BEYOND SINGLE POSITIVES: AN EMPIRICAL AUDIT OF SOLUTION-SET ADAPTATION FOR CODE RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Competitive-programming problems provide multiple accepted solutions and rejected submissions for the same task. We audit whether adapting code encoders with this solution-set supervision improves retrieval beyond a single-positive baseline and the original pretrained encoder without adaptation (the frozen encoder). We evaluate MPR-Hybrid (Multi-Positive Retrieval Hybrid), a training system that trains language-specific LoRA adapters with a Multi-Positive Pooled Contrastive (MPPC) loss, linking each problem query to multiple accepted solutions and sampled rejected submissions selected from candidate pools using submission verdicts and embedding similarity. We compare it with a single-positive in-batch contrastive-learning system trained with the InfoNCE loss, using the other examples in each batch as negatives on six encoder backbones, trained with CodeContests supervision and evaluated on its separate test set, with zero-shot CodeSearchNet transfer, a fixed pool of code candidates selected by keyword matching (BM25), including lexically similar incorrect distractors, to test retrieval when keyword overlap is misleading, and CodeXGLUE AdvTest. We also evaluate the original pretrained encoders without fine-tuning as frozen references on the available CodeContests and CodeSearchNet evaluations, to assess whether adaptation improves retrieval beyond the pretrained representation. The observed benefits vary across encoder backbones and evaluation protocols. On the fixed CodeContests official-test protocol for unseen problems, MPR-Hybrid has higher average scores for some backbones, but no result meets our two-metric confirmation rule for a MPR-Hybrid win. In zero-shot CodeSearchNet Java transfer, where candidate code almost never contains the query text, MPR-Hybrid beats the single-positive baseline on three of six backbones, but none improves on its frozen encoder under the same two-metric rule. On Python, every protocol with a MPR-Hybrid win also contains the query text verbatim in candidate documentation. Removing documentation and comments leaves two confirmed MPR-Hybrid gains over the single-positive baseline: i) one other confirmed win becomes a loss, one formerly inconclusive comparison becomes a loss, and ii) two confirmed wins become inconclusive. The audit shows how several apparent gains change interpretation under frozen-encoder comparisons and documentation removal, while identifying settings where gains over the single-positive baseline remain. The comparison evaluates complete adaptation systems and therefore does not independently isolate the effects of the loss function, candidate sampling, or which encoder layers receive the LoRA adapters.

1 INTRODUCTION

Text-to-code retrieval typically treats a query and a single code item as a positive pair. In competitive programming, however, a problem may have multiple accepted submissions with different implementations, while the same problem may also have rejected submissions. CodeContests provides accepted and rejected examples with this structure (Li et al., 2022). This structure motivates training with multiple positives and negatives selected using submission verdicts. We audit whether this richer supervision improves retrieval. Beating another trained adapter alone does not establish improvement over the frozen pretrained encoder.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We evaluate MPR-Hybrid as a solution-set adaptation system, which trains language-specific LoRA adapters on prioritized candidate pools built from submission outcomes and uses a Multi-Positive Pooled Contrastive (MPPC) loss to associate each problem query with multiple accepted solutions while sampling rejected submissions. As the comparison baseline, a single-positive in-batch contrastive-learning system is trained with the InfoNCE loss, where the remaining examples in each batch provide negatives. This baseline closely follows the attention-targeted LoRA contrastive-training design used by LoRACode (Chaturvedi et al., 2025). Within each language, both systems adapt the same pretrained backbone. The single-positive baseline targets attention projections, whereas MPR-Hybrid targets MLP projections except for CodeBERT C++. Recent controlled LoRA experiments motivate this MLP-targeted choice by finding that attention-only adaptation can underperform MLP-only or all-layer adaptation (Schulman, 2025). The MPR-Hybrid design draws on complementary embedding-training ideas: CoRNStack's curated contrastive data and mined hard negatives, CoCoHaNeRe's searchable memory bank for hard-negative retrieval, and Nomic Embed's reproducible contrastive-training recipe (Suresh et al., 2025; Fan et al., 2025; Nussbaum et al., 2025). We adapt these ideas to verdict-labeled solution sets by combining multiple accepted positives with tiered rejected candidates. This LoRA placement is one of several training configuration differences, so we evaluate the complete systems rather than attributing their difference to one loss term. Appendix Section A.3 provides the full configuration, including the available trainable parameter counts for both setups.

