Hierarchy-Aware Code Retrieval via Hyperbolic Entailment Learning

Anonymous ACL submission

Abstract

Code retrieval serves as the cornerstone of modern software engineering. The prevailing paradigm typically represents source code in Euclidean space, relying on the cosine similarity of normalized embeddings for retrieval. Such approaches capture only flat semantics and struggle to account for the inherent hierarchical structure of source code. Hyperbolic space, characterized by its exponential volume growth, is naturally suitable for representing hierarchical relationships. Motivated by this, we propose **HACR**, a **H**ierarchy-**A**ware **C**ode **R**etrieval framework that leverages the geometric properties of hyperbolic manifolds to help existing transformer-based models capture the hierarchical semantics of source code. **HACR** is built on the premise that source code is not merely a sequence of tokens but a composition of code snippets. Specifically, we map both source code and code snippets into hyperbolic space, utilizing negative geodesic distance as the similarity measure to ensure that representations reflect hierarchical depth. Furthermore, we introduce hyperbolic entailment cones as geometric constraints to explicitly model the asymmetric entailment relationships between code snippets and their corresponding source code. Empirical evaluations on monolingual and cross-lingual code-to-code retrieval benchmarks demonstrate that **HACR** outperforms Euclidean baselines across various metrics while maintaining comparable inference efficiency. Our code is available at `https://anonymous.4open.science/r/HEAL-FB14`.

1 Introduction

Code retrieval enables developers to identify functionally equivalent source code within massive repositories, supporting critical downstream applications such as code clone and plagiarism detection (Ebrahim and Joy, 2024; Hrkút et al., 2023; Ebrahim and Joy, 2023), code completion and retrieval-augmented code generation (Yang et al., 2025; Du et al., 2024). In recent years, transformer-based models, such as CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021), have driven a paradigm shift from lexical matching toward semantic alignment. By pre-training on large-scale code corpora, these models learn rich semantic representations of source code and achieve superior retrieval performance. Despite their success, these models treat source code as a flat sequence of tokens, thereby neglecting its inherent hierarchical structure. To mitigate this limitation, structure-aware approaches, such as GraphCodeBERT (Guo et al., 2021), inject data-flow information during pre-training, while GNN-based methods directly encode code graphs, such as Abstract Syntax Trees (ASTs) (Alon et al., 2019a,b) and Data Flow Graphs (DFGs) (Guo et al., 2021), to capture program structure dependencies. However, these works often require language-specific graph construction, which is cumbersome compared with modern transformer architecture. How to endow transformer-based models with structural awareness without changing their underlying backbone

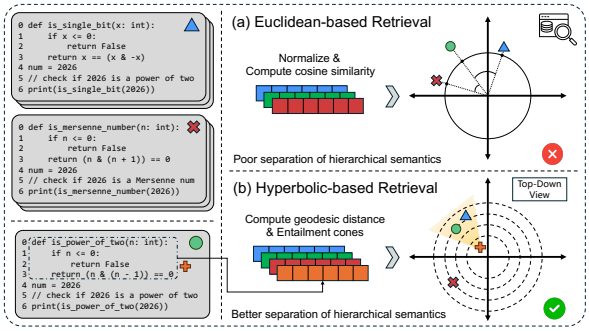


Figure 1: Euclidean vs. hyperbolic code retrieval. Euclidean-based retrieval mainly relies on surface-level semantic similarity, resulting in misleading matches (red). In contrast, our hyperbolic-based retrieval captures the underlying hierarchical relationships, enabling accurate retrieval of functional equivalents despite syntactic variations (blue).

or introducing expensive graph construction? We revisit it from a geometric perspective.

Existing works predominantly rely on the assumption that source code can be effectively represented in high-dimensional Euclidean space (Radovanovic et al., 2010; Tomasev et al., 2014), where semantic similarity is measured by the cosine similarity between normalized embeddings. This paradigm inevitably flattens the hierarchical depth of representations, failing to preserve the hierarchical organization of source code in the embedding space. To address this limitation, we turn to hyperbolic space. Unlike Euclidean space, where volume grows polynomially with radius, hyperbolic space expands exponentially (Nickel and Kiela, 2017; Krioukov et al., 2010). This property allows it to accommodate the exponential growth of nodes in tree-like structures and embed them with minimal distortion (Sarkar, 2011), which is mathematically impossible in Euclidean space. As illustrated in Figure 1, our hyperbolic retrieval framework incorporates hierarchical depth into representations, enabling the effective preservation of hierarchical relationships. In contrast, embeddings in Euclidean-based retrieval are normalized, limiting their capacity to capture hierarchical semantics.

Although hyperbolic space provides an effective inductive bias for embedding hierarchical structures, explicitly modeling the hierarchical relationships within source code remains non-trivial. This presents two key challenges. First, the intrinsic hierarchy of source code is often flattened in standard datasets, making it difficult to recover structural dependencies from linearized sequences. Second, without explicit entailment-oriented training objectives, merely mapping source code into hyperbolic space does not necessarily induce a meaningful hierarchical organization.

To address these challenges, we propose **HACR**, a plug-and-play **H**ierarchy-**A**ware **C**ode **R**etrieval framework. **HACR** consists of three stages. First, to compensate for the limited structural information in existing benchmarks, we augment the training samples by extracting code snippets from syntax subtrees of source code. Second, we embed both source code and extracted snippets into hyperbolic space, where negative geodesic distance serves as the similarity metric. Finally, we employ hyperbolic entailment cones to explicitly model asymmetric entailment relationships between snippets and their corresponding source code.

Our contributions are summarized as follows:

- **New Perspective.** We reveal the limitations of existing transformer-based models in capturing the hierarchical structure of source code and revisit code-to-code retrieval from a hyperbolic geometric perspective.
- **Novel Framework.** We propose **HACR**, a hierarchy-aware code retrieval framework that leverages geodesic distance and hyperbolic entailment cones to capture hierarchical semantics beyond Euclidean representations.
- **Superior Performance.** Extensive experiments on both monolingual and cross-lingual code-to-code retrieval benchmarks demonstrate that **HACR** consistently outperforms Euclidean baselines while maintaining comparable inference efficiency.

2 Preliminaries

2.1 Hyperbolic Space

Hyperbolic space H_c^n is a Riemannian manifold with constant negative sectional curvature $-c$ ($c > 0$). While several isometric models of hyperbolic space exist—such as the Poincaré ball and Klein models—we adopt the Lorentz model due to its numerical stability and computational efficiency.

2.2 The Lorentz Model

The Lorentz model represents hyperbolic space as a submanifold embedded in an $(n+1)$-dimensional Minkowski space. We designate the zeroth dimension as the temporal component.

