GeoFusion: Entanglement-Guided Fusion of Geometry-aware Representations and Spectral Priors for Code Summarization

Anonymous authors
Paper under double-blind review

Abstract

Code summarization aims to generate concise and accurate natural language descriptions of source code. Recent methods already exploit multi-source heterogeneous program information, yet relations with distinct structures are still placed in one Euclidean space, and their geometric properties receive limited attention. We propose GeoFusion, an entanglement-guided method that fuses geometry-aware representations with spectral priors. GeoFusion encodes abstract syntax trees (ASTs) in a Poincaré ball to preserve hierarchical syntax. Data-flow and sequential relations remain in Euclidean space, which better fits non-hierarchical dependence. To align programs with natural language, GeoFusion organizes summary semantics, source-code functionality, and line-level comment functionality in a corpus-level spectral taxonomy graph, and derives a spectral taxonomy prior. Entanglement-guided adaptive fusion then uses coupling entropy to regulate contributions across sources and within each source. Experiments on Java and Python datasets show that GeoFusion achieves the state-of-the-art performance, improving existing methods by 4.61, 4.27 and 3.02 in terms of BLEU, ROUGE-L and METEOR on Java, and by 3.78, 3.52 and 2.52 on Python.

1 Introduction

Code summarization aims to generate concise and accurate natural language descriptions of source code. Early neural approaches mainly modeled programs as token sequences. Later studies incorporated abstract syntax trees (ASTs), data-flow relations, and pretrained code representations Wu et al. (2021); Alon et al. (2018); Guo et al. (2020). Recent methods further combine multiple structures. For example, Guo et al. (2024) organizes heterogeneous program relations in one code graph so that they can interact. These studies show that multi-source information helps code summarization.

They still, however, place structurally different relations in one Euclidean space. This choice gives limited attention to geometry. ASTs are hierarchical and expand by recursive branching. Data-flow and code-order relations are non-hierarchical and record dependence and sequential continuity. A shared flat space can therefore mismatch the geometry of the embedding to the structure of each relation. This raises a central question. **Should each program relation be represented in a space whose geometry matches its structure?**

Figure 1 illustrates this mismatch on the same AST. In Euclidean space, the room available at a given distance from the origin grows only polynomially with that distance, whereas a branching tree can add nodes exponentially with depth. Deep nodes and different subtrees can therefore become crowded and harder to separate in a low-dimensional representation. In the Poincaré ball, this room grows exponentially with hyperbolic distance from the origin, providing greater capacity to accommodate hierarchical expansion and to separate different subtrees. This geometric compatibility suggests that **program relations with different structures may benefit from spaces of different geometry**.

We propose GeoFusion, an entanglement-guided method that fuses geometry-aware representations with spectral taxonomy priors. GeoFusion first encodes ASTs in a Poincaré ball and keeps data-flow and sequential relations in Euclidean space, matching each relation to a suitable geometry. It then builds a corpus-level spectral taxonomy graph over summary semantics, source-code functionality,

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.

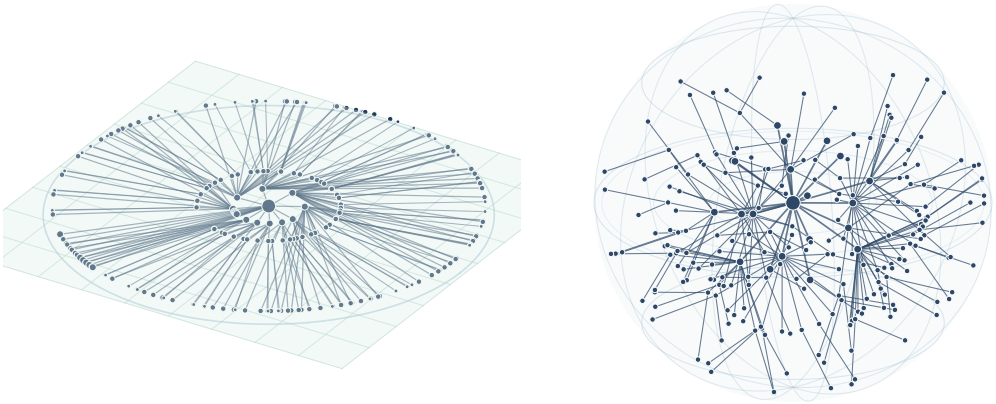

Figure 1: The same AST in Euclidean space and in a Poincaré ball.

and line-level comment functionality, and derives a spectral taxonomy prior to align programs with natural language. Entanglement-guided adaptive fusion uses coupling entropy to regulate contributions across sources and within each source.

The main contributions of this work are summarized as follows:

- We propose a geometry-aware representation for code summarization. Tree-structured AST relations are placed in Poincaré space, and non-hierarchical dependency and sequential relations remain in Euclidean space.
- We construct a spectral taxonomy graph over summary semantics, source-code functionality, and line-level comment functionality, and derive a spectral taxonomy prior. We further introduce entanglement-guided adaptive fusion, which uses coupling entropy to integrate these views across sources and within each source.
- We conduct extensive experiments on Java and Python datasets. GeoFusion achieves the state-of-the-art performance, improving existing methods by 4.61, 4.27 and 3.02 in terms of BLEU, ROUGE-L and METEOR on Java, and by 3.78, 3.52 and 2.52 on Python.

2 Related Work

Source Code Summarization Early studies on code summarization mainly relied on information retrieval Eddy et al. (2013); Wong et al. (2015). With the advances in deep learning, researchers began to employ Seq2Seq models, attention mechanisms, and Transformers to generate summaries from code sequences Ahmad et al. (2020). To overcome the limited capability of sequential representations in capturing program structures, subsequent studies incorporated ASTs, data flow graphs (DFGs), and other structural features Feng et al. (2020); Wang et al. (2021); Alon et al. (2018); Guo et al. (2020). For example, Guo et al. (2024) organizes multiple sources of program information into a heterogeneous code graph. Overall, existing studies have continuously enriched the information sources, but have primarily focused on how to incorporate and organize heterogeneous program information, while paying limited attention to whether different program structures exhibit distinct intrinsic geometries.

Neural Code Representation Neural networks have been widely adopted to learn sequential and structural representations of source code. Existing approaches typically enhance structural interactions through different node and edge types, tree-based architectures, or attention mechanisms Huynh Nguyen et al. (2024); Yang & Zhu (2025); Chen et al. (2026), while most of them still learn program representations within a unified Euclidean space. Meanwhile, studies on hyperbolic representation learning have shown that negatively curved spaces can represent tree-structured data with hierarchical expansion at relatively low distortion Nickel & Kiela (2017); Chami et al. (2019).In contrast to existing approaches, GeoFusion matches different program structures with representation spaces according to their intrinsic geometry properties.