Our evaluation protocol covers six model backbones, namely CodeBERT (Feng et al., 2020), UniXcoder (Guo et al., 2022), CodeSage-Small (CodeSage, 2024), CodeRankEmbed (Suresh et al., 2025), Jina Embeddings v2 Base Code (Jina AI, 2024), and EmbeddingGemma 300M (Google, 2025). When the required data and official adapter pairs are available, both systems are tested on unseen CodeContests problems. The trained adapters are reused without additional training on CodeSearchNet (Husain et al., 2019), a general code-search benchmark, to ask whether training on competitive-programming data transfers to another code-search setting. For Python, two stress tests are added, i.e. a fixed set of code candidates chosen independently by keyword matching (BM25), and CodeXGLUE AdvTest (Lu et al., 2021), which compares clean code with masked identifiers' versions. For each comparison, both systems' scores and paired query-bootstrap confidence intervals for their difference are reported. We also evaluate the frozen pretrained encoders on CodeContests and CodeSearchNet to check whether either training system improves retrieval beyond the model before adaptation.

RQ1. Does adaptation using solution sets improve retrieval on general code-search benchmarks compared with the single-positive baseline and the frozen pretrained encoder?

RQ2. How consistently do the relative benefits hold on unseen problems, keyword-matched candidates, and identifier-obfuscated code?

RQ3. How do retrieval gains change when documentation and comments are removed from candidate code?

We investigate RQ3 on Python, comparing retrieval before and after removing documentation and comments.

The third question matters because a retrieval gain may reflect matching a query to candidate documentation rather than stronger code understanding. Every Python protocol in which MPR-Hybrid shows a confirmed win places the query text verbatim inside candidate code for most or all evaluation queries, while held-out CodeContests shows no confirmed MPR-Hybrid win and CodeSearchNet Java, which has almost no query containment, gives mixed results. Stripping documentation and comments from the candidate code with an AST-verified transform changes four of the six outcomes in a direction that is less favorable to MPR-Hybrid, turning one confirmed win into a confirmed loss, turning one inconclusive difference into a confirmed loss, and removing two other confirmed wins, while two confirmed gains remain. After removing documentation and comments, both adapted systems score close to the frozen encoder for CodeSage-Small, CodeRankEmbed, Jina v2 Code, and EmbeddingGemma. For these backbones, the scores indicate little change from adaptation. Our contributions are a six-backbone audit of solution-set adaptation, comparisons that distinguish gains over another adapter from gains over frozen encoders, and a documentation-removal analysis that tests the stability of Python transfer findings. MPR-Hybrid provides the training construction examined in this study.

2 Related Work

Pretrained code encoders provide a common starting point for code retrieval. CodeBERT (Feng et al., 2020) and UniXcoder (Guo et al., 2022) learn shared representations of code and natural-language descriptions. LoRA (Hu et al., 2022) adapts such encoders by training small low-rank updates while keeping the base weights fixed. LoRACode (Chaturvedi et al., 2025) applies this approach to code retrieval with language-specific adapters and in-batch contrastive training. Our setting follows this adapter-based retrieval pattern but changes the supervision: each query can have several accepted solutions and rejected submissions.

Contrastive-learning work has also improved retrieval by changing the training data and the negative examples. Supervised contrastive learning (Khosla et al., 2020) provides a general formulation for multiple positives. CoRNStack (Suresh et al., 2025) studies curated contrastive data and mined hard negatives, while Fan et al. (Fan et al., 2025) examine hard-negative mining for code search. OASIS (Gao et al., 2025) adds ordering information to the training signal, and Nomic Embed (Nussbaum et al., 2025) describes a reproducible contrastive-training recipe. These studies motivate our use of multiple accepted solutions and selected negative examples. We examine this combination using CodeContests submission outcomes and evaluate how its retrieval performance changes across backbones and evaluation settings.

Code-retrieval benchmarks differ in task structure and difficulty. CodeSearchNet (Husain et al., 2019) and CoSQA (Huang et al., 2021) provide text-to-code pairs, while CoSQA+ (Gong et al., 2024) adds multiple choices and XLCoST (Zhu et al., 2022) supports cross-lingual code search. CoIR (Li et al., 2025) broadens evaluation across code-information-retrieval tasks. More targeted tests include the identifier-obfuscation setting in CodeXGLUE AdvTest (Lu et al., 2021), the adversarial C/C++ benchmark CLARC (Wang et al., 2026), the multitask benchmark and explicit distractor construction in CoREB (Xue et al., 2026), and the lexical BM25 baseline (Robertson & Zaragoza, 2009). CodeContests, introduced for competition-level code generation (Li et al., 2022), supplies accepted and rejected submissions for the same problem. The existing retrieval benchmarks above generally do not combine multiple accepted implementations with submission-verdict labels for one competitive-programming problem. Our study uses this structure to train the adapters and then tests both in-domain retrieval and transfer to general code search.

3 Method

3.1 CodeContests data and negative sampling

For a natural-language problem q, CodeContests provides each submission with a problem identifier (p_{id}), programming language identifier (l_{id}), solution identifier (s_{id}), and verdict (Li et al., 2022). We construct separate language-specific training examples for each programming language. Accepted submissions for a given problem and language form the positive set $P(q)$ while rejected submissions provide the negative candidate pool from which the training negative set $N(q)$ is sampled. CodeContests provides separate training, validation, and test partitions, which we use for training, checkpoint selection, and final evaluation, respectively.