Lorentzian Inner Product Let R^{n+1} be an $(n+1)$-dimensional vector space. The Lorentzian inner product (also known as the Minkowski inner product) of two vectors $\mathbf{x}, \mathbf{y} \in R^{n+1}$ is defined as:

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} = -x_0 y_0 + \sum_{i=1}^{n} x_i y_i. \tag{1}$$

Lorentz Model The Lorentz model of an n-dimensional hyperbolic space with curvature $-c$, denoted as $\mathcal{L}_c^n$, is defined as the upper sheet of a hyperboloid:

$$\mathcal{L}_c^n = \left\{ \mathbf{x} \in R^{n+1} : \langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -1/c,\ x_0 > 0 \right\}. \tag{2}$$

Tangent Space For any point $\mathbf{x} \in \mathcal{L}_c^n$, the tangent space $T_{\mathbf{x}}\mathcal{L}_c^n$ consists of vectors in R^{n+1} orthogonal to $\mathbf{x}$ under the Lorentzian inner product:

$$T_{\mathbf{x}}\mathcal{L}_c^n = \left\{ \mathbf{v} \in R^{n+1} : \langle \mathbf{x}, \mathbf{v} \rangle_{\mathcal{L}} = 0 \right\}. \tag{3}$$

Geodesic Distance The geodesic distance between two points $\mathbf{x}, \mathbf{y} \in \mathcal{L}_c^n$ is defined as the length of shortest path connecting them on the manifold:

$$d_{\mathcal{L}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{c}} \operatorname{arccosh}(-c\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}). \quad (4)$$

Exponential and Logarithmic Maps To facilitate mapping between tangent space and the manifold, we utilize exponential and logarithmic maps.

The **exponential map** $\exp_{\mathbf{x}} : T_{\mathbf{x}}\mathcal{L}_c^n \to \mathcal{L}_c^n$ projects a tangent vector $\mathbf{v} \neq \mathbf{0}$ onto the manifold:

$$\exp_{\mathbf{x}}(\mathbf{v}) = \cosh(\sqrt{c}\|\mathbf{v}\|_{\mathcal{L}})\mathbf{x} + \frac{\sinh(\sqrt{c}\|\mathbf{v}\|_{\mathcal{L}})}{\sqrt{c}\|\mathbf{v}\|_{\mathcal{L}}}\mathbf{v}. \quad (5)$$

The **logarithmic map** $\log_{\mathbf{x}} : \mathcal{L}_c^n \to T_{\mathbf{x}}\mathcal{L}_c^n$ performs the inverse operation, mapping a point $\mathbf{y}$ on the manifold back to the tangent space at $\mathbf{x}$:

$$\log_{\mathbf{x}}(\mathbf{y}) = \frac{\operatorname{arccosh}(-c\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}})}{\sqrt{c^2\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}^2 - 1}}(\mathbf{y} + c\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\mathbf{x}). \quad (6)$$

In this work, we apply the exponential map at the origin $\mathbf{o} = (\sqrt{1/c}, \mathbf{0})^\top$. This simplifies the projection of initial Euclidean embeddings from $T_{\mathbf{o}}\mathcal{L}_c^n$ directly into the hyperbolic space $\mathcal{L}_c^n$.

3 Methodology

Our primary objective is to characterize the intrinsic hierarchy of source code and impose the corresponding hierarchical constraints within the hyperbolic manifold. The overall architecture of HACR is illustrated in Figure 2. In an Abstract Syntax Tree (AST), the root node represents the complete program, whereas descendant nodes correspond to code snippets. Based on our empirical observations, we hypothesize that code snippets tend to exhibit greater semantic generality than the complete program. This may appear counter-intuitive, since the root node structurally subsumes its descendants. However, a complete program typically encodes a specific semantic context, while its constituent snippets often instantiate more general syntactic units. For example, a function that implements a generic operation such as sorting or string matching can be reused across many different programs, whereas an entire program is usually tailored to a particular task and therefore corresponds only to a limited set of constituent functions. (Further justification is provided in Appendix B). This observation motivates our modeling of hierarchical entailment relationships between code snippets and the complete program.

3.1 Data Augmentation

To capture hierarchical semantics of source code, training samples must include not only complete programs but also code snippets. As illustrated in Figure 2(a), we utilize Tree-sitter[1] as a static parsing tool to extract code snippets from the AST. In particular, the extracted snippets are utilized only during the training phase and are not directly involved in the semantic alignment, thereby ensuring both inference efficiency and experimental fairness.

Let the original dataset be $D = \{s_i\}_{i=1}^N$, where s_i denotes the i-th training sample. The dataset D consists of L distinct labels, with each label represented by R implementations. For each $s_i \in D$, there exists a set of $(R-1)$ positive samples $S_i^+ \subset D$ and a set of $(L-1) \times R$ negative samples $S_i^- \subset D$. We parse each s_i into an AST and extract its internal code snippets $P_i = \{p_{i1}, p_{i2}, \ldots, p_{ik}\}$ (Implementation details are provided in Appendix D). To maintain training efficiency, we select a single snippet $p_i \in P_i$ for each source code s_i, forming an augmented dataset $D_{aug} = \{(s_i, p_i)\}_{i=1}^N$. During training, we iterate through D_{aug} and randomly sample one positive instance $s_i^+ \in S_i^+$ and one negative instance $s_i^- \in S_i^-$. Consequently, the training instance for the i-th sample is represented as a quadruple:

$$T_i = (s_i, p_i, s_i^+, s_i^-). \quad (7)$$

3.2 Hyperbolic Retrieval Framework

We retain the transformer backbone (Vaswani et al., 2017) of pre-trained models, but replace the Euclidean representation space and the optimization objective with their hyperbolic counterparts.

3.2.1 Encoding & Mapping

As illustrated in Figure 2(b), let f_θ denote a pre-trained model. For any input sequence x in the training quadruple $T_i = (s_i, p_i, s_i^+, s_i^-)$, we first obtain its Euclidean representation by applying average pooling to the last hidden states:

$$\mathbf{v}_x = \operatorname{AvgPool}\big(f_\theta(x)\big). \quad (8)$$

To project this Euclidean representation onto the n-dimensional Lorentz manifold $\mathcal{L}_c^n$ with constant negative curvature $-c$ $(c > 0)$, we first lift $\mathbf{v}_x$ to the tangent space $T_{\mathbf{o}}\mathcal{L}_c^n$ at the origin $\mathbf{o} = (\sqrt{1/c}, \mathbf{0})^\top$ by concatenating a zero time component:

$$\tilde{\mathbf{v}}_x = (0, \mathbf{v}_x)^\top \in T_{\mathbf{o}}\mathcal{L}_c^n. \quad (9)$$

[1] https://tree-sitter.github.io/tree-sitter.