3 GEOFUSION

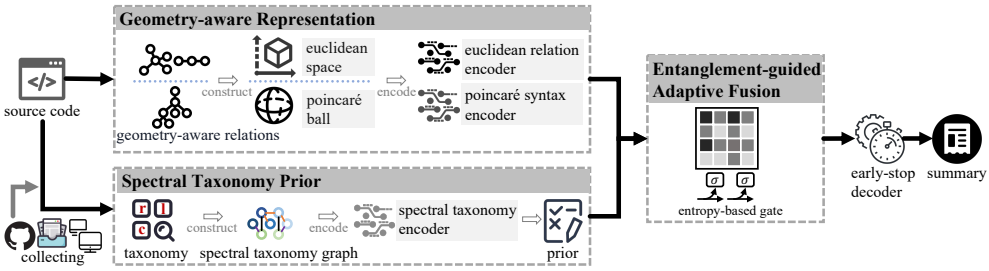

Figure 2: The overall framework of GeoFusion.

Figure 2 presents the architecture of GeoFusion, which consists of three main components: geometry-aware representation, spectral semantic prior, and entanglement-guided adaptive fusion. Given a source code snippet x, GeoFusion first extracts multiple heterogeneous program relations, including AST, DFG, code-order relations, and sibling-order relations, and maps them into representation spaces according to their structural geometric properties. Specifically, ASTs are modeled in Poincaré space, whereas data-flow and ordering relations are encoded in Euclidean space. Beyond structural representation, GeoFusion further constructs a corpus-level spectral taxonomy prior from code functionality, line-level comments functionality, and summary semantics, to provide global semantic guidance for cross-view alignment between programming language and natural language. Finally, the entanglement-guided adaptive fusion module estimates the coupling state among heterogeneous information sources and adaptively regulates the contribution of different views using spectral-entropy signals. The fused representation is then fed into the decoder to generate the summary.

3.1 GEOMETRY-AWARE REPRESENTATION

Geometry-aware Relation Construction Given a node set V, GeoFusion extracts four complementary program relations: AST $\mathcal{E}_{\text{ast}}$, DFG $\mathcal{E}_{\text{dfg}}$, code order $\mathcal{E}_{\text{seq}}$, and sibling order $\mathcal{E}_{\text{sib}}$. According to their structural geometry, they are partitioned into a hyperbolic relation set $\mathcal{E}_{\mathbb{B}}$, modeled in Poincaré space to preserve tree-like hierarchy, and a Euclidean relation set $\mathcal{E}_{\mathbb{R}}$, modeled in Euclidean space to preserve dependency and ordering relations:

$$\mathcal{G}_{\text{geo}} = (V, \mathcal{E}_{\mathbb{B}}, \mathcal{E}_{\mathbb{R}}), \qquad \mathcal{E}_{\mathbb{B}} = \mathcal{E}_{\text{ast}}, \qquad \mathcal{E}_{\mathbb{R}} = \mathcal{E}_{\text{dfg}} \cup \mathcal{E}_{\text{seq}} \cup \mathcal{E}_{\text{sib}} \tag{1}$$

Each relation is stored in both directions, so the instance graph has eight directed edge types on the same V. The construction is detailed in Appendix A.1.

Poincaré Syntax Encoder (PSE) GeoFusion models the AST in a p-dimensional Poincaré ball $\mathbb{B}_c^p = \left\{ z \in \mathbb{R}^p : c\|z\|_2^2 < 1 \right\}$ with curvature $-c\,(\,c > 0)$. For an AST node v, let d_v denote its depth from the root and $d_{\max}$ the maximum depth of the current AST. At layer ℓ, let $\hat{h}_v^{(\ell)} \in \mathbb{R}^p$ denote the representation of node v in the tangent space at the origin. PSE jointly determines the node position in the Poincaré ball from its normalized depth $\delta_v = \frac{d_v}{\max(d_{\max}, 1)}$ and node representation:

$$r_v^{(\ell)} = r_{\min} + (r_{\max} - r_{\min})\sigma\left(\alpha(2\delta_v - 1) + w_r^\top \hat{h}_v^{(\ell)}\right) \qquad z_v^{(\ell)} = \frac{r_v^{(\ell)}}{\sqrt{c}} \frac{\hat{h}_v^{(\ell)}}{\max(\|\hat{h}_v^{(\ell)}\|_2, \epsilon)} \tag{2}$$

Here, $r_v^{(\ell)}$ is the normalized radial coefficient of node v, $r_{\min}$ and $r_{\max}$ bound its valid range, α and w_r are learnable parameters, $\sigma(\cdot)$ is the sigmoid function, and ϵ is used for numerical stability. The depth signal encourages nodes to expand gradually from the root toward deeper levels, while the node representation further determines their specific radial positions and directions, avoiding overly rigid placement of nodes at the same depth. PSE aggregates parent- and child-directed AST information in the tangent space and maps the updated states back to the Poincaré ball. After multiple layers, the hyperbolic syntax representations $Z_{\mathbb{B}}$ are mapped to the tangent space and scaled by s_{ast}, yielding the tangent-space AST representation $H_{\text{ast}} = s_{\text{ast}} \log_0^c(Z_{\mathbb{B}})$. Depth is not a fixed radius, it

means that nodes at the same depth can still take different radii through $w_r^\top \hat{h}_v^{(\ell)}$, and their directions keep different branches from collapsing into one region.

To enable subsequent propagation, GeoFusion further adapts H_{ast} to the original Euclidean node states. Let e_v denote the initial Euclidean representation of node v, and $h_{\text{ast},v}$ its corresponding tangent-space AST representation. We define

$$\phi_v = [e_v; h_{\text{ast},v}; |e_v - h_{\text{ast},v}|; e_v \odot h_{\text{ast},v}] \tag{3}$$

$$\bar{e}_v = \text{GraphNorm}\left(e_v + \sigma\left(w_g^\top \phi_v + b_g\right) h_{\text{ast},v}\right) \tag{4}$$

Here, ϕ_v combines the original representation, the AST representation, their difference, and their element-wise interaction; w_g and b_g are learnable parameters; $\odot$ denotes element-wise multiplication; and $\bar{e}_v$ is the adapted node representation. This mechanism allows different nodes to selectively incorporate AST information according to their local states and provides a unified representation for subsequent Euclidean relation encoding. The gate is a per-node scalar. For other details, see Appendix A.2.