3.1.1 Offline metadata and candidate-pool construction

Before training, we prepare separate metadata for the training and validation partitions. These saved indexes link each problem and language to accepted solutions, rejected submissions, and eligible candidates from other problems. Training reads these indexes instead of repeating corpus-wide searches at each update. Appendix Section A.1 gives the construction steps and saved records.

3.1.2 Tiered negative sampling

For the MPR-Hybrid system, the tiered sampler reads the indexed candidate pools prepared in the preceding metadata step. It then draws negatives from three tiers: Tier 1 uses rejected submissions for the same problem in the target language. Tier 2 uses rejected submissions for different problems in that language selected by a query-embedding similarity band. Tier 3 uses rejected submissions for the same problem in another supported language. The offline Tier-2 cache keeps up to 50 different

problem candidates whose query-embedding similarity is at least 0.30 and below 0.90. These lower and upper bounds define candidate eligibility, not algorithmic equivalence. Each training group contains five positives and ten negatives where the sampler fills ten negative slots in Tier-1, Tier-2, then Tier-3 order. The observed tier distribution in Appendix Section A.2 shows that Tier 3 candidates were available but not selected in these runs, possibly because the relatively broad similarity band and Tier 2 pool of up to 50 candidates allowed the earlier tiers to fill all negative slots, so the aggregate results cannot identify any tier's causal contribution.

3.2 Memory-bank retrieval and refresh

Following CoCoHaNeRe's use of a KNN-searchable memory bank for hard-negative retrieval (Fan et al., 2025), we organize the cached candidate embeddings as a memory bank and use candidate similarity to weight the sampling. A temperature schedule controls how strongly the sampler favors more similar candidates during training. The memory bank first encodes a candidate when it is needed. After each MPR-Hybrid forward pass, the bank replaces the stored embeddings of the selected candidates with embeddings computed by the current model. Other entries retain their previous embeddings until they are selected again.

3.3 MPPC objective and single-positive baseline

3.3.1 MPPC objective

Each MPR-Hybrid training group consists of one problem query, five accepted solutions, and ten sampled rejected solutions. Let $s(q, c)$ denote the cosine similarity between normalized query and code embeddings, and let τ be the loss temperature. MPR-Hybrid uses $\tau = 0.05$. The group loss assigns probability mass to the accepted solution set:

$$\mathcal{L}_{\mathrm{MPPC}}(q) = -\log \frac{\sum_{p \in P(q)} \exp(s(q,p)/\tau)}{\sum_{p \in P(q)} \exp(s(q,p)/\tau) + \sum_{n \in N(q)} \exp(s(q,n)/\tau)}. \tag{1}$$

This is the multi-positive supervised-contrastive objective of Khosla et al. (2020) applied to one CodeContests group: the five accepted solutions are positives and the ten sampled rejected solutions are negatives. The standard form averages the five positive terms by $1/|P(q)|$. Because $|P(q)| = 5$ is fixed here, omitting that factor changes each group loss by the constant $-\log 5$ but leaves its gradients unchanged.

3.3.2 Single-positive in-batch contrastive baseline

We use a single-positive in-batch contrastive-learning baseline that optimizes the InfoNCE loss to test whether MPR-Hybrid's multiple accepted solutions and selected rejected candidates improve retrieval beyond this standard setup. The baseline samples 128 distinct eligible problem queries and pairs each query with one randomly selected accepted solution in the target language. Each paired solution is the only positive for its query, while the other 127 paired solutions in the same logical batch are negatives. Unlike MPR-Hybrid, this baseline does not use rejected submissions, tiered candidate pools, or memory-bank mining. The encoder normalizes every query and code embedding to unit length, so the 128×128 score matrix contains cosine similarities divided by the temperature $\tau = 0.2$. The model minimizes the InfoNCE cross-entropy loss (van den Oord et al., 2018) by selecting the matching diagonal pair for each query.

Together, these steps define two training paths from CodeContests: i) a single-positive baseline using in-batch negatives, and ii) MPR-Hybrid using five accepted solutions with selected rejected submissions. Figure 1 shows how both paths lead to LoRA adapters selected on validation data.

