarly, embedding-based pruning strategies [48] have been explored to remove outliers in the latent space.

Despite these advancements, existing code pruning methods often rely on *single-dimensional* metrics—either focusing solely on syntactic validity, execution outcomes, or distinctive semantic features [29, 56]. Our framework, **MAP4CS**, addresses this limitation by introducing a *multi-dimensional* perspective. We integrate semantic consistency, syntactic structure (AST), and distributional density to perform a holistic assessment of data value, specifically tailored for retrieval-augmented generation tasks.

6.3 Retrieval-Augmented Code Generation (RACG)

RACG enhances Large Language Models (LLMs) by retrieving relevant context from external knowledge bases, thereby mitigating hallucinations and adapting to private domains. Recently, RACG has been widely applied across various SE tasks.

RACG has been widely adopted across tasks such as code completion [36, 60], in-context learning [40], and domain-specific applications including vulnerability detection [10], library generation [34], and translation [42]. Recent studies also investigate retrieval granularity [12] to optimize generation.

Despite the diverse applications of RACG, its performance relies heavily on the quality of the retriever. While extensive research focuses on optimizing the generator, expanding applications, or refining inference-time retrieval strategies (e.g., re-ranking) [54], the efficiency of *adapting* the retriever itself remains under-explored. Maintaining an effective retriever for rapidly evolving codebases requires frequent updates. Our work addresses the training efficiency bottleneck in RACG by demonstrating that high-performance retrievers can be trained with significantly less data, making continuous adaptation feasible for resource-constrained environments.

7 Conclusion

In this paper, we address the critical challenge of efficient and robust retriever fine-tuning for RAG in software engineering. We propose **MAP4CS**, an adaptive multi-dimensional data pruning framework specifically tailored for code search tasks. By selecting a small, high-quality core subset from massive, noisy code corpora, MAP4CS innovatively integrates syntactic structure, semantic consistency, and distributional representation to evaluate sample value, and employs a rigorous rule-based filtering pipeline to ensure data uniqueness and engineering quality.

Our experiments demonstrate that MAP4CS consistently outperforms random sampling baselines across both in-domain and out-of-domain benchmarks. Remarkably, using only ~5% of the training data, our method achieves performance comparable or even superior to fine-tuning on the full dataset. This empirical evidence strongly supports the “less is more” hypothesis in data-centric AI: data quality and density are more critical than sheer volume for effective domain adaptation.

Furthermore, our linguistic analysis reveals an adaptive optimization mechanism inherent in MAP4CS. We find that the framework automatically acts as a *de-duplicator* for redundant corpora and a *denoiser* for chaotic corpora, effectively solving the cold-start problem in low-resource regimes. This dual capability allows it to construct a training corpus that is both lexically diverse and information-dense. In future work, we plan to extend MAP4CS to larger-scale decoder-only LLMs and investigate iterative pruning strategies for continuous learning scenarios.

8 Data availability

To facilitate the replication study, we have released our data and code at: <https://anonymous.4open.science/r/MAP4CS-097A>.

References

- [1] Miltiadis Allamanis. 2019. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2019, Athens, Greece, October 23-24, 2019*. ACM, 143–153. doi:10.1145/3359591.3359735
- [2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.* 3, POPL (2019), 40:1–40:29. doi:10.1145/3290353