Euclidean Relation Encoder (ERE) After obtaining hierarchy-enhanced node representations, GeoFusion employs ERE to model dependency and ordering relations. Specifically, DFG and code-orders form the main propagation messages $m_{v,\text{main}}^{(\ell)}$, while sibling-orders provide complementary local ordering information $m_{v,\text{sib}}^{(\ell)}$ among nodes sharing the same syntactic parent. The Euclidean representation $h_v^{(\ell)}$ of node v at layer ℓ is:

$$h_v^{(\ell+1)} = \text{GraphNorm}\left(h_v^{(\ell)} + \text{Dropout}\left(\text{ReLU}\left(m_{v,\text{main}}^{(\ell)} + \sigma(\gamma_{\text{sib}}) m_{v,\text{sib}}^{(\ell)}\right)\right)\right) \tag{5}$$

Here, γ_{sib} is a learnable parameter whose sigmoid value controls the relative contribution of sibling-order information. This design allows DFG and code-orders to carry the primary dependency and local sequential signals, while preventing sibling-orders from dominating simply because they share a syntactic parent. After multiple propagation layers, ERE produces the final Euclidean relation representation H_{rel}, which is subsequently used for semantic-prior integration and multi-source fusion.

3.2 Spectral Taxonomy Prior (STP)

Geometry-aware representation primarily captures structural relations within individual programs, whereas code summarization also requires modeling the cross-view correspondence between programming-language and natural-language. To establish a category space with clear semantic boundaries, we consolidate category definitions from prior studies and public datasets on code comment classification and code classification. We further analyze the reference summaries, source code, and line-level comments generated following Tang et al. (2026) in the target corpus, merge semantically overlapping categories, clarify the boundaries between easily confused categories, and introduce fallback categories for instances that cannot be reliably classified.

The resulting taxonomy contains three complementary but independently assigned views. Reference summaries are divided into six semantic-intent categories according to the information they convey, whereas source code and line-level comments are each divided into eleven functional categories according to their primary program functionality. These three views characterize what a summary expresses, what functionality the source code implements, and what local functionality is revealed by line-level comments, respectively, thereby providing a category-level bridge between programming-language and natural-language semantics. Detailed category definitions are provided in Appendix A.6 Table 3.

Spectral Taxonomy Graph (STG) For each sample i, the classification results from the three views are represented as a category triplet $q_i = \left(r_i, f_i^{\text{code}}, f_i^{\text{lineC}}\right)$, where r_i denotes the semantic category of the reference summary, and f_i^{code} and f_i^{lineC} denote the functional categories of the source code and line-level comments, respectively.

GeoFusion constructs a STG from category-association statistics over the training corpus. For each sample, three undirected relations are established: between the reference-summary category and the source-code category, between the reference-summary category and the line-comment category, and between the source-code category and the line-comment category. A category pair receives a

larger edge weight when it is observed more frequently across the corpus. Consequently, STG captures corpus-level relationships among semantic and functional categories, such as which summary semantics are commonly associated with a particular code functionality and how local functions reflected by line-level comments relate to the overall functionality of the source code.

Spectral Taxonomy Encoder (STE) Let A and D denote the weighted adjacency matrix and degree matrix of STG, respectively. The STE first constructs the normalized graph Laplacian and extracts its low-frequency spectral structure through eigendecomposition:

$$L_{\text{tax}} = I - D^{-1/2}AD^{-1/2} \qquad L_{\text{tax}}U = U\Lambda \tag{6}$$

Let U_K contain the eigenvectors associated with the K smallest eigenvalues. For category a, its gated spectral coordinate is defined as $\psi_a = U_K[a] \odot \sigma(\theta)$, where $U_K[a]$ denotes the spectral coordinate of category a in the low-frequency subspace, θ is a learnable spectral-dimension gating parameter, and $\sigma(\cdot)$ denotes the sigmoid function. STE then combines the spectral coordinates with view-specific category embeddings to obtain the taxonomy representation of sample i: $s_i^{\text{tax}} = \text{LN}\left(\text{MLP}\left[E_C(r_i) + W_C\psi_{r_i}; E_F^{\text{code}}(f_i^{\text{code}}) + W_F\psi_{f_i^{\text{code}}}; E_F^{\text{lineC}}(f_i^{\text{lineC}}) + W_F\psi_{f_i^{\text{lineC}}}\right]\right)$. Here, E_C denotes the category embedding for reference summaries, E_F^{code} and E_F^{lineC} denote the functional-category embeddings for source code and line-level comments, respectively, and W_C and W_F are projection matrices for the spectral coordinates. Since source code and line-level comments share the same functional taxonomy, they use the same category-level spectral coordinates while retaining separate category embeddings to preserve view-specific information. The resulting s_i^{tax} serves as the STP for sample i, jointly encoding its category assignments across the three views and their global association structure in the corpus. STP is then gated into each node before ERE:

$$g_{iv}^{\text{tax}} = \sigma\big((w_x^{\text{tax}})^\top h_{iv} + (w_s^{\text{tax}})^\top s_i^{\text{tax}} + (w_d^{\text{tax}})^\top E_d(\min(d_{iv}, 63)) + b_{\text{tax}}\big) \tag{7}$$

$$\hat{h}_{iv} = \text{GraphNorm}\big(h_{iv} + g_{iv}^{\text{tax}} s_i^{\text{tax}}\big) \tag{8}$$

The adjacency of STG and the spectral coordinates are specified in Appendix A.3.

3.3 Entanglement-Guided Adaptive Fusion (EAF)

Different structural and semantic sources contribute differently to summary generation, so GeoFusion introduces EAF, which explicitly models the coupling among heterogeneous sources and dynamically adjusts their fusion. For each sample i, EAF collects multiple representations p_i^k. In the full model, the sources are the graph state, the code-token state, STP, the AST tangent state, and pooled states of the AST-base, sibling, DFG, and sequence families, so coupling is measured across sources and within each source. These representations are projected into a shared space and normalized to construct a cross-source coupling matrix:

$$q_i^k = \frac{W_k^q p_i^k}{\|W_k^q p_i^k\|_2 + \epsilon} \qquad \rho_{i,ab} = \frac{\left|(q_i^a)^\top q_i^b\right|}{\sum_{m=1}^S \sum_{n=1}^S |(q_i^m)^\top q_i^n| + \epsilon} \tag{9}$$

where S denotes the number of information sources and $\rho_{i,ab}$ measures the relative coupling strength between sources a and b. EAF further summarizes the overall coupling distribution using normalized entropy: $\mathcal{H}_i = -\frac{1}{\log(S^2)} \sum_{a=1}^S \sum_{b=1}^S \rho_{i,ab} \log(\rho_{i,ab} + \epsilon)$.