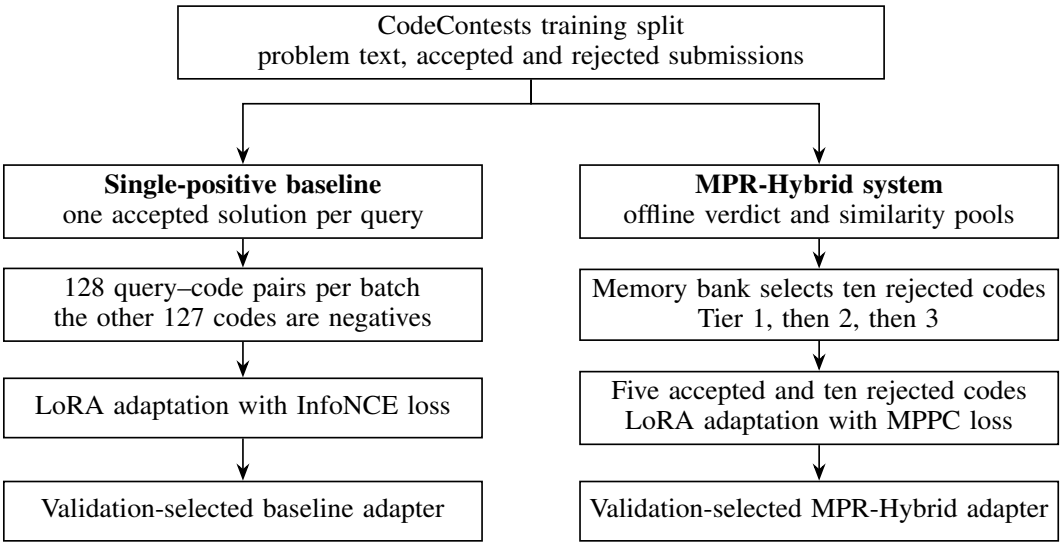

Figure 1: Training paths. Both use LoRA on a frozen encoder. MPR-Hybrid selects tiered negatives with a memory bank and refreshes embeddings of mined candidates.

3.4 Evaluation protocols and scope

We use four evaluations to test retrieval on unseen contest problems, transfer to general code search, and sensitivity to difficult candidates or renamed identifiers. Both adapted systems rank the same candidates in each comparison.

CodeContests official test. Queries describe problems absent from training and validation. Each fixed candidate set contains every accepted solution to the query problem and exactly ten sampled same-language solutions from other problems. This tests whether the encoder retrieves solutions to the correct problem. It does not directly test whether it can distinguish accepted from rejected submissions to that same unseen problem. The pools contain a median of roughly 87 accepted solutions for C++, 70 for Java, and 78.50 for Python against ten negatives. With so many positives, uniformly random rankings already score 0.93, 0.77, and 0.80 MRR@10, respectively. The small test split and ten cross-problem negatives per pool also limit evaluation diversity.

CodeSearchNet zero-shot transfer. We apply the CodeContests-selected adapters to Java and Python code search without further training on CodeSearchNet (Husain et al., 2019). Each query is a natural-language description with one labeled matching code item. This tests whether learning from competitive-programming solutions helps retrieve code in a different setting. Frozen encoders provide a reference for performance before adaptation.

BM25 candidate pools. For each Python query, the encoder ranks one labeled positive and 100 distractors in a fixed pool constructed using keyword matching (BM25). Lexically similar distractors test retrieval when keyword overlap alone may be misleading. We compare the original code with code whose identifiers are masked, keeping candidate identities and relevance labels fixed.

CodeXGLUE AdvTest. This separate Python stress test compares retrieval on aligned original and identifier-obfuscated code (Lu et al., 2021). It tests sensitivity to identifier names while retaining the same examples across the two conditions. We use only examples shared by the clean and obfuscated sources, so scores describe this aligned subset.

CodeContests can contain several relevant solutions per query. The other evaluations provide one labeled relevant code item per query, and the remaining candidates are treated as negatives when computing scores. We additionally remove documentation and comments from CodeSearchNet Python candidates to test how those text fields affect the measured gains. Figure 2 summarizes the evaluations and this documentation check.

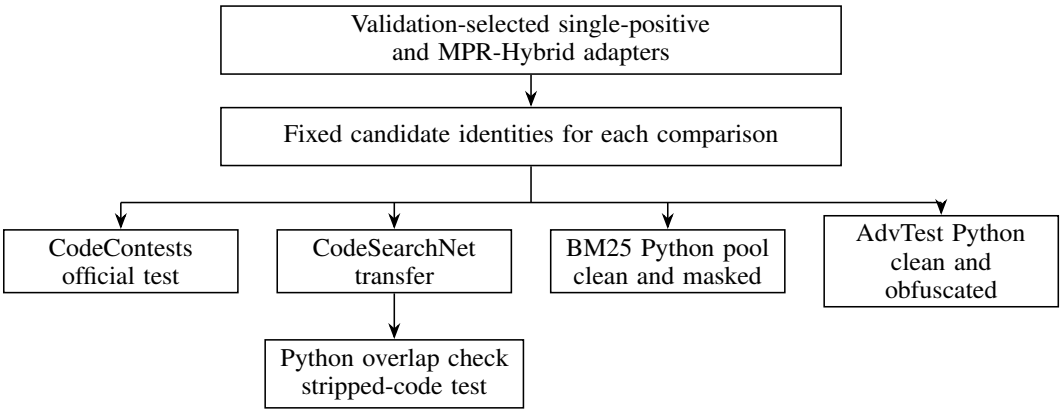

Figure 2: Evaluation paths for the locked adapters, with an additional documentation check on CodeSearchNet Python.