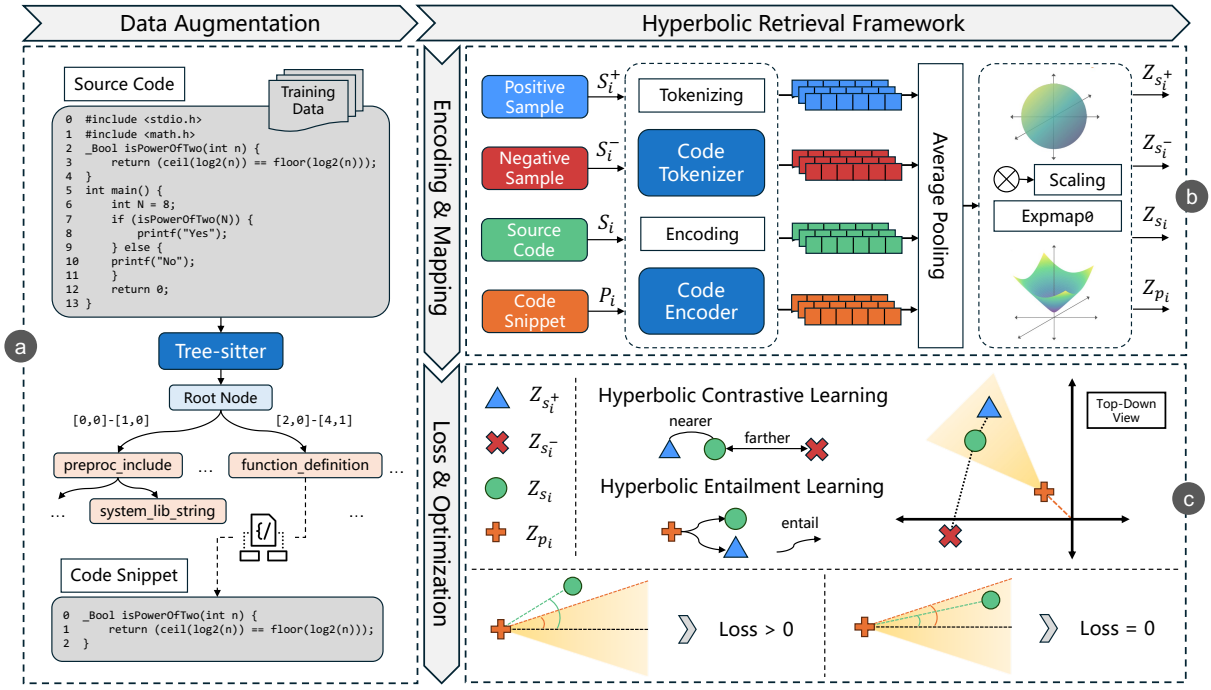


Figure 2: The overall architecture of our proposed framework.

Subsequently, the tangent vector is mapped onto the manifold through the exponential map $\exp_{\mathbf{o}}^{c}(\cdot)$. To improve numerical stability and allow the model to adapt the radial scale during training, we introduce a learnable scalar α and compute the hyperbolic representation as

$$\mathbf{z}_x = \exp_{\mathbf{o}}^{c}(\alpha \tilde{\mathbf{v}}_x). \tag{10}$$

By applying this transformation to the elements of our training quadruple T_i, we obtain a set of hyperbolic representations $\{\mathbf{z}_{s_i}, \mathbf{z}_{p_i}, \mathbf{z}_{s_i^+}, \mathbf{z}_{s_i^-}\} \subset \mathcal{L}_c^n$. These representations support a dual objective, where the geodesic distance and the hyperbolic entailment cones are leveraged to jointly enforce semantic alignment and hierarchical constraints.

3.2.2 Loss and Optimization

Hyperbolic Contrastive Learning In hyperbolic space, we use the negative geodesic distance as the similarity metric. For a mini-batch of size B, consider the i-th source code sample s_i with label y_i. To mitigate the impact of false negatives and improve discriminability, we define a set of negative samples $\mathcal{N}_i$. It includes both the explicitly sampled negative instance s_i^- from the quadruple T_i and the in-batch negatives derived from other samples s_j whose labels differ from y_i:

$$\mathcal{N}_i = \{\mathbf{z}_{s_i^-}\} \cup \{\mathbf{z}_{s_j} \mid j \in \{1, \ldots, B\},\ y_j \neq y_i\}. \tag{11}$$

The hyperbolic contrastive loss for the i-th training instance is formulated as:

$$\ell_{\text{con}}^{(i)} = -\log \frac{A_i}{A_i + \sum_{\mathbf{z}' \in \mathcal{N}_i} \exp(-d_{\mathcal{L}}(\mathbf{z}_{s_i}, \mathbf{z}')/\tau)},$$
$$A_i = \exp\left(-d_{\mathcal{L}}(\mathbf{z}_{s_i}, \mathbf{z}_{s_i^+})/\tau\right). \tag{12}$$

where τ is the temperature parameter.

Hyperbolic Entailment Learning To explicitly enforce the asymmetric entailment relationships introduced in Section 3.1, we utilize the concept of hyperbolic entailment cones (Ganea et al., 2018; Le et al., 2019). As shown in Figure 2(c), a point $\mathbf{x}$ entails another point $\mathbf{y}$ if $\mathbf{y}$ lies inside the cone rooted at $\mathbf{x}$, enabling partial orders to be represented within a continuous manifold.

For a point $\mathbf{x} \in \mathcal{L}_c^n$, the entailment cone is characterized by its half-aperture angle $\psi(\mathbf{x})$, which controls the width of the cone and is inversely related to the norm of the embedding's spatial components. Since points farther from the origin correspond to more specific concepts, their cones should be narrower. The exterior angle $\phi(\mathbf{x}, \mathbf{y})$ measures the angular deviation of $\mathbf{y}$ from the cone rooted at $\mathbf{x}$. For $\mathbf{x}, \mathbf{y} \in \mathcal{L}_c^n$, we compute

$$\psi(\mathbf{x}) = \arcsin\left(\frac{2K_0}{\sqrt{c}\,\|\mathbf{x}_{1:n}\|}\right), \tag{13}$$

$$\phi(\mathbf{x}, \mathbf{y}) = \arccos\left(\frac{y_0 + x_0 c\langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}}{\|\mathbf{x}_{1:n}\|\sqrt{\left(c\langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}\right)^2 - 1}}\right), \tag{14}$$

where K_0 is a boundary constant that prevents the cone from collapsing near the origin, and $\mathbf{x}_{1:n}$ denotes the spatial components of $\mathbf{x}$. The entailment condition is therefore $\phi(\mathbf{x}, \mathbf{y}) < \psi(\mathbf{x})$.