The multi-source representations and $\mathcal{H}_i$ are then jointly encoded into a shared context, whose residual contribution is adaptively controlled by an entropy-based gate:

$$c_i = \text{LN}\left(\text{MLP}\left[\frac{1}{S}\sum_{k=1}^S W_k^c p_i^k; \mathcal{H}_i\right]\right) \qquad g_i^{\text{ent}} = \sigma\left(b_{\text{ent}} + w_H(\tau - \mathcal{H}_i)\right) \tag{10}$$

$$\tilde{H}_{\text{rel},i} = H_{\text{rel},i} + g_i^{\text{ent}} \,\text{Broadcast}(c_i) \qquad \tilde{H}_{\text{code},i} = H_{\text{code},i} + g_i^{\text{ent}} \,\text{Broadcast}(c_i) \tag{11}$$

where τ denotes the target coupling entropy. g_i^{ent} is a sample-level scalar residual gate. It does not replace H_{rel} or H_{code}, and a larger $\mathcal{H}_i$ does not mean stronger fusion. Pooling and the entropy regularizer are given in Appendix A.4.

3.4 EARLY-STOP DECODER (ESD)

To reduce the computational cost of summary generation, GeoFusion employs a ESD, which terminates decoding at the earliest layer whose prediction confidence exceeds a predefined threshold:

$$\kappa_t^{(\ell)} = \frac{1}{B}\sum_{b=1}^{B} \max_y P_{b,t}^{(\ell)}(y) \qquad e_t = \min\left\{\ell \in \mathcal{D} : \ell \geq \ell_{\min} \wedge \kappa_t^{(\ell)} \geq \delta\right\} \tag{12}$$

where B is the batch size and δ is the confidence threshold. If no candidate layer satisfies the condition, the final layer L is used. The exit drops only later decoder layers at the current generation step. The summary sequence continues, and the chosen state still goes through the copy generator. Details is described in Appendix A.5 and A.7.

4 EXPERIMENTS

4.1 SETUP

We evaluate GeoFusion on Java Hu et al. (2018) and Python Wan et al. (2018). The Java dataset contains 69,708/8,714/8,714 samples for training, validation, and testing, respectively, while the corresponding split of the Python dataset is 55,538/18,505/18,502. Following prior studies Guo et al. (2024); Tang et al. (2026), we adopt BLEU Papineni et al. (2002), METEOR Banerjee & Lavie (2005), and ROUGE-L Lin (2004) as automatic evaluation metrics. GeoFusion is trained using the Adam optimizer Kingma & Ba (2015) with an initial learning rate of 5×10^{-4}, which is decayed by 5%. The dropout rate is set to 0.2, and the batch sizes are 200 and 300 for Java and Python, respectively. All experiments are conducted on an NVIDIA GeForce RTX 5090 GPU. For more details about baselines, please refer to Appendix A.8.

4.2 RQ1: HOW EFFECTIVE IS GEOFUSION FOR CODE SUMMARIZATION?

Table 1: Performance comparison between GeoFusion and baselines.

Model	Java			Python		
	BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
CodeNN Iyer et al. (2016)	16.71	16.85	42.51	17.63	11.17	36.87
Code2Seq Alon et al. (2018)	15.69	14.86	22.1	20.72	10.92	34.19
ASTNN Zhang et al. (2019)	26.68	23.78	32.47	30.54	20.42	37.85
CoCoAST Shi et al. (2023)	36.39	23.36	46.29	31.36	21.74	45.97
CodeBERT Feng et al. (2020)	35.97	23.29	48.23	30.72	21.53	47
CodeT5 Wang et al. (2021)	31.96	23.31	49.87	32.57	22.26	49.2
UniXCoder Guo et al. (2022)	43.56	27.83	61.75	31.89	21.29	45.59
GraphCodeBERT Guo et al. (2020)	36.74	22.83	55.61	31.73	20.91	45
ChatGPT OpenAI (2023)	26.52	28.61	48.69	30.17	22.28	40.34
HierarchyNet Huynh Nguyen et al. (2024)	45.16	29.25	53.24	28.17	16.1	39.57
Hetcos Guo et al. (2024)	46.75	33.16	59.67	34.47	25.58	51.66
GeoFusion	**51.36**	**36.18**	**63.94**	**38.25**	**28.1**	**55.18**
-w/o Geo	49.33	35.1	61.79	37.23	27.07	54.07
-w/o STP	48.67	34.63	60.84	35.75	25.75	51.9
-w/o EAF	49.56	35.89	61.72	37.03	26.94	53.82
-w/o ESD	50.86	36.04	62.76	37.97	27.89	54.73

Table 1 reports the comparison between GeoFusion and the baseline methods on the Java and Python datasets. Overall, GeoFusion achieves the best performance across all three metrics on both datasets. Compared with sequence- and structure-based methods, including CodeNN, Code2Seq, ASTNN, and CoCoAST, GeoFusion consistently obtains higher scores on all metrics. Against Hetcos, the strongest heterogeneous-structure baseline, GeoFusion improves BLEU, METEOR, and ROUGE-L by 4.61%, 3.02%, and 4.27% on Java, and by 3.78%, 2.52%, and 3.52% on Python. Notably, HierarchyNet also emphasizes hierarchical and heterogeneous program representations, but its performance varies considerably across the two datasets, particularly on Python. GeoFusion also outperforms pre-trained code models, including CodeBERT, CodeT5, UniXCoder, and GraphCodeBERT, as well as the ChatGPT baseline. Overall, the results demonstrate that GeoFusion consistently improves code summarization across different model families and programming languages, supporting the effectiveness of geometry-aware representation and adaptive heterogeneous information fusion.

4.3 RQ2: What is the contribution of each key component?

To evaluate the contribution of each component, we construct four variants: w/o Geo replaces the Poincaré AST representation with a unified Euclidean representation; w/o STP removes the spectral taxonomy prior; w/o EAF replaces EAF with standard attention; and w/o ESD disables dynamic early-stop decoding. As shown in Table 1, removing any component degrades performance. STP has the largest impact, reducing ROUGE-L by 3.10 and 3.28 points on Java and Python, respectively, which confirms the benefit of corpus-level semantic associations for cross-view alignment. The consistent degradation of w/o Geo supports the use of geometry-matched representations, while the advantage over w/o EAF shows that coupling-guided fusion is more effective than standard attention for integrating heterogeneous information. ESD mainly improves efficiency: although disabling it causes only minor performance changes, ESD reduces the average time per epoch from 9min34s to 2min35s on Java and from 5min17s to 1min34s on Python, corresponding to reductions of 73.0% and 70.3%, respectively.