4 EXPERIMENTS

We compare validation-selected MPR-Hybrid adapters with those from a single-positive in-batch baseline trained with the InfoNCE loss. Both use CodeContests training across six backbones, with separate C++, Java, and Python adapters where available. Both systems use the same fixed candidates within each evaluation. Appendix Section A.3 gives the run settings and evaluation-set sizes. We assess retrieval by ranking code for each query: MRR@10 rewards placing the first relevant result high in the top ten, while nDCG@10 scores the positions of all known relevant results. For each evaluation, we subtract the single-positive baseline's score from the MPR-Hybrid score on the same queries. We estimate uncertainty in that difference by resampling the same queries for both systems 10,000 times (paired bootstrap, seed 42) to obtain 95% confidence intervals for both metrics. An MPR-Hybrid win requires both intervals entirely above zero, while a loss requires both entirely below zero. All other results are inconclusive. The appendix places absolute scores and paired intervals with their corresponding experiments. Since each adapter was trained once, these intervals reflect variation across evaluation queries, not training runs.

4.1 CODECONTESTS TEST RESULTS

For the unseen-problem part of RQ2, Table 1 shows no confirmed MPR-Hybrid win. The comparisons meeting our two-metric rule are losses for CodeSage-Small and CodeRankEmbed on C++ and UniXcoder on Java. UniXcoder's C++ and Python MRR@10 intervals include zero, so their lower nDCG@10 scores alone do not establish losses under this rule. Several other Java nDCG@10 scores decline, but their MRR@10 intervals also include zero. CodeBERT's higher C++ score needs caution because both adapters score below the 0.93 MRR@10 reference for random ranking in pools with many accepted solutions but only ten negatives. Appendix Table S5 reports frozen and adapted scores in panels (a) and (b), with paired between-system intervals in panel (c).

Table 1: CodeContests test: MPR-Hybrid minus the single-positive baseline (MRR@10 / nDCG@10). Intervals are in Appendix Table S5. A signed <0.001 denotes a nonzero magnitude below 0.001.

Backbone	C++	Java	Python
CodeBERT	$+0.03/+0.04$	$+0.02/+0.02$	$+0.01/+0.01$
UniXcoder	$-0.02/-0.06$	$-0.07/-0.09$	$-0.02/-0.05$
CodeSage-Small	$-0.04/-0.03$	$-0.02/-0.04$	$-0.006/-0.01$
CodeRankEmbed	$-0.03/-0.03$	$-0.01/-0.03$	$-0.002/-0.01$
Jina v2 Code	$+0.006/+0.001$	$-0.03/-0.02$	$-0.003/-0.005$
EmbeddingGemma	$+0.00/-<0.001$	$-0.004/-0.003$	$+0.005/+0.005$

4.2 ZERO-SHOT TRANSFER TO CODESEARCHNET

The official CodeContests test asks how the adapters perform on new contest problems. RQ1 asks whether training on those problems helps when the task changes to general code search. We evaluate the adapters selected on CodeContests validation data on Java and Python CodeSearchNet without further training. CodeSearchNet has no C++ split, so this transfer test cannot include C++ (Husain et al., 2019). Table 2 compares MPR-Hybrid with the single-positive adapter of the same backbone. Within each language, both adapters rank the same CodeSearchNet candidates. In Python, MPR-Hybrid meets the two-metric criterion for a gain with CodeBERT, UniXcoder, CodeSage-Small, CodeRankEmbed, and Jina v2 Code. The Python EmbeddingGemma comparison is inconclusive because the confidence intervals for both metric differences include zero. In Java, MPR-Hybrid meets the same criterion for a gain with CodeSage-Small, CodeRankEmbed, and Jina v2 Code. It shows confirmed declines with CodeBERT and EmbeddingGemma, while UniXcoder remains inconclusive. CodeRankEmbed has the largest Java gain. For CodeBERT, both adapted systems have near-zero absolute Java scores (Appendix Table S6).

Table 2: Zero-shot CodeSearchNet transfer. Each cell reports MPR-Hybrid minus the single-positive baseline in MRR@10 / nDCG@10.

