COPS: FIRST-PRINCIPLES LONG-CONTEXT CODE COMPRESSION VIA CONSTRUCTING CONTEXT OPTIMAL PRESERVATION SETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Code large language models incur substantial computational costs and suffer from interference caused by irrelevant context during long-context reasoning. Existing code compression methods typically reduce the problem to a static ranking task based on a single context–query relevance criterion, rather than deriving the compression objective from first principles as a dynamic subset-selection problem. Consequently, they suffer from severe performance degradation when the token budget is aggressively reduced, while still retaining considerable information redundancy that interferes with downstream tasks. In this paper, we revisit long-context code compression from an information-theoretic perspective and formulate it as constructing *Context Optimal Preservation Sets*. Through a top-down decomposition of the mutual-information objective, we identify three fundamental principles that govern effective context selection: task relevance, dependency coverage, and information gain. Based on these principles, we propose **COPS**, a training-free, plug-and-play framework that instantiates all criteria with Approximated Mutual Information (AMI) and constructs preservation sets through a principled coarse-to-fine pipeline: (i) AMI-based recall followed by greedy preservation-set construction over candidate contexts; (ii) an information-saturation trigger that dynamically terminates context expansion via a one-sided concentration bound; and (iii) perplexity-anchored semantic block partitioning with positive-advantage iterative pruning for intra-function refinement. Extensive experiments on long-context benchmarks for code completion, summarization, and question answering demonstrate that COPS consistently establishes a superior Pareto frontier between compression ratio and downstream performance, achieving up to $6.8\times$ compression with negligible degradation while significantly reducing inference latency and memory consumption.

1 INTRODUCTION

Code large language models (LLMs) have substantially advanced code completion (Guo et al., 2023; Lu et al., 2022), code question answering (Strich et al., 2024; Chen et al., 2025a), and program repair (Chen et al., 2025b; Jin et al., 2023), reshaping modern software development. Yet repository-level use of widely adopted models such as DeepSeek-Coder (Guo et al., 2024), Qwen2.5-Coder (Hui et al., 2024), and Seed-Coder (Seed et al., 2025) increasingly requires reasoning over contexts spanning tens of thousands of tokens, making efficient long-context processing a critical bottleneck.

Source code is not merely text: its syntax and program structure induce nonlocal dependencies across definitions, calls, and files (Ding et al., 2023; Peng et al., 2026; Li et al., 2025b). Most existing compression methods do not adequately preserve this structure. General-purpose prompt compressors, such as LLMLingua (Jiang et al., 2023) and Selective Context (Li et al., 2023b), overlook code-specific organization, while retrieval-augmented generation (RAG) (Cheng et al., 2024) prioritizes embedding similarity and can miss latent dependencies. Code-oriented methods, including DietCode (Zhang et al., 2022) and SlimCode (Wang et al., 2024), exploit program structure but primarily prune short-context units. LongCodeZip (Shi et al., 2025) improves long-context compression, yet still retains a fundamental limitation shared by the prevailing paradigm.

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.

Specifically, existing methods cast compression as a *static ranking problem*: each fragment receives a fixed score—typically based on context–query similarity or conditional perplexity—and the highest-scoring fragments are retained under a preset budget. This formulation has two intrinsic weaknesses. First, fragmentwise scores ignore set-level interactions: individually relevant fragments may be mutually redundant, whereas a fragment of limited standalone relevance may provide essential dependency scaffolding. Second, a fixed budget is insensitive to diminishing marginal utility, causing useful evidence to be discarded under tight budgets and redundant or even distracting context to be retained under loose ones. Effective compression should therefore dynamically determine both *which* fragments to preserve and *when* additional context ceases to be informative.

We address this problem from first principles by formulating long-context code compression as the construction of *Context Optimal Preservation Sets*: subsets that maximizes its mutual information with the original context and the task query under a token budget. A top-down decomposition of this objective yields three requirements for effective selection: **task relevance**, **dependency coverage**, and **information gain**. These respectively capture direct utility for the query, preservation of dependencies in the full context, and the nonredundant marginal contribution of a candidate relative to the current set. We instantiate all three with Approximated Mutual Information (AMI), a computable proxy based on the perplexity reduction that one fragment induces for another fragment or the query, thereby grounding the pipeline in a unified information measure.

Based on this formulation, we propose **COPS**, a training-free, plug-and-play, and model-agnostic framework with a coarse-to-fine architecture. At the coarse scale, COPS performs AMI-based recall and greedily constructs a preservation set by jointly optimizing relevance, coverage, and information gain. It then applies an information-saturation trigger calibrated by Cantelli's one-sided concentration inequality to stop expansion when further context is unlikely to be useful. Finally, COPS partitions each retained fragment into perplexity-anchored semantic blocks and iteratively removes blocks with non-positive advantage, increasing information density without introducing an additional pruning hyperparameter.