Table 2: Performance comparison of geometry-aware representation, STP, and EAF on the Python dataset.

Models	BLEU	METEOR	ROUGE-L
GeoFusion	**38.25**	**28.1**	**55.18**
All-Euclidean	37.23	27.07	54.07
All-Hyperbolic	36.18	26.44	52.79
w/o STP	35.75	25.75	51.9
w/o Code Prior	37.14	26.99	53.98
w/o Summary Prior	36.09	26.32	52.23
w/o Line-level Comment Prior	36.94	26.97	53.86
Concat Fusion	37.03	26.94	53.82
Attention Fusion	37.12	27.04	53.98

4.4 RQ3: Does geometry-aware representation better preserve program structures?

Table 2 compares three geometric configurations: GeoFusion, All-Euclidean, and All-Hyperbolic. All-Euclidean encodes all program relations in Euclidean space, whereas All-Hyperbolic represents all of them in hyperbolic space. On the Python dataset, GeoFusion outperforms both alternatives, indicating that the performance gain comes from matching different program relations with appropriate geometry presentation spaces rather than simply introducing hyperbolic or Euclidean geometry. The structural analysis in Figure 3 further supports this observation. For the depth-radius correlation in Figure 3(a), the y-axis reports the Spearman correlation between AST depth and the norm of the corresponding node representation; a larger positive value indicates a stronger monotonic correspondence between syntactic depth and radial position. Both GeoFusion and All-Hyperbolic achieve a median correlation of approximately 0.939, substantially higher than 0.395 for All-Euclidean, showing that Poincaré representations effectively encode AST depth as a radial hierarchy.

For Figure 3(b), the y-axis measures the difference between the average cross-subtree and same-subtree angles, and a larger positive value indicates clearer angular separation between different subtrees. GeoFusion achieves a positive median gap of 2.168°, whereas All-Euclidean is close to zero and All-Hyperbolic yields -0.149°, indicating better preservation of branch-level structure. Figure 3(c) evaluates tree-distance preservation using the Spearman correlation between AST shortest-path distance and representation-space distance, where a higher value indicates better preservation of the original tree metric. GeoFusion reaches 0.408, compared with 0.314 for All-Hyperbolic and -0.176 for All-Euclidean. Finally, Figure 3(d) reports the relative distance gap between cross-subtree and same-subtree node pairs; positive values indicate that nodes from different subtrees are, on average, farther apart than nodes within the same subtree. GeoFusion is the only variant with a positive median value (0.052). Overall, these results show that Poincaré geometry is effective for preserving AST hierarchy and tree-level distances, Euclidean is more suitable for non-hierarchical relations such as DFG, code order, and sibling order.

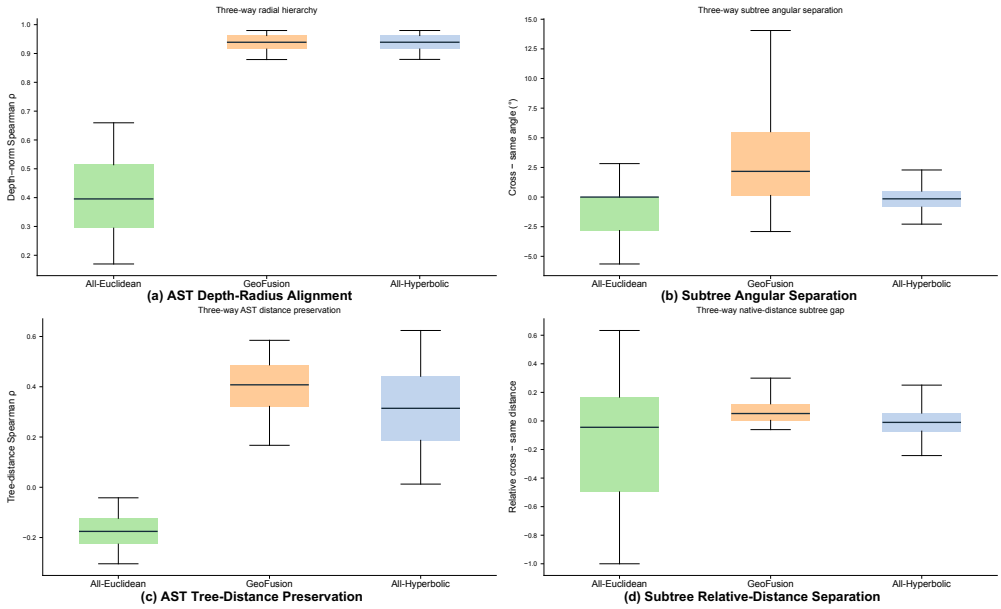

Figure 3: Comparison of AST structural preservation under different geometric configurations on Python: depth-radius correlation, cross-subtree angular separation, tree-distance preservation, and subtree distance separation.

4.5 RQ4: Does the spectral taxonomy provide an effective cross-view prior?

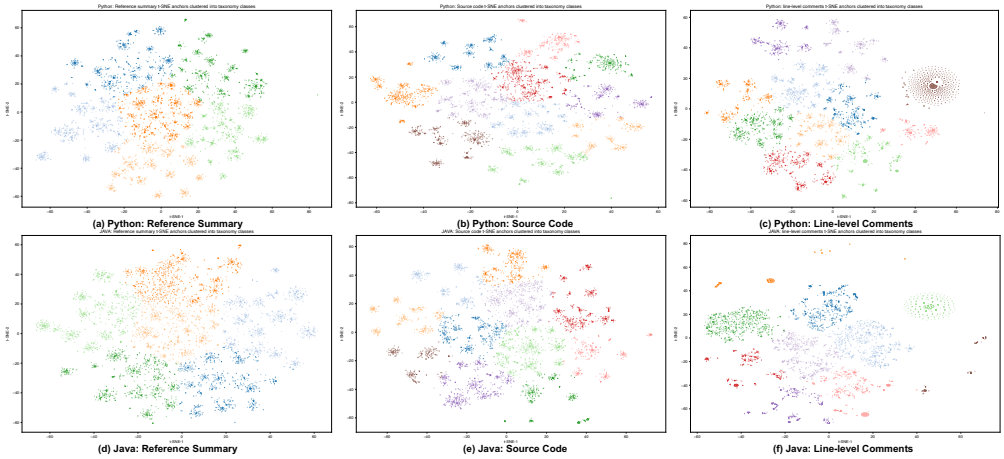

Figure 4: t-SNE visualization of the taxonomy across reference summary, source code, and line-level comment views on the Java and Python datasets. Colors denote different semantic or functional categories, with detailed category definitions and color mappings provided in Appendix A.9, Table 4.