Backbone	Java	Python
CodeBERT	$-0.002/-0.003$	$+0.05/+0.06$
UniXcoder	$+<0.001/+0.003$	$+0.02/+0.02$
CodeSage-Small	$+0.002/+0.002$	$+0.005/+0.004$
CodeRankEmbed	$+0.03/+0.03$	$+0.006/+0.005$
Jina v2 Code	$+0.01/+0.01$	$+0.002/+0.002$
EmbeddingGemma	$-0.002/-0.002$	$+<0.001/+<0.001$

To check whether either adapter improves on its original encoder, we also compare both adapters with the frozen encoder on the same Java and Python queries and candidate pools. The exploratory paired intervals in Appendix Tables S10 and S11 show no MPR-Hybrid gain over frozen on Java. On Python, MPR-Hybrid exceeds frozen for CodeBERT and UniXcoder. Its differences from frozen are tiny for the four newer backbones, with only CodeSage-Small's negative interval excluding zero. The single-positive baseline scores below frozen on both metrics in ten of the twelve Java and Python comparisons. Thus, several MPR-Hybrid gains over that baseline reflect the baseline's decline rather than an improvement over the pretrained encoder. These intervals measure variation across queries, not across training runs. Appendix Table S6, panels (a) and (b), gives the absolute Java and Python scores, with paired differences in panel (c).

On CodeContests and CodeSearchNet, MPR-Hybrid scores for CodeSage-Small and CodeRankEmbed remain close to those of the frozen encoders. On CodeSearchNet, their advantages over the single-positive baseline largely reflect lower scores for that baseline, with little measured improvement over the pretrained encoders.

4.3 BM25 HARD-CANDIDATE RETRIEVAL AND ADVTEST ROBUSTNESS

For the keyword-matched part of RQ2, we test whether the Python adapters can retrieve the known positive from a pool of 101 code candidates. For each query, keyword matching (BM25) chooses 100 other candidates independently of either system, making shared words alone less useful for ranking (Robertson & Zaragoza, 2009). To test whether retrieval depends on code names, we compare the original pool with a version in which replaceable Python names are changed to neutral placeholders. Queries and candidate identities stay fixed, while comments and docstrings remain. On clean code, MPR-Hybrid shows confirmed gains over the single-positive baseline for five of six backbones. In the identifier-masked version of this same BM25 test, it retains confirmed gains for CodeBERT, CodeSage-Small, CodeRankEmbed, and Jina v2 Code, but UniXcoder changes to a confirmed loss. EmbeddingGemma remains inconclusive in both conditions. CodeRankEmbed's gain increases rather than decreases after masking. Thus the advantage holds for several backbones in this fixed keyword-matched test, but the test does not isolate any negative tier or measure program correctness. Appendix Table S7 gives absolute scores and paired between-system differences. An illustrative query appears in Appendix Figure S1.

For the identifier-obfuscation part of RQ2, we use AdvTest to ask whether retrieval depends on familiar names in Python code. It provides matched clean and identifier-obfuscated candidates for the same queries (Figure 3).

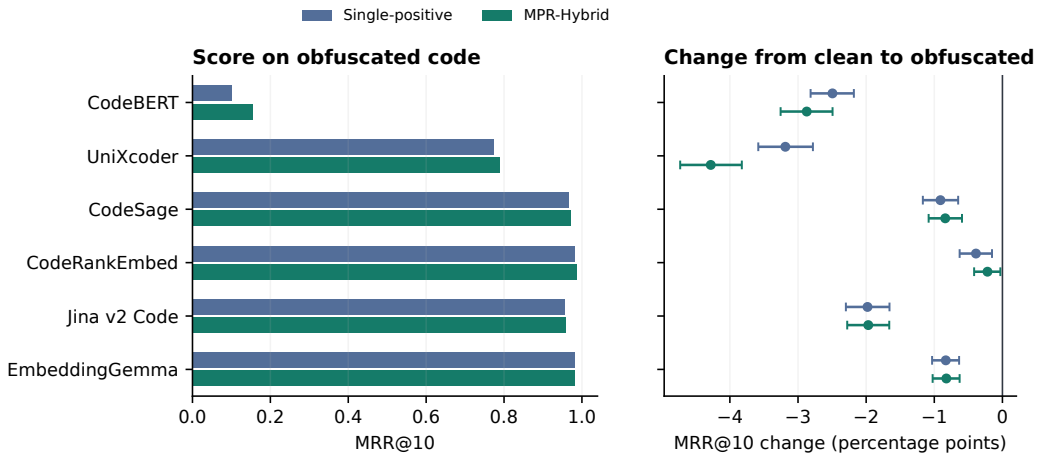

Figure 3: AdvTest Python: MRR@10 on obfuscated code (left) and change from clean code (right). More negative changes mean larger drops. Whiskers are paired 95% query intervals.