We use this geometry to constrain code snippet embeddings to entail source code embeddings, thereby preserving the intrinsic hierarchy within the hyperbolic manifold. Specifically, the snippet embedding $\mathbf{z}_{p_i}$ is required to entail both the source code embedding $\mathbf{z}_{s_i}$ and its positive counterpart $\mathbf{z}_{s_i^+}$. We assign equal importance to these two constraints. To prevent over-regularization, we employ a margin-based penalty that yields zero loss if the target embedding already lies in the valid entailment cone. The resulting hyperbolic entailment loss for the i-th training instance is formulated as:

$$\begin{aligned}\mathcal{L}_{\text{ent}}^{(i)} = \frac{1}{2}\Big[&\big(\phi(\mathbf{z}_{p_i}, \mathbf{z}_{s_i}) - \psi(\mathbf{z}_{p_i})\big)_+ \\ &+ \big(\phi(\mathbf{z}_{p_i}, \mathbf{z}_{s_i^+}) - \psi(\mathbf{z}_{p_i})\big)_+\Big],\end{aligned} \tag{15}$$

where $(a)_+ = \max(0, a)$.

Overall Optimization Objective The total batch loss is the weighted sum of the contrastive and entailment losses:

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{B}\left(\ell_{\text{con}}^{(i)} + \lambda \mathcal{L}_{\text{ent}}^{(i)}\right), \tag{16}$$

where λ controls the trade-off between semantic alignment and hierarchical constraints. By jointly optimizing these objectives, HACR learns representations that exhibit both semantic discriminability and hierarchical fidelity.

4 Experiments

4.1 Experimental Settings

4.1.1 Benchmarks

To evaluate the effectiveness of HACR, we conduct experiments on two widely recognized code-to-code retrieval benchmarks.

- **POJ-104.** Derived from the Peking University Online Judge, POJ-104 covers a broad range of algorithmic problems. It contains 52,000 C/C++ programs from 104 functional classes, with 500 functionally equivalent implementations in each class (Lu et al., 2021).
- **XLCoST.** XLCoST is a cross-lingual benchmark for code intelligence that spans seven programming languages: C++, Java, Python, C#, JavaScript, PHP, and C. It provides aligned snippets and programs across languages, so each snippet or program has corresponding implementations in the other six languages (Zhu et al., 2022).

4.1.2 Metrics

We evaluate retrieval performance with Mean Reciprocal Rank (MRR) and MAP@R. Given Q queries, MRR is defined as

$$\text{MRR} = \frac{1}{|Q|}\sum_{q \in Q}\frac{1}{\text{rank}_q}, \tag{17}$$

where rank_q is the rank of the first relevant result for query q. MAP@R averages precision at the ranks where relevant items appear among the top-R retrieved results:

$$\text{MAP@}R = \frac{1}{|Q|}\sum_{q \in Q}\frac{1}{R_q}\sum_{k=1}^{R} P_q(k)\, I_q(k), \tag{18}$$

where $P_q(k)$ denotes precision at rank k, $I_q(k)$ indicates whether the item ranked at k is relevant, and R_q is the number of relevant items considered for query q. The maximum value of R is 499 for POJ-104 and is 6 for XLCoST.

4.1.3 Baselines and Backbones

We compare HACR with a Euclidean baseline that maps source code to a unit hypersphere and ranks candidates by cosine similarity between query and target embeddings. Since HACR changes only the representation space and the training objective, we use the same five pre-trained backbones for both the baseline and HACR: RoBERTa (Liu et al., 2019), CodeBERT (Feng et al., 2020), PLBART (Ahmad et al., 2021), CodeT5 (Wang et al., 2021), and UniXcoder (Guo et al., 2022). Since structure-aware approaches rely on language-specific graph construction, making them difficult to apply to cross-lingual code retrieval; therefore, we have included additional experimental results using GraphCodeBERT for monolingual code retrieval.

4.2 Experimental Results

Inference In the Lorentz model, minimizing the geodesic distance is equivalent to maximizing the Lorentzian inner product (Proof is provided in Appendix C). Therefore, HACR ranks candidates by the Lorentzian inner product, while the Euclidean baseline ranks candidates by cosine similarity. Although HACR introduces additional code snippets during training, it utilizes only source code at inference time and therefore does not compromise efficiency.

Monolingual Retrieval Performance Table 1 reports monolingual retrieval results on POJ-104 under $R \in \{99, 199, 299, 499\}$. Across all backbones, HACR consistently improves over the Euclidean baseline, especially on MAP@499, which evaluates retrieval over the full set of relevant programs. This indicates that hyperbolic representations better capture the hierarchical structure of source code and improve discrimination among functionally equivalent programs with syntactic variations. The improvement is consistent across both earlier models such as RoBERTa and stronger backbones such as UniXcoder, suggesting that Euclidean space is a limiting factor for code retrieval.

Cross-lingual Retrieval Performance Table 2 reports cross-lingual retrieval results on the program-level XL_Code_Search task from XLCoST. HACR improves over the Euclidean baseline for most backbone-language combinations, showing that the learned hyperbolic representations can encode hierarchical signals that transfer across various programming languages.

4.3 Ablation Study

We conduct two ablation studies to examine the contribution of key design choices in HACR.

Contribution of loss terms We evaluate monolingual retrieval across three representative backbones: PLBART, CodeT5, and UniXcoder. $\mathrm{Cos}_{\mathrm{euc}}$ is the Euclidean baseline, while $\mathrm{Dist}_{\mathrm{hyp}}$ uses only hyperbolic contrastive learning based on negative geodesic distance, corresponding to $\lambda = 0$. HACR denotes the complete framework with both hyperbolic contrastive learning and hyperbolic entailment learning. As shown in Table 3, removing the entailment loss consistently reduces performance, confirming that explicit hierarchical modeling helps the model capture program structure. Meanwhile, $\mathrm{Dist}_{\mathrm{hyp}}$ still substantially outperforms $\mathrm{Cos}_{\mathrm{euc}}$, indicating that hyperbolic space itself provides a more effective inductive bias than Euclidean space for code retrieval.

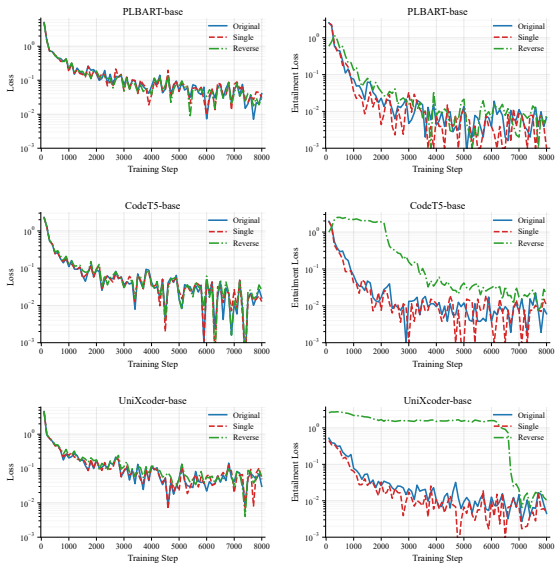


Figure 3: Training dynamics under different entailment variants. Rows correspond to PLBART, CodeT5, and UniXcoder, respectively. The left column shows the total training loss, and the right column shows the entailment loss.