Table 2 first evaluates the contribution of the semantic taxonomy from different views (w/o STP, w/o Code Prior, w/o Summary Prior, w/o Line-level Comment Prior). Removing the complete STP (w/o STP) decreases BLEU, METEOR, and ROUGE-L to 35.75/25.75/51.90, demonstrating the overall benefit of incorporating corpus-level semantic associations. Removing any individual view also degrades performance, while the largest reduction occurs when the summary prior is removed (w/o Summary Prior), with BLEU and ROUGE-L dropping by 2.16 and 2.95 points, respectively. The code and line-comment priors provide additional and complementary improvements.

We further examine whether the proposed taxonomy is consistent with the underlying semantic distribution of the data. Figure 4 visualizes the three views using t-SNE. All three views exhibit distinguishable category structures on both Java and Python. Quantitatively, the silhouette scores for the reference-summary, code, and line-level comment views are 0.368, 0.406, and 0.439 on Python, and 0.385, 0.369, and 0.486 on Java, respectively. The consistent separation across languages and views suggests that the proposed taxonomy scheme is not merely derived from manually consolidated category definitions, but also reflects the semantic organization observed in the data. Together with the performance degradation caused by removing STP or any individual prior, these results support the effectiveness of the taxonomy as a cross-view semantic prior.

4.6 RQ5: Does entanglement-guided fusion enable effective adaptive fusion?

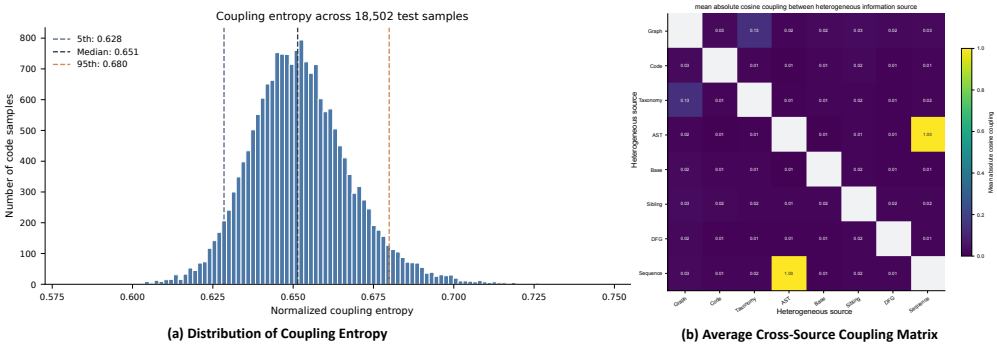

Figure 5: Coupling behavior of EAF on Python: coupling-entropy distribution and cross-source coupling matrix.

We first compare EAF with two commonly used multi-source fusion strategies, namely feature concatenation (Concat Fusion) and attention-based fusion (Attention Fusion). As shown in Table 2, EAF outperforms (Concat Fusion) and (Attention Fusion). These results indicate that explicitly modeling the coupling states among heterogeneous information sources enables more effective utilization of multi-source program information.

We further examine the internal coupling behavior of EAF. As shown in Figure 5, the 5th, 50th, and 95th percentiles of the coupling-entropy distribution on the Python are 0.63, 0.65, and 0.68, respectively, suggesting that the learned coupling patterns vary across instances. The cross-source coupling heatmap further reveals a highly non-uniform correlation structure. We can find that the average coupling strength between AST and Sequence reaches 0.9967, substantially higher than the 0.1332 observed between Graph and Taxonomy. This result shows that some information sources share highly overlapping structural evidence, whereas others provide relatively independent structural or semantic signals, supporting the need to explicitly model cross-source correlations during adaptive fusion.

5 Conclusion

This paper presents GeoFusion, an entanglement-guided method that fuses geometry-aware representations with spectral priors for code summarization. GeoFusion encodes abstract syntax trees (ASTs) in a Poincaré ball to preserve hierarchical syntax. Data-flow and sequential relations remain in Euclidean space, which better fits non-hierarchical dependence. To align programs with natural language, GeoFusion organizes summary semantics, source-code functionality, and line-level comment functionality in a corpus-level spectral taxonomy graph, and derives a spectral taxonomy prior. Entanglement-guided adaptive fusion then uses coupling entropy to regulate contributions across sources and within each source. Experiments on Java and Python datasets show that GeoFusion achieves the state-of-the-art performance, improving existing methods by 4.61, 4.27 and 3.02 in terms of BLEU, ROUGE-L and METEOR on Java, and by 3.78, 3.52 and 2.52 on Python.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing and assistance with code generation. The AI-assisted text was reviewed and revised by the authors to ensure clarity, accuracy, and consistency with the intended technical content. AI-generated or AI-assisted code was manually inspected, tested, and validated before use in the experiments. We take full responsibility for the final content of this work, including all text, and codes produced with the aid of generative AI.

REFERENCES

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4998–5007, 2020.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL https://aclanthology.org/W05-0909.

Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 4868–4879, 2019.

Zeqi Chen, Zhaoyang Chu, Yi Gui, Feng Guo, Yao Wan, and Chuan Shi. CGBridge: Bridging code graphs and large language models for better structure-aware code understanding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8945–8966. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.434. URL https://aclanthology.org/2026.findings-acl.434/.

Brian P Eddy, Jeffrey A Robinson, Nicholas A Kraft, and Jeffrey C Carver. Evaluating source code summarization techniques: Replication and expansion. In *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 13–22. IEEE, 2013.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7212–7225, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.499. URL https://aclanthology.org/2022.acl-long.499/.

Juncai Guo, Jin Liu, Xiao Liu, and Li Li. Summarizing source code through heterogeneous feature fusion and extraction. *Information Fusion*, 103:102058, 2024. doi: 10.1016/j.inffus.2023.102058.

Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. Summarizing source code with transferred api knowledge. IJCAI, 2018.

Minh Huynh Nguyen, Nghi D. Q. Bui, Truong Son Hy, Long Tran Thanh, and Tien N. Nguyen. HierarchyNet: Learning to summarize source code with heterogeneous representations. In *Findings of the Association for Computational Linguistics: EACL 2024*, pp. 2355–2367. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-eacl.156. URL https://aclanthology.org/2024.findings-eacl.156/.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2073–2083, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1195. URL https://aclanthology.org/P16-1195/.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015. URL http://arxiv.org/abs/1412.6980.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.

OpenAI. Gpt-4 technical report, 2023. URL https://arxiv.org/abs/2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. Cocoast: representing source code via hierarchical splitting and reconstruction of abstract syntax trees. *Empirical Software Engineering*, 28(6):135, 2023.