AdvTest keeps candidate identities and known positives aligned across the clean and obfuscated versions. We therefore compare both each system's score on obfuscated code and its drop from clean code. MPR-Hybrid scores higher on obfuscated code for five backbones, while EmbeddingGemma is nearly tied. However, its drop from clean to obfuscated code is larger than the single-positive baseline's for CodeBERT and UniXcoder. A higher obfuscated score therefore does not by itself mean greater resistance to identifier changes. Appendix Table S8 gives the absolute clean and obfuscated scores with paired between-system intervals.

4.4 Effect of documentation overlap on Python retrieval

The BM25 and AdvTest checks above leave candidate documentation in place. RQ3 asks whether this documentation contributes to the Python transfer gain. We audited the Python evaluation candidates and found that the known-positive code includes the query text verbatim in its documentation for most or all queries in every Python test with a confirmed MPR-Hybrid gain. We test this dependence by removing docstrings and comments from CodeSearchNet Python candidates and scoring the same locked adapters again. We compared each candidate's parsed code structure (its abstract syntax tree, or AST) before and after removal, ignoring docstrings, to verify that identifiers and the remaining structure stayed unchanged wherever parsing succeeded. The 247 candidates that cannot be parsed are left unchanged. Queries, relevance labels, candidate identities, and row order stay fixed.

Removing documentation changes four of the six Python comparisons (Table 3). CodeBERT and CodeSage-Small lose their confirmed gains, UniXcoder changes from a gain to a loss, and EmbeddingGemma changes from inconclusive to a loss. Only CodeRankEmbed and Jina v2 Code retain confirmed gains. For the four newer backbones, the frozen encoder scores close to both adapters on stripped code. The same pattern holds after excluding queries with any remaining verbatim overlap. These results show that several Python gains are sensitive to documentation removal. They do not establish a general improvement in retrieval without documentation.

Table 3: CodeSearchNet Python MRR@10. Δ is MPR-Hybrid minus the single-positive baseline (paired 95% query intervals).

Backbone	Original Δ	Stripped Δ	Frozen base (stripped)
CodeBERT	$+0.05$ $[+0.05, +0.06]$	$+0.001$ $[-<0.001, +0.001]$	0.008
UniXcoder	$+0.02$ $[+0.02, +0.03]$	-0.05 $[-0.05, -0.04]$	0.49
CodeSage-Small	$+0.005$ $[+0.003, +0.006]$	$+0.001$ $[-0.001, +0.003]$	0.76
CodeRankEmbed	$+0.006$ $[+0.005, +0.007]$	$+0.004$ $[+0.003, +0.005]$	0.85
Jina v2 Code	$+0.002$ $[+0.001, +0.003]$	$+0.003$ $[+0.001, +0.004]$	0.80
EmbeddingGemma	$+<0.001$ $[-<0.001, +<0.001]$	-0.001 $[-0.001, -<0.001]$	0.85

5 Limitations and future work

This audit evaluates complete training systems under the tested protocols. Their differences in loss, negative selection, and LoRA placement prevent attributing gains to any one component. Query-based intervals from a single training seed do not measure variation across training runs. Future matched-budget studies should change one component at a time with the same backbone and negative count, repeat key runs across seeds, and retrain adapters with matched LoRA targets. For negative mining, they should vary the Tier-2 pool size and similarity bounds and ensure Tier 3 is sampled before comparing retrieval. Tier 3 was unused here, so we cannot infer whether cross-language negatives help.

For CodeSage-Small and CodeRankEmbed MPR-Hybrid, validation selected checkpoints during the initial phase in which the learning rate increases (warmup). Appendix Section A.3 records the selected checkpoints and learning rates. These results describe the selected adapters under the tested training schedule. They do not establish how these backbones would perform after more extensive optimization, or whether warmup caused the near-frozen scores.

CodeContests ranks accepted solutions against code from other problems, not rejected submissions to the same unseen problem. BM25 tests keyword-similar candidates and AdvTest changes identifiers, but neither measures program correctness. Future tests should rank accepted and rejected solutions to one unseen problem in larger pools with multiple judged positives and negatives.

Documentation removal shows that several Python transfer comparisons are sensitive to candidate documentation, without isolating the training mechanism responsible. CodeSearchNet Java has only 0.40% verbatim query overlap, but Java candidates exclude Javadoc whereas Python candidates include docstrings, so this is not a controlled language comparison. Future work should vary documentation overlap while keeping queries, candidate identities, and relevance labels fixed.

6 Conclusion

The central finding is that gains over another adapter do not necessarily establish gains over the frozen encoder, and their interpretation depends on the evaluation protocol. Compared with the single-positive in-batch baseline, MPR-Hybrid has no confirmed gain among the available CodeContests official-test pairs. In zero-shot CodeSearchNet Java transfer, it gains on three of six backbones, declines on two, and leaves one comparison inconclusive. None of those three Java gains over the single-positive baseline is a gain over the frozen encoder. Python transfer initially looks more favorable, but removing candidate documentation and comments eliminates or reverses several gains while two remain confirmed over the single-positive baseline. These remaining wins do not by themselves establish improvement over frozen encoders. Together, the transfer comparisons (RQ1), tests on unseen problems and altered candidates (RQ2), and documentation-removal analysis (RQ3) identify conditional benefits and limits of the tested adaptation systems.