Contribution of entailment variants Figure 3 compares the training dynamics of three entailment variants on POJ-104 across PLBART, CodeT5, and UniXcoder. The original setting, denoted as HACR in Table 4, applies cross-sample entailment, where the code snippet entails both the source code and its positive counterpart. The single variant keeps the same entailment direction but only constrains each snippet to entail its corresponding source code, while the reverse variant flips the entailment direction by forcing full programs to entail snippets. Across all three backbones, the original setting exhibits more stable optimization, especially in the entailment-loss curves. The reverse variant tends to produce larger or more oscillatory entailment losses, indicating that reversing the hierarchy introduces an inconsistent training signal. These trends support our design choice: snippets encode reusable syntactic units that should entail the full programs containing them, and cross-sample entailment further regularizes semantically equivalent programs. Table 4 further shows that HACR consistently outperforms the single and reverse variants.

Backbone	Params	Method	MAP@499 (%)	MAP@299 (%)	MAP@199 (%)	MAP@99 (%)
RoBERTa	125M	Cos_{euc}	**85.63**	95.98	97.23	98.07
RoBERTa	125M	HACR	85.04	**96.22**	**97.6**	**98.48**
CodeBERT	125M	Cos_{euc}	84.42	96.11	97.58	98.39
CodeBERT	125M	HACR	**86.90**	**96.77**	**97.93**	**98.58**
GraphCodeBERT	125M	Cos_{euc}	85.56	96.43	97.65	98.39
GraphCodeBERT	125M	HACR	**88.12**	**97.32**	**98.19**	**98.73**
PLBART	140M	Cos_{euc}	86.69	96.76	97.73	98.4
PLBART	140M	HACR	**89.41**	**97.88**	**98.46**	**98.88**
CodeT5	220M	Cos_{euc}	88.27	96.78	97.88	98.63
CodeT5	220M	HACR	**91.74**	**98.44**	**98.9**	**99.23**
UniXcoder	125M	Cos_{euc}	88.96	97.43	98.01	98.64
UniXcoder	125M	HACR	**92.16**	**98.33**	**98.76**	**99.09**

Table 1: Monolingual retrieval on POJ-104 (MAP@R). Best results are bolded.

Backbone	Params	Method	C++	Java	Python	C#	JavaScript	PHP	C
RoBERTa	125M	Cos_{euc}	48.28±0.07	47.66±0.08	46.11±0.09	46.40±0.07	47.60±0.06	43.76±0.10	40.15±0.11
RoBERTa	125M	HACR	**48.55±0.11**	**48.98±0.12**	**47.23±0.13**	**48.57±0.11**	**48.30±0.09**	**44.56±0.14**	**40.68±0.15**
CodeBERT	125M	Cos_{euc}	48.71±0.06	48.33±0.07	47.24±0.08	47.96±0.06	47.66±0.07	44.02±0.09	40.43±0.10
CodeBERT	125M	HACR	**48.76±0.09**	**49.08±0.10**	**47.48±0.11**	**48.79±0.09**	**48.43±0.10**	**44.05±0.12**	**41.45±0.14**
PLBART	140M	Cos_{euc}	48.53±0.08	48.69±0.07	**48.08±0.09**	48.45±0.08	48.25±0.06	43.90±0.10	39.48±0.12
PLBART	140M	HACR	**48.88±0.12**	**49.11±0.11**	48.05±0.07	**48.94±0.12**	**48.58±0.10**	**44.06±0.13**	**41.52±0.16**
CodeT5	220M	Cos_{euc}	48.73±0.07	48.86±0.06	**48.20±0.05**	48.42±0.08	48.30±0.06	43.77±0.09	40.92±0.10
CodeT5	220M	HACR	**48.87±0.11**	**49.13±0.09**	48.15±0.08	**48.82±0.12**	**48.65±0.09**	**44.32±0.13**	**41.83±0.14**
UniXcoder	125M	Cos_{euc}	48.76±0.06	49.05±0.09	**48.43±0.08**	48.58±0.07	48.49±0.05	44.38±0.08	41.07±0.09
UniXcoder	125M	HACR	**49.12±0.10**	**49.07±0.06**	48.30±0.07	**48.92±0.11**	**48.70±0.09**	**44.69±0.12**	**41.49±0.13**

Table 2: Cross-lingual retrieval performance on XLCoST (MRR, mean ± std over 3 seeds). Best results are bolded.

Backbone	Method	MAP@499 (%)	MAP@299 (%)
PLBART	Cos_{euc}	86.69	96.76
PLBART	Dist_{hyp}	89.29	97.47
PLBART	HACR	**89.41**	**97.88**
CodeT5	Cos_{euc}	88.27	96.78
CodeT5	Dist_{hyp}	90.94	97.99
CodeT5	HACR	**91.74**	**98.44**
UniXcoder	Cos_{euc}	88.96	97.43
UniXcoder	Dist_{hyp}	90.78	98.08
UniXcoder	HACR	**92.16**	**98.33**

Table 3: Ablation study of loss terms.

Backbone	Method	MAP@499 (%)	MAP@299 (%)
PLBART	Reverse	88.42	96.02
PLBART	Single	89.27	96.39
PLBART	HACR	**89.41**	**97.88**
CodeT5	Reverse	91.25	97.19
CodeT5	Single	91.49	97.30
CodeT5	HACR	**91.74**	**98.44**
UniXcoder	Reverse	90.67	97.03
UniXcoder	Single	91.58	97.49
UniXcoder	HACR	**92.16**	**98.33**

Table 4: Ablation study of entailment variants.

4.4 Hyperparameter Analysis

Figure 4 shows the sensitivity of HACR to the main hyperparameters on POJ-104 with UniXcoder as the backbone.

Impact of λ The coefficient λ controls the strength of the entailment loss. Performance peaks at $\lambda = 0.01$: smaller values weaken hierarchical constraints, whereas larger values interfere with semantic alignment.

Impact of α The scaling parameter α controls the norm of representations before exponential mapping. Performance is best at $\alpha = 1.0$. Smaller values keep embeddings too close to the Lorentz origin, while larger values may cause numerical instability.

Impact of c The curvature parameter c determines the curvature $-c$ of the Lorentz manifold. We observe the best performance at $c = 1.0$. A moderate curvature value balances semantic alignment and hierarchical modeling.