- [3] Inas Bachiri, Hadjer Benmeziene, Smail Niar, Riyadh Baghdadi, Hamza Ouarnoughi, and Abdelkrime Aries. 2024. Combining Neural Architecture Search and Automatic Code Optimization: A Survey. *arXiv abs/2408.04116* (2024). doi:10.48550/ARXIV.2408.04116
- [4] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. AlpaGasus: Training a Better Alpaca with Fewer Data. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. <https://openreview.net/forum?id=FdVXgSJhvz>
- [5] Yuxuan Chen, Guangsheng Ou, Mingwei Liu, Yanlin Wang, and Zibin Zheng. 2024. Are Decoder-Only Large Language Models the Silver Bullet for Code Search? *arXiv abs/2410.22240* (2024). doi:10.48550/ARXIV.2410.22240
- [6] Ching-Yao Chuang, R. Devon Hjelm, Xin Wang, Vibhav Vineet, Neel Joshi, Antonio Torralba, Stefanie Jegelka, and Yale Song. 2022. Robust Contrastive Learning against Noisy Views. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 16649–16660. doi:10.1109/CVPR52688.2022.01617
- [7] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Association for Computational Linguistics, 670–680. doi:10.18653/V1/D17-1070
- [8] 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. doi:10.18653/V1/N19-1423
- [9] Qianlong Du, Chengqing Zong, and Jiajun Zhang. 2023. MoDS: Model-oriented Data Selection for Instruction Tuning. *arXiv abs/2311.15653* (2023). doi:10.48550/ARXIV.2311.15653
- [10] Xueying Du, Geng Zheng, Kaixin Wang, Jiayi Feng, Wentai Deng, Mingwei Liu, Bihuan Chen, Xin Peng, Tao Ma, and Yiling Lou. 2024. Vul-RAG: Enhancing LLM-based Vulnerability Detection via Knowledge-level RAG. *arXiv abs/2406.11147* (2024). doi:10.48550/ARXIV.2406.11147
- [11] 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, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*. Association for Computational Linguistics, 1536–1547. doi:10.18653/V1/2020.FINDINGS-EMNLP.139
- [12] Wenchao Gu, Juntao Chen, Yanlin Wang, Tianyue Jiang, Xingzhe Li, Mingwei Liu, Xilin Liu, Yuchi Ma, and Zibin Zheng. 2025. What to Retrieve for Effective Retrieval-Augmented Code Generation? An Empirical Study and Beyond. *arXiv abs/2503.20589* (2025). doi:10.48550/ARXIV.2503.20589
- [13] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 933–944. doi:10.1145/3180155.3180167
- [14] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need. *arXiv abs/2306.11644* (2023). doi:10.48550/ARXIV.2306.11644
- [15] 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), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 7212–7225. doi:10.18653/V1/2022.ACL-LONG.499
- [16] 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 B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. <https://openreview.net/forum?id=jLoC4ez43PZ>
- [17] Kailash A. Hambarde and Hugo Proença. 2023. Information Retrieval: Recent Advances and Beyond. *IEEE Access* 11 (2023), 76581–76604. doi:10.1109/ACCESS.2023.3295776
- [18] Hojae Han, Minsoo Kim, Seung-won Hwang, Nan Duan, and Shuai Lu. 2023. Intervention-Based Alignment of Code Search with Execution Feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023 (Findings of ACL, Vol. EMNLP 2023)*. Association for Computational Linguistics, 2241–2263. doi:10.18653/V1/2023.FINDINGS-EMNLP.148
- [19] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html>
- [20] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar T. Devanbu. 2012. On the naturalness of software. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 837–847. doi:10.1109/ICSE.2012.6227135
- [21] Daniel S. Hirschberg. 1975. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Commun. ACM* 18, 6 (1975), 341–343. doi:10.1145/360825.360861
- [22] 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, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 5690–5700. doi:10.18653/V1/2021.ACL-LONG.442

- [23] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv abs/1909.09436* (2019). <http://arxiv.org/abs/1909.09436>
- [24] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* 2022 (2022). <https://openreview.net/forum?id=jKN1pXi7b0>
- [25] Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph Gonzalez, and Ion Stoica. 2021. Contrastive Code Representation Learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 5954–5971. doi:10.18653/V1/2021.EMNLP-MAIN.482
- [26] Amir Jalilifard, Vinicius Fernandes Caridá, Alex Mansano, and Rogers Cristo. 2020. Semantic Sensitive TF-IDF to Determine Word Relevance in Documents. *arXiv abs/2001.09896* (2020). <https://arxiv.org/abs/2001.09896>
- [27] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *arXiv abs/2001.08361* (2020). <https://arxiv.org/abs/2001.08361>
- [28] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2023. The Stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research* 2023 (2023). <https://openreview.net/forum?id=pxpbTdUEpD>
- [29] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*. Proceedings of Machine Learning Research, 1885–1894. <http://proceedings.mlr.press/v70/koh17a.html>
- [30] 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. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>
- [31] 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 V, Jason T. 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! *Transactions on Machine Learning Research* 2023 (2023). <https://openreview.net/forum?id=KoFog41haE>
- [32] Xiaonan Li, Yeyun Gong, Yelong Shen, Xipeng Qiu, Hang Zhang, Bolun Yao, Weizhen Qi, Daxin Jiang, Weizhu Chen, and Nan Duan. 2022. CodeRetriever: A Large Scale Contrastive Pre-Training Method for Code Search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Association for Computational Linguistics, 2898–2910. doi:10.18653/V1/2022.EMNLP-MAIN.187
- [33] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Trans. Inf. Theory* 37, 1 (1991), 145–151. doi:10.1109/18.61115
- [34] Mingwei Liu, Tianyong Yang, Yiling Lou, Xueying Du, Ying Wang, and Xin Peng. 2023. CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 434–445. doi:10.1109/ASE56229.2023.00159
- [35] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv abs/2402.19173* (2024). doi:10.48550/ARXIV.2402.19173
- [36] Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. 2022. ReACC: A Retrieval-Augmented Code Completion Framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 6227–6240. doi:10.18653/V1/2022.ACL-LONG.431
- [37] Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. 2024. *Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate SQL queries from natural language prompts*. <https://huggingface.co/datasets/gretelai/synthetic-text-to-sql>
- [38] Baharan Mirzasoileiman, Jeff A. Biles, and Jure Leskovec. 2020. Coresets for Data-efficient Training of Machine Learning Models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. Proceedings of Machine Learning Research, 6950–6960. <http://proceedings.mlr.press/v119/mirzasoileiman20a.html>