More broadly, this study shows why solution-set training should be evaluated against both a single-positive adapter and the original pretrained encoder, using candidate pools that limit simple text matching. That gives future work a clearer way to test whether learning from multiple solutions improves code retrieval.

AI USE STATEMENT

Codex assisted with finding and synthesizing relevant literature, implementing and checking code, setting up experiments, inspecting results, preparing figures and tables, and drafting and editing parts of this manuscript. Gemini Notebook (formerly NotebookLM) was used to organize the papers and research notes we gathered, compare ideas across those sources, and help formulate possible research approaches. Both tools provided feedback on evaluation design and interpretation. The authors reviewed and revised AI-assisted material, checked reported results against saved run records, made the final decisions about the methods, experiments, and claims, and take responsibility for the paper.

REPRODUCIBILITY STATEMENT

To support reproducibility of the reported comparisons, we document how the adapters were trained, which checkpoints were selected, and which code candidates were ranked in each evaluation. Section 3 describes the training data, adaptation systems, and evaluation protocols, while Section 4 explains the retrieval metrics and uncertainty estimates. The appendix supplements these descriptions with run settings, evaluation-set sizes, and detailed results, including the documentation-removal analysis. Saved configuration files and evaluation records link the reported scores to the selected checkpoints and candidate pools.

REFERENCES

Saumya Chaturvedi, Aman Chadha, and Laurent Bindschaedler. LoRACode: LoRA adapters for code embeddings. In *Deep Learning for Code (DL4C) Workshop at ICLR 2025*, 2025.

CodeSage. CodeSage-Small model card. Hugging Face, 2024. URL `https://huggingface.co/codesage/codesage-small`. Accessed September 2026.

Ye Fan, Chuanyi Li, Jidong Ge, LiGuo Huang, and Bin Luo. Effective hard negative mining for contrastive learning-based code search. *ACM Transactions on Software Engineering and Methodology*, 34(3):1–35, 2025.

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

Zuchen Gao, Zizheng Zhan, Xianming Li, Erxin Yu, Haotian Zhang, Bin Chen, Yuqun Zhang, and Jing Li. OASIS: Order-augmented strategy for improved code search. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18451–18467. Association for Computational Linguistics, 2025.

Jing Gong, Yanghui Wu, Linxi Liang, Yanlin Wang, Jiachi Chen, Mingwei Liu, and Zibin Zheng. CoSQA+: Pioneering the multi-choice code search benchmark with test-driven agents. *arXiv preprint arXiv:2406.11589*, 2024.

Google. EmbeddingGemma 300M model card. Hugging Face, 2025. URL `https://huggingface.co/google/embeddinggemma-300m`. Accessed September 2026.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 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)*, pp. 7212–7225. Association for Computational Linguistics, 2022.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. CoSQA: 20,000+ web queries for code search and question answering. In *Proceedings of the*

59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 5690–5700. Association for Computational Linguistics, 2021.

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

Jina AI. Jina Embeddings v2 Base Code model card. Hugging Face, 2024. URL `https://huggingface.co/jinaai/jina-embeddings-v2-base-code`. Accessed September 2026.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 18661–18673, 2020.

Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Hao Zhang, Xinyi Dai, Yasheng Wang, and Ruiming Tang. CoIR: A comprehensive benchmark for code information retrieval models. *arXiv preprint arXiv:2407.02883*, 2025. ACL 2025 (Main Conference).

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021. NeurIPS 2021 Datasets and Benchmarks Track; includes the AdvTest code-search task.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*, 2025.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

John Schulman. LoRA Without Regret. Thinking Machines Lab blog, September 2025. URL `https://thinkingmachines.ai/blog/lora/`.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. CoRNStack: High-quality contrastive data for better code retrieval and reranking. In *International Conference on Learning Representations (ICLR)*, 2025.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Kaicheng Wang, Liyan Huang, Weike Fang, and Weihang Wang. CLARC: C/C++ benchmark for robust code search. In *International Conference on Learning Representations (ICLR)*, 2026.

Siqiao Xue, Zihan Liao, Jin Qin, Ziyin Zhang, Yixiang Mu, Fan Zhou, and Hang Yu. Beyond retrieval: A multitask benchmark and model for code search. *arXiv preprint arXiv:2605.04615*, 2026.

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K. Reddy. XLCoST: A benchmark dataset for cross-lingual code intelligence. *arXiv preprint arXiv:2206.08474*, 2022.