4.5 Qualitative Analysis

We analyze embedding norms for snippets and full programs in the POJ-104 test set with UniXcoder as the backbone. We extract the spatial components of hyperbolic embeddings and visualize them in two dimensions. As shown in Figure 5, both Cos_{euc} and Dist_{hyp} produce relatively uniform distributions. In contrast, Figure 6 shows a clear radial

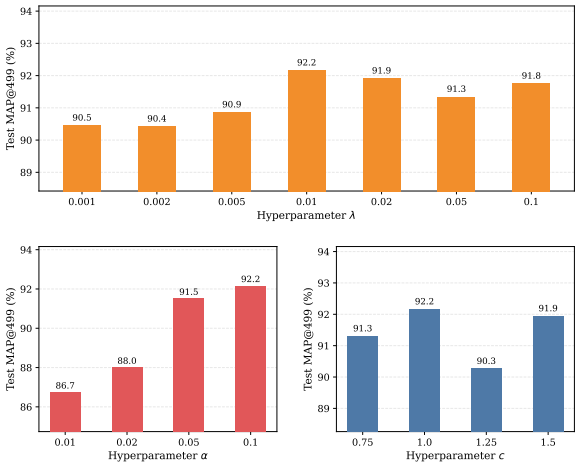
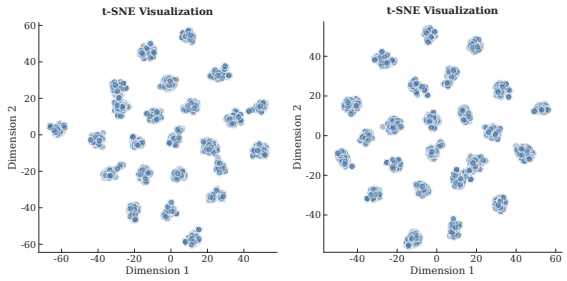


Figure 4: Analysis of the hyperparameters λ, α, and c.



(a) Cos_{euc} (b) Dist_{hyp}

Figure 5: t-SNE visualizations of the Euclidean cosine baseline and the hyperbolic distance variant.

structure: snippets are closer to the Lorentz origin, while full programs are distributed farther away. This pattern empirically supports HACR's ability to model asymmetric entailment relationships between fine-grained snippets and full programs.

5 Related Works

Transformer-based code retrieval Pre-trained transformer models have become the prevailing paradigm for code retrieval. Models such as CodeBERT (Feng et al., 2020), PLBART (Ahmad et al., 2021), CodeT5 (Wang et al., 2021), and UniXcoder (Guo et al., 2022) learn contextual code representations and scale well to large repositories. However, they usually encode source code as token sequences and optimize retrieval in Euclidean space, making it difficult to preserve hierarchical program structure. This limitation is especially evident when structural awareness is required. Even structure-aware variants such as GraphCodeBERT (Guo et al., 2021) mainly inject auxiliary structural signals during pre-training. In contrast, HACR retains the transformer backbone

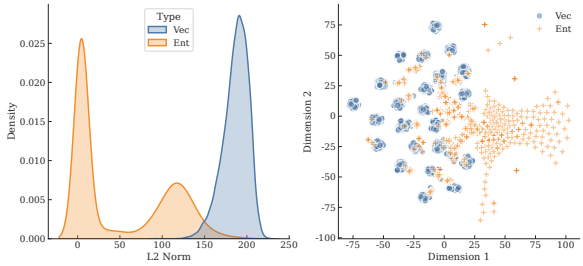


Figure 6: Embedding norm statistics (left) and t-SNE visualization of the HACR framework (right).

but uses hyperbolic geometry to encode hierarchy-awareness into the representation space.

GNN-based code retrieval Another line of work represents programs as code graphs and applies GNNs to model structural dependencies. Such graphs can capture syntactic hierarchies, data flow, and control-flow dependencies (Allamanis et al., 2018); FA-AST and GraphSearchNet use GNNs for clone detection and code search (Wang et al., 2020; Liu et al., 2021). However, these methods often require language-specific graph construction, graph batching, and iterative message passing, which complicates deployment in industrial scenarios. Their dependence on explicit graph schemas also makes it harder to transfer a unified retrieval pipeline across programming languages with heterogeneous syntax and analysis tools. HACR avoids such graph-centric pipelines: it uses snippets only during training and injects hierarchical inductive bias through hyperbolic entailment, preserving transformer scalability while enabling hierarchy-aware retrieval.

6 Conclusion

In this paper, we propose HACR, a hierarchy-aware code retrieval framework that extracts code snippets from ASTs and maps them into hyperbolic space. Using geodesic distance and hyperbolic entailment cones, HACR optimizes a joint objective that combines hyperbolic contrastive and entailment losses, enabling semantic alignment while enforcing hierarchical constraints. Experiments on both monolingual and cross-lingual code retrieval benchmarks show that HACR consistently outperforms Euclidean baselines. Ablation studies and qualitative analysis further demonstrate that HACR induces a meaningful hierarchical organization in hyperbolic space, improving the interpretability of learned representations.

Limitations

In this work, we focus on code-to-code retrieval and have not yet evaluated HACR on natural-language-to-code retrieval tasks. Since nl2code retrieval requires aligning natural language queries with source code, its hierarchy may differ from the code-code hierarchy modeled in this paper. Extending HACR to nl2code retrieval therefore requires further investigation. In addition, HACR introduces several hyperparameters, including the entailment weight λ, the scaling parameter α, and the curvature parameter c. Although our experiments show stable improvements under the selected settings, the optimal configuration may vary across datasets, programming languages, and backbones, requiring additional tuning in new deployment scenarios.

References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. 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*, pages 2655–2668. Association for Computational Linguistics.

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *International Conference on Learning Representations*.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019a. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019b. code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.*, 3(POPL):40:1–40:29.

Kounianhua Du, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. 2024. Codegrag: Extracting composed syntax graphs for retrieval augmented cross-lingual code generation. *CoRR*, abs/2405.02355.

Fahad Ebrahim and Mike Joy. 2023. Source code plagiarism detection with pre-trained model embeddings and automated machine learning. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing, RANLP 2023, Varna, Bulgaria, 4-6 September 2023*, pages 301–309. INCOMA Ltd., Shoumen, Bulgaria.

Fahad Ebrahim and Mike Joy. 2024. Semantic similarity search for source code plagiarism detection: An exploratory study. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Milan, Italy, July 8-10, 2024*. ACM.

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*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. 2018. Hyperbolic entailment cones for learning hierarchical embeddings. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1632–1641. PMLR.

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*, pages 7212–7225. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin 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.

Patrik Hrkút, Michal Duracík, Stefan Toth, and Matej Mesko. 2023. Current trends in the search for similarities in source codes with an application in the field of plagiarism and clone detection. In *33rd Conference of Open Innovations Association, FRUCT 2023, Zilina, Slovakia, May 24-26, 2023*, pages 77–84. IEEE.

Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. 2010. Hyperbolic geometry of complex networks. *CoRR*, abs/1006.5169.

Matt Le, Stephen Roller, Laetitia Papaxanthos, Douwe Kiela, and Maximilian Nickel. 2019. Inferring concept hierarchies from text corpora via hyperbolic embeddings. 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*, pages 3231–3241. Association for Computational Linguistics.

Shangqing Liu, Xiaofei Xie, Jingkai Siow, Lei Ma, Guozhu Meng, and Yang Liu. 2021. Graphsearchnet: Enhancing gnns via capturing global dependency for semantic code search. *ACM Transactions on Software Engineering and Methodology*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. 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, and 3 others. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Maximilian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6338–6347.

Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *J. Mach. Learn. Res.*, 11:2487–2531.

Rik Sarkar. 2011. Low distortion delaunay embedding of trees in hyperbolic plane. In *Graph Drawing - 19th International Symposium, GD 2011, Eindhoven, The Netherlands, September 21-23, 2011, Revised Selected Papers*, volume 7034 of *Lecture Notes in Computer Science*, pages 355–366. Springer.

Nenad Tomasev, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2014. The role of hubness in clustering high-dimensional data. *IEEE Trans. Knowl. Data Eng.*, 26(3):739–751.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. 2020. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 261–271. IEEE.

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*, pages 8696–8708. Association for Computational Linguistics.

Zezhou Yang, Sirong Chen, Cuiyun Gao, Zhenhao Li, Xing Hu, Kui Liu, and Xin Xia. 2025. An empirical study of retrieval-augmented code generation: Challenges and opportunities. *ACM Trans. Softw. Eng. Methodol.*, 34(7):188:1–188:28.

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

A Ollivier-Ricci Curvature of ASTs

To provide further justification for the geometric assumption behind HACR, we measure the discrete curvature of ASTs built from the program-level code-to-code retrieval split of XLCoST. For each programming language, we sample up to 100 valid ASTs and compute the Ollivier-Ricci curvature on AST edges. For an edge (x, y), the curvature is defined as

$$\kappa(x, y) = 1 - \frac{W_1(m_x, m_y)}{d(x, y)}, \tag{19}$$

where W_1 denotes the Wasserstein-1 distance between the local random-walk measures m_x and m_y. Since AST edges have unit length, $\kappa(x, y) = 1 - W_1(m_x, m_y)$. When an edge connects a parent node to a subtree with branching structure, the endpoint neighborhoods can diverge across multiple child subtrees, increasing the transport cost and often yielding negative curvature.

Table 5 shows that the sampled ASTs exhibit negative average Ollivier-Ricci curvature across all seven languages, with the minimum edge curvature ranging from -1.2500 to -1.7778. Moreover, approximately 40–44% of AST edges have negative curvature across languages. These results provide empirical evidence that program syntax trees contain neighborhood-expansion patterns characteristic of negatively curved spaces. This observation is consistent with the role of hyperbolic geometry in modeling tree-like structures: the Lorentz model used in HACR has constant negative curvature and exponential volume growth, making it naturally suited to preserving the hierarchical organization of source code.

Language	ASTs	Nodes	Depth	Branching	Max Width	Avg. κ	Min. κ	Neg. Edges
C++	100	32.13	5.31	2.41	13.04	-0.2343	-1.6889	42.02%
Java	100	32.59	7.58	2.04	12.85	-0.2363	-1.7000	42.99%
Python	100	31.82	3.80	2.48	11.92	-0.3224	-1.7778	43.19%
C#	100	32.43	7.61	2.00	12.14	-0.2305	-1.5128	43.49%
JavaScript	100	28.33	5.81	2.24	9.62	-0.2407	-1.5325	41.20%
PHP	100	24.09	5.02	2.41	9.36	-0.2352	-1.5960	40.19%
C	100	15.29	5.43	1.95	7.00	-0.2041	-1.2500	44.00%

Table 5: Ollivier-Ricci curvature statistics of ASTs on seven programming languages in the XLCoST program-level code-to-code retrieval data. "Nodes", "Depth", "Branching", and "Max Width" are averaged over valid ASTs; "Neg. Edges" denotes the percentage of AST edges with negative curvature.

B Programming Language Perspective on Entailment Direction

From the perspective of programming language theory, a complete program can be viewed as a contextualized instantiation of its constituent syntactic units. Let p denote an AST snippet and let $C[\cdot]$ denote a well-formed program context, i.e., a program with a hole that can be filled by p. A complete program containing p can be written as

$$s = C[p]. \tag{20}$$

Under the principle of compositional semantics, the meaning of the complete program is obtained by interpreting the snippet under its surrounding context:

$$\mathrm{Sem}(s) = \mathrm{Sem}(C[p]) = F_C(\mathrm{Sem}(p)), \tag{21}$$

where F_C denotes the semantic transformation induced by $C[\cdot]$. The context supplies additional bindings, type constraints, control-flow structure, call relations, and possible input-output constraints. These contextual constraints specialize the behavior expressed by the snippet rather than replacing it.

This yields a natural partial order between snippets and complete programs. Let $\Gamma(p)$ denote the set of well-formed program meanings compatible with snippet p. Since $C[\cdot]$ adds contextual constraints to p, contextualization restricts the compatible meaning set:

$$\Gamma(C[p]) \subseteq \Gamma(p). \tag{22}$$

Thus, the snippet corresponds to a more general semantic object, whereas the complete program corresponds to a more specific contextualized instance. Equivalently, the complete program preserves the semantic constraints of the snippet while adding context-specific information. In a hierarchical representation space, this inclusion relation supports treating the snippet as entailing the program:

$$p \Rightarrow C[p]. \tag{23}$$

This interpretation aligns with the entailment direction used in HACR, where snippet embeddings are encouraged to entail the embeddings of their corresponding complete programs.

C Proof of the Lorentzian Ranking

$$c > 0, \quad \mathbf{x}, \mathbf{y} \in \mathcal{L}_c^n. \tag{24}$$

$$d_{\mathcal{L}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{c}} \operatorname{arccosh}\left(-c\langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}\right). \tag{25}$$

$$\frac{d}{du}\left[\frac{1}{\sqrt{c}} \operatorname{arccosh}(u)\right] = \frac{1}{\sqrt{c}\sqrt{u^2-1}} > 0, \quad u > 1. \tag{26}$$

$$\begin{aligned} \arg\min_{\mathbf{y}} d_{\mathcal{L}}(\mathbf{x}, \mathbf{y}) &= \arg\min_{\mathbf{y}} \operatorname{arccosh}\left(-c\langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}\right) \\ &= \arg\min_{\mathbf{y}} \left(-c\langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}\right) \\ &= \arg\max_{\mathbf{y}} \langle \mathbf{x}, \mathbf{y}\rangle_{\mathcal{L}}. \end{aligned} \tag{27}$$

$$\begin{aligned} d_{\mathcal{L}}(\mathbf{x}, \mathbf{y}_1) \le d_{\mathcal{L}}(\mathbf{x}, \mathbf{y}_2) &\iff -c\langle \mathbf{x}, \mathbf{y}_1\rangle_{\mathcal{L}} \le -c\langle \mathbf{x}, \mathbf{y}_2\rangle_{\mathcal{L}} \\ &\iff \langle \mathbf{x}, \mathbf{y}_1\rangle_{\mathcal{L}} \ge \langle \mathbf{x}, \mathbf{y}_2\rangle_{\mathcal{L}}. \end{aligned} \tag{28}$$

Therefore, decreasing geodesic-distance ranking is equivalent to increasing Lorentzian-inner-product ranking.