Yijia Tang, YaoShen Yu, Zhiqiu Huang, Bowei Xia, and Yukun Cao. P2n2s: Bridging the gap between natural and programming languages for code summarization via large language models. *World Wide Web*, 29(1):7, 2026.

Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, pp. 397–407, 2018.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 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*, pp. 8696–8708, 2021.

Edmund Wong, Taiyue Liu, and Lin Tan. Clocom: Mining existing source code for automatic comment generation. In *2015 IEEE 22nd International conference on software analysis, evolution, and reengineering (SANER)*, pp. 380–389. IEEE, 2015.

Hongqiu Wu, Hai Zhao, and Min Zhang. Code summarization with structure-induced transformer. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 1078–1090, 2021.

Guang Yang and Yujie Zhu. HGAdapter: Hypergraph-based adapters in language models for code summarization and clone detection. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 14823–14833. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.800. URL https://aclanthology.org/2025.findings-emnlp.800/.

Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pp. 783–794. IEEE / ACM, 2019. ISBN 978-1-7281-0869-8. doi: 10.1109/ICSE.2019.00086.

A Appendix

A.1 Geometry-Aware Relation Construction

For a snippet x_i, the preprocessor builds a shared node set V_i of AST internal nodes and code-token leaves. Each node stores a token id, a code-leaf mask, a copy-source map, and an AST depth. The eight directed relations are

$$\begin{aligned} \mathcal{E}_{\text{ast}} &= \mathcal{E}_{\text{base_child}} \cup \mathcal{E}_{\text{base_father}}, \quad & \mathcal{E}_{\text{sib}} &= \mathcal{E}_{\text{sibling_next}} \cup \mathcal{E}_{\text{sibling_prev}}, \\ \mathcal{E}_{\text{dfg}} &= \mathcal{E}_{\text{dfg_next}} \cup \mathcal{E}_{\text{dfg_prev}}, & \mathcal{E}_{\text{seq}} &= \mathcal{E}_{\text{code_next}} \cup \mathcal{E}_{\text{code_prev}} \end{aligned} \tag{13}$$

Taxonomy labels travel with the batch as a separate carrier and are not edges on V_i.

A.2 Poincaré Syntax Encoder

On-ball vectors are projected into the open ball. The encoder places z_v with the soft radial map in the main text and does not use the exponential map. The origin logarithm and the hyperbolic distance are

$$\log_0^c(z) = \operatorname{artanh}(\sqrt{c}\|z\|_2)\frac{z}{\sqrt{c}\|z\|_2} \tag{14}$$

$$d_c(x,y) = \frac{1}{\sqrt{c}}\operatorname{arcosh}\left(1 + \frac{2c\|x-y\|_2^2}{(1-c\|x\|_2^2)(1-c\|y\|_2^2)}\right) \tag{15}$$

Let $\hat{h}_v^{(0)} = W_{\text{in}} e_v/\sqrt{p}$ and $t_v^{(\ell)} = \log_0^c(z_v^{(\ell)})$. Parent- and child-directed means are averaged and updated as

$$\hat{h}_v^{(\ell+1)} = \frac{1}{\sqrt{p}}\,\text{LN}\Big(W_{\text{self}}^{(\ell)} t_v^{(\ell)} + \text{Dropout}(\text{ReLU}(W_{\text{msg}}^{(\ell)} m_v^{(\ell)}))\Big) \tag{16}$$

The same radial map then recalibrates $z_v^{(\ell+1)}$. The main-text gate is implemented as four $p \to 1$ maps

$$g_v^{\text{ast}} = \sigma\big(w_x^\top e_v + w_h^\top h_{\text{ast},v} + w_\Delta^\top |e_v - h_{\text{ast},v}| + w_\odot^\top (e_v \odot h_{\text{ast},v}) + b_g\big) \tag{17}$$

The three weak regularizers are

$$\mathcal{L}_{\text{pc}} = \frac{1}{|\mathcal{E}_{\text{ast}}^+|} \sum_{(u,v)\in\mathcal{E}_{\text{ast}}^+} \big[d_c(z_u,z_v) - |d_c(0,z_u) - d_c(0,z_v)| - m_{\text{pc}}\big]_+ \tag{18}$$

$$\mathcal{L}_{\text{rad}} = \frac{1}{|\mathcal{E}_\downarrow|} \sum_{(u,v)\in\mathcal{E}_\downarrow} \big[d_c(0,z_u) + m_{\text{rad}} - d_c(0,z_v)\big]_+ \quad \mathcal{E}_\downarrow = \{(u,v) : d_v > d_u\} \tag{19}$$

$$\mathcal{L}_{\text{tri}} = \frac{1}{|\mathcal{T}|} \sum_{(a,p,n)\in\mathcal{T}} \big[m_{\text{tri}} + \cos(z_a,z_n) - \cos(z_a,z_p)\big]_+ \tag{20}$$

A.3 Spectral Taxonomy Graph and Encoder

Summary categories are $\mathcal{C} = \{C_1,\dots,C_6\}$. Code and line comments share $\mathcal{F} = \{F_1,\dots,F_{10},F_{99}\}$, so STG has 17 category nodes. With $\mathcal{P}_i = \{(r_i, f_i^{\text{code}}), (r_i, f_i^{\text{lineC}}), (f_i^{\text{code}}, f_i^{\text{lineC}})\}$ and self-loop weight η,

$$A = \eta I + \sum_i \sum_{(a,b)\in\mathcal{P}_i} (e_a e_b^\top + e_b e_a^\top) \tag{21}$$

After eigendecomposition, each row of U_K is ℓ_2-normalized before the frequency gate θ. Defaults are $\eta = 1$ and $K = 8$. The node-level gate in the main text is applied after this encoder, so STE has one corpus-level coordinate and one instance-level injection.