- [39] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, 1287–1293. doi:10.1609/AAAI.V30I1.10139
- [40] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2450–2462. doi:10.1109/ICSE48619.2023.00205
- [41] Hai-Long Nguyen, Yew-Kwong Woon, and Wee Keong Ng. 2015. A survey on data stream clustering and classification. *Knowl. Inf. Syst.* 45, 3 (2015), 535–569. doi:10.1007/S10115-014-0808-1
- [42] Guangsheng Ou, Mingwei Liu, Yuxuan Chen, Xueying Du, Shengbo Wang, Zekai Zhang, Xin Peng, and Zibin Zheng. 2025. Enhancing LLM-based Code Translation in Repository Context via Triple Knowledge-Augmented. *arXiv abs/2503.18305* (2025). doi:10.48550/ARXIV.2503.18305
- [43] Md. Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021 (Findings of ACL, Vol. EMNLP 2021)*. Association for Computational Linguistics, 2719–2734. doi:10.18653/V1/2021.FINDINGS-EMNLP.232
- [44] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.
- [45] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Association for Computational Linguistics, 5835–5847. doi:10.18653/V1/2021.NAACL-MAIN.466
- [46] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3, 4 (2009), 333–389. doi:10.1561/15000000019
- [47] Ensheng Shi, Yanlin Wang, Wenchao Gu, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2023. CoCoSoDa: Effective Contrastive Learning for Code Search. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2198–2210. doi:10.1109/ICSE48619.2023.00185
- [48] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. http://papers.nips.cc/paper_files/paper/2022/hash/7b75da9b61eda40fa35453ee5d077df6-Abstract-Conference.html
- [49] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 3645–3650. doi:10.18653/V1/P19-1355
- [50] Zhensu Sun, Li Li, Yan Liu, Xiaoning Du, and Li Li. 2022. On the Importance of Building High-quality Training Datasets for Neural Code Search. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 1609–1620. doi:10.1145/3510003.3510160
- [51] Mariya Toneva, Alessandro Sordani, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. 2019. An Empirical Study of Example Forgetting during Deep Neural Network Learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. <https://openreview.net/forum?id=BJlxm30cKm>
- [52] MK Vijaymeena and K Kavitha. 2016. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal* 3, 2 (2016), 19–28.
- [53] Ellen M. Voorhees. 1999. The TREC-8 Question Answering Track Report. In *Proceedings of The Eighth Text REtrieval Conference, TREC 1999, Gaithersburg, Maryland, USA, November 17-19, 1999 (NIST Special Publication, Vol. 500-246)*. National Institute of Standards and Technology (NIST). http://trec.nist.gov/pubs/trec8/papers/qa_report.pdf
- [54] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 1069–1088. doi:10.18653/V1/2023.EMNLP-MAIN.68
- [55] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 8696–8708. doi:10.18653/V1/2021.EMNLP-MAIN.685
- [56] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data. In *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*. European Language Resources Association, 4003–4012. <https://aclanthology.org/2020.lrec-1.494/>
- [57] Yutao Xie, Jiayi Lin, Hande Dong, Lei Zhang, and Zhonghai Wu. 2024. Survey of Code Search Based on Deep Learning. *ACM Transactions on Software Engineering and Methodology* 33, 2 (2024), 54:1–54:42. doi:10.1145/3628161
- [58] 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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. <https://openreview.net/forum?id=zeFrgyZln>

- [59] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM* 64, 3 (2021), 107–115. doi:10.1145/3446776
- [60] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 2471–2484. doi:10.18653/V1/2023.EMNLP-MAIN.151
- [61] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - Industry Track, Miami, Florida, USA, November 12-16, 2024*. Association for Computational Linguistics, 1393–1412. doi:10.18653/V1/2024.EMNLP-INDUSTRY.103
- [62] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv abs/2506.05176* (2025). <https://doi.org/10.48550/arXiv.2506.05176>
- [63] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends. *arXiv abs/2311.10372* (2023). doi:10.48550/ARXIV.2311.10372
- [64] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an Understanding of Large Language Models in Software Engineering Tasks. *Empirical Software Engineering* 30, 2 (2025), 50. doi:10.1007/S10664-024-10602-0
- [65] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less Is More for Alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/ac662d74829e4407ce1d126477f4a03a-Abstract-Conference.html