D Implementation Details

HACR is implemented in PyTorch with Hugging Face Transformers and Geoopt. We use Tree-sitter to parse source code and extract one internal AST snippet for each training instance. The snippet extractor first removes non-semantic nodes such as imports, preprocessor directives, using directives,

and comments. When a language-specific main function or method can be identified, the extractor samples a sibling or child subtree around that main node; otherwise, it samples a valid child subtree from the root. This procedure produces the entailment snippet used only during training.

For each anchor program, the data loader samples one positive program with the same label and a different index, one negative program from a different label, and the extracted snippet. Source code and snippets are tokenized with a maximum sequence length of 400. For encoder-only backbones, including RoBERTa, CodeBERT, and UniXcoder, we apply attention-mask mean pooling over the last hidden states. For CodeT5 and PLBART, we apply mean pooling over the last decoder hidden states.

The pooled Euclidean representation is scaled by a learnable factor initialized to $\alpha = 0.1$ and then lifted to the tangent space at the Lorentz origin by concatenating a zero temporal component. We map the resulting vector to the Lorentz manifold with the exponential map. The curvature parameter is initialized as $c = 1.0$ and optimized during training. Retrieval scores are computed by the Lorentz inner product, with temperature $\tau = 0.05$ in the contrastive objective.

Training minimizes the weighted sum of the hyperbolic contrastive loss and the hyperbolic entailment loss. The contrastive loss compares each anchor against its explicit positive, explicit negative, and in-batch negatives from different labels. The entailment loss enforces the extracted snippet embedding to entail both the anchor program and its positive counterpart, using an equal weight for the two constraints. We set the entailment weight to 0.01.

We optimize all parameters with RiemanniananAdam and a linear learning-rate schedule with 10% warmup. Gradients are clipped to a maximum norm of 1.0. After each epoch, the model is evaluated on the validation set, and the checkpoint with the best validation MAP is saved. During evaluation and testing, only full-program embeddings are used; snippets are not included in the retrieval index or query representation. The test script reloads the best validation checkpoint and reports MAP-based retrieval metrics.

Hyperparameter	Value
Epochs	2
Maximum sequence length	400
Training batch size	8
Evaluation batch size	16
Learning rate	2×10^{-5}
Warmup ratio	0.1
Maximum gradient norm	1.0
Weight decay	0.0
Initial curvature c	1.0
Initial scale α	0.1
Entailment weight	0.01
Seed	123456

Table 6: Main implementation hyperparameters used for HACR training and evaluation.

Scientific artifacts and intended use Our experiments use two existing public benchmarks, POJ-104 and XLCoST, together with publicly released pre-trained backbones, namely RoBERTa, CodeBERT, GraphCodeBERT, PLBART, CodeT5, and UniXcoder. The benchmark statistics used in this paper are summarized in Section 4: POJ-104 contains 52,000 programs from 104 functional classes, while XLCoST covers seven programming languages for cross-lingual code retrieval. We use these artifacts only for academic research on code retrieval, which is consistent with their intended benchmark use. We do not introduce a new dataset or pre-train a new foundation model from scratch in this work.

Licensing, documentation, and data content The datasets and model checkpoints used here are distributed by their original creators and are widely used in prior code intelligence research. For the two benchmarks used in this paper, XLCoST is distributed under CC BY-SA 4.0, and the POJ-104 benchmark is used through the CodeXGLUE release, whose datasets follow the Computational Use of Data Agreement (C-UDA) license. The licenses of the pre-trained backbones may differ across original releases, so users should consult the official model pages before redistribution or downstream deployment. In the manuscript, we cite the original sources of the datasets and pre-trained models in Section 4. The paper also documents the main benchmark characteristics needed for reproducibility, including the number of programs, functional classes, programming languages, evaluation metrics, and model sizes. We follow the standard benchmark splits provided by the original dataset releases.

These benchmarks contain source code rather than direct human-subject records. We therefore

do not intentionally process personal profiles or demographic attributes. However, publicly available code corpora may still contain incidental identifiers or other repository-specific strings. In our snippet extraction pipeline, we remove comments and several non-semantic nodes from the extracted training snippets to reduce unnecessary exposure to such content, although this preprocessing does not guarantee the removal of every potentially sensitive string in the underlying corpora.

Computational budget, statistics, and software All experiments in this paper are fine-tuning runs on pre-trained models rather than large-scale pre-training runs. The model sizes used in our study range from 125M to 220M parameters, as reported in Tables 1 and 2. The experiments were conducted on three NVIDIA A800 GPUs. The main optimization hyperparameters are listed in Table 6. For cross-lingual retrieval on XLCoST, we report mean and standard deviation over three random seeds, as shown in Table 2. For monolingual retrieval on POJ-104, we report the retrieval scores from a single fixed seed, using the configuration in Table 6. The software environment includes gdown 5.2.0, geoopt 0.5.0, matplotlib 3.9.4, numpy 1.23.5, pandas 2.2.0, scikit-learn 1.6.1, scipy 1.13.1, seaborn 0.13.2, sentencepiece 0.2.1, tensorboard 2.11.0, tokenizers 0.13.3, torch 2.3.0, tqdm 4.67.1, transformers 4.30.2, and tree-sitter 0.20.4. We will release the code and environment specification to support reproducibility.

Potential risks and broader impact HACR is designed for research on code retrieval, but code retrieval systems can also be misused to accelerate plagiarism, license-violating reuse, or the discovery of vulnerable code patterns. In addition, the two benchmarks used here are derived from existing public resources and may inherit biases in language coverage, coding style, problem distribution, and repository conventions. As a result, strong benchmark performance should not be interpreted as guaranteeing equal performance across underrepresented programming languages, private industrial codebases, or safety-critical deployment settings. These considerations motivate careful downstream evaluation and license compliance before practical deployment.

Use of AI assistants We used ChatGPT only for limited English polishing and sentence-level rewriting during manuscript preparation. It was not used to design experiments, generate results, analyze findings, or determine the scientific conclusions of this work.