A.4 Entanglement-Guided Adaptive Fusion

The eight sample states are $\mathcal{S}_i = \{p_i^{\text{graph}}, p_i^{\text{code}}, p_i^{\text{tax}}, p_i^{\text{ast}}, p_i^{\text{base}}, p_i^{\text{sib}}, p_i^{\text{dfg}}, p_i^{\text{seq}}\}$. Graph and AST states are means over V_i. The code state is the mean over code leaves. A relation family is the mean of endpoint averages on its edges. An empty edge set returns a zero vector.

$$\mathcal{L}_{\text{ent}} = \frac{1}{N} \sum_{i=1}^{N} (\mathcal{H}_i - \tau)^2 \tag{22}$$

A.5 Early-stop Decoder

The decoder has L layers and candidate set $\mathcal{D}$. Training runs all L layers and applies the copy generator on $\mathcal{D} \setminus \{L\}$. At inference step t, the batch shares one exit layer. Later blocks of that step are skipped, the sequence continues, and $P_t = \text{CopyGen}(H_t^{(e_t)}, \tilde{H}_{\text{code}}, \text{src_map})$. There is no per-example exit inside a batch and no cross-step KV cache. Defaults are $L = 8$, $\mathcal{D} = \{2, 4, 6\}$, $\ell_{\min} = 2$, and $\delta = 0.85$. Deep supervision uses

$$\mathcal{L}_{\text{deep}} = \frac{1}{|\mathcal{D}'|} \sum_{\ell \in \mathcal{D}'} \mathcal{L}_{\text{gen}}^{(\ell)} \tag{23}$$

A.6 The definition of three-view spectral taxonomy used in STP

Table 3: Definitions of spectral taxonomy.

View	Category	Definition
Reference Summary	C1 Functional Semantics	Describes the functional purpose, behavior, implementation mechanism, or design rationale of the code.
	C2 Interface and Contract	Describes parameters, return values, public interfaces, usage, pre-/post-conditions, or exceptions.
	C3 Maintenance and Evolution	Describes TODOs, development status, deprecation and migration, incomplete implementations, or temporary code.
	C4 Metadata and References	Describes authorship, licenses, versions, external links, issues, or related resources.
	C5 Tooling and Formatting	Contains compiler or IDE directives, formatting markers, visual separators, or auto-generated content.
	C6 Noise or Unknown	Contains no reliable semantic information or cannot be assigned to any of the preceding categories.
Source Code / Line-level Comments	F1 Data Computation and Transformation	Performs in-memory computation, text processing, parsing, transformation, filtering, matching, sorting, or general validation.
	F2 In-Memory Object and State Management	Creates, updates, copies, resets, caches, or removes objects and states within the current process.
	F3 Persistent Storage and Local I/O	Reads or modifies files, directories, streams, databases, persistent caches, archives, or serialized objects.
	F4 Communication and Distributed Interaction	Performs remote requests, service handling, socket communication, message exchange, session management, or external service integration.
	F5 System, Runtime, and Resource Control	Manages processes, threads, synchronization, resources, configuration, scheduling, logging, operating systems, or devices.
	F6 User Interface and Presentation	Handles user interaction, rendering, layout, visualization, navigation, reporting, or display formatting.
	F7 Security and Trustworthiness	Performs authentication, authorization, cryptographic operations, credential management, security sanitization, or integrity checking.
	F8 Modeling and Intelligent Analysis	Performs model data preparation, training, inference, evaluation, model selection, or parameter tuning.
	F9 Testing and Development Support	Supports testing, assertions, fixtures, mocks, debugging, building, packaging, code generation, static analysis, or refactoring.
	F10 Workflow and Integration Orchestration	Coordinates multiple functional steps through pipelines, event dispatching, lifecycle management, recovery, adaptation, or delegation.
	F99 Undetermined or No Functional Semantics	Contains insufficient functional evidence or does not express a distinguishable executable purpose.

A.7 Training objective

The full loss is

$$\mathcal{L} = \mathcal{L}_{\text{gen}} + \lambda_{\text{pc}}\mathcal{L}_{\text{pc}} + \lambda_{\text{rad}}\mathcal{L}_{\text{rad}} + \lambda_{\text{tri}}\mathcal{L}_{\text{tri}} + \lambda_{\text{ent}}\mathcal{L}_{\text{ent}} + \lambda_{\text{deep}}\mathcal{L}_{\text{deep}} \tag{24}$$

where $\mathcal{L}_{\text{gen}}$ is the copy-generator cross entropy. Default weights are $\lambda_{\text{pc}} = 0.01$, $\lambda_{\text{rad}} = 0.005$, $\lambda_{\text{tri}} = 0.005$, $\lambda_{\text{ent}} = 5 \times 10^{-4}$, and $\lambda_{\text{deep}} = 0.05$.

A.8 Details on Baselines

We compare GeoFusion with representative code summarization approaches covering sequence-based models, structure-aware models, pre-trained code models, and large language models.

CodeNN Iyer et al. (2016) is an attention-based encoder–decoder model that generates natural-language summaries directly from sequential source-code representations.

ASTNN Zhang et al. (2019) represents programs by decomposing ASTs into statement-level subtrees and sequentially encoding their structural representations.

Code2Seq Alon et al. (2018) models a code snippet as a collection of AST paths and applies attention to aggregate structure-relevant paths for summary generation.

CoCoAST Shi et al. (2023) hierarchically decomposes and reconstructs ASTs to integrate lexical, syntactic, and structural information into code representations.

Hetcos Guo et al. (2024) constructs a heterogeneous code graph that jointly incorporates AST, data-flow, sibling, and sequential relations for code summarization.

HierarchyNet Huynh Nguyen et al. (2024) learns hierarchical heterogeneous code representations by jointly modeling lexical, syntactic, and semantic information with Transformer-, tree-, and graph-based encoders.

CodeT5 Wang et al. (2021) is an encoder–decoder pre-trained code model that incorporates code-specific objectives, including identifier-aware learning, to support code understanding and generation.

CodeBERT Feng et al. (2020) is a bimodal pre-trained Transformer that learns joint representations from large-scale programming-language and natural-language corpora.

UniXCoder Guo et al. (2022) is a unified cross-modal pre-trained model that supports code understanding and generation through shared representations of source code and natural language.

GraphCodeBERT Guo et al. (2020) extends code pre-training with explicit data-flow structures to capture semantic dependencies beyond lexical token sequences.

ChatGPT OpenAI (2023) (ChatGPT-4o) represents general-purpose large language models capable of performing code summarization through instruction-based generation without task-specific structural modeling.

A.9 Details Color Mapping Used in t-SNE Visualization

Table 4: Color mapping used in t-SNE Visualization.

Taxonomy	Category	Color
Reference Summary	C1 Functional Semantics	
	C2 Interface and Contract	
	C3 Maintenance and Evolution	
	C4 Metadata and References	
	C5 Tools and Formatting	
	C6 Noise or Unknown	
Source Code / Line-level Comments	F1 Data Computation and Transformation	
	F2 In-memory Object and State Management	
	F3 Persistent Storage and Local I/O	
	F4 Communication and Distributed Interaction	
	F5 System, Runtime and Resource Control	
	F6 User Interface and Presentation	
	F7 Security and Trust	
	F8 Model-based Analytics	
	F9 Testing and Development Support	
	F10 Workflow and Integration Orchestration	
	F99 Unclear or Non-functional	