We evaluate COPS on Long Code Completion (Guo et al., 2023), Long Module Summarization (Bogomolov et al., 2024), RepoQA (Liu et al., 2024), and Long Code QA (Rando et al., 2025). Across tasks and backbone models, COPS establishes a consistently stronger Pareto frontier between compression and downstream performance, and it transfers to lightweight 0.5B- and 1.5B-parameter compressors. COPS achieves up to $6.8\times$ compression with negligible performance degradation while substantially reducing inference latency and memory consumption. In summary, the contributions of this paper are as follows:

- We formulate long-context code compression as constructing Context Optimal Preservation Sets and derive task relevance, dependency coverage, and information gain as three complementary selection principles, all uniformly instantiated with AMI.
- We introduce COPS, a training-free hierarchical framework that couples greedy preservation-set construction with a statistically grounded saturation trigger and hyperparameter-free semantic-block refinement, jointly deciding what to retain and when to stop.
- We conduct comprehensive experiments on code completion, summarization, and question answering. COPS consistently outperforms state-of-the-art baselines, uses up to 15% fewer compressed tokens than the strongest baseline, and reaches up to $6.8\times$ compression with negligible performance degradation.

2 Related Work

Large Language Models for Code. General-purpose LLMs, such as GPT (Achiam et al., 2023), Qwen (Bai et al., 2023), and DeepSeek (Guo et al., 2025), have demonstrated strong code understanding and generation abilities through large-scale pretraining on mixed text and code data. Meanwhile, specialized code LLMs, including CodeLlama (Roziere et al., 2023), StarCoder (Li et al., 2023a), DeepSeek-Coder (Guo et al., 2024), and Qwen2.5-Coder (Hui et al., 2024), further enhance code reasoning and generation through code-specific optimization (Nijkamp et al., 2022; Wang et al., 2023). However, these models still struggle with long-context code scenarios due to context forgetting and disrupted cross-module dependencies. Existing efforts, including long-code

benchmarks (Rando et al., 2025; Bogomolov et al., 2024; Yan et al., 2024) and long-context optimized models (Guo et al., 2023; Li et al., 2025a), mainly focus on improving model architectures and training strategies.

Prompt Compression. Prompt compression has been widely explored in natural language and code domains. General text compression methods, such as LLMLingua (Jiang et al., 2023), LongLLMLingua (Jiang et al., 2024), Selective Context (Li et al., 2023b), often overlook code-specific structures, while code-oriented methods, including DietCode (Zhang et al., 2022), SlimCode (Wang et al., 2024), and LongCodeZip (Shi et al., 2025), better preserve code semantics but mainly rely on static or query-aware compression.

3 Motivation: A First-Principles Formulation

Problem Setup. Given a long code context list $C = \{c_1, \ldots, c_n\}$ containing n context fragments and a task instruction Q, where each fragment c_i consists of multiple tokens, the goal of context compression is to generate a compressed context $C' \subseteq C$ satisfying $|C'| \leq B$, where B denotes the token budget. The objective is to maximize downstream task performance under the budget constraint. The key challenge is to maximize the *information density* of the compressed context for the given query while reducing irrelevant and redundant information.

3.1 First-Principles Compression Objective

Rather than following the conventional static relevance scoring-and-ranking paradigm, we rethink the problem from a first-principles perspective. Given the full context set C and the query Q, the goal of compression is to retain a subset $S \subseteq C$ such that reasoning based on (S, Q) remains consistent with that based on (C, Q). We formalize this as an information-theoretic objective:

$$S^* = \arg \max_{S \subseteq C,\ |S| \leq B} I(S;\, C, Q), \tag{1}$$

where $I(\cdot;\, \cdot)$ denotes mutual information. This defines the first principle of code context compression: the optimal subset is the one that maximally preserves information about both the original context C and the query Q.

3.2 Decomposition of the First-Principles Objective

To examine the internal structure of the mutual information objective, we decompose it via the chain rule:

$$I(S;\, C, Q) = \underbrace{I(S;\, Q)}_{\text{task relevance}} + \underbrace{I(S;\, C \mid Q)}_{\text{dependency coverage}} = I(S;\, Q) + H(C \mid Q) - H(C \mid S, Q). \tag{2}$$

These two components capture the essential criteria for optimal subset construction. **Task relevance** $I(S; Q)$ quantifies the direct utility of selected fragments for the query. **Dependency coverage** $I(S; C \mid Q)$ ensures S retains critical context from C. By suppressing $H(C \mid S, Q)$, it safeguards the dependencies the query inherently relies on. Let the accumulated set of selected items at the k-th iteration be $S_k = \{c_{(1)}, \ldots, c_{(k)}\}$. We can then express the overall mutual information as an accumulation of step-wise marginal gains:

$$I(S_k;\, C, Q) = \sum_{i=1}^{k} I\big(c_{(i)};\, C, Q \mid S_{i-1}\big). \tag{3}$$

Thus, a fragment's utility is conditioned on the existing selection. Formally, the marginal gain of incorporating $c_{(i)}$ into S_{i-1} is quantified by the conditional mutual information:

$$\Delta_i = I(S_i;\, C, Q) - I(S_{i-1};\, C, Q) = I\big(c_{(i)};\, C, Q \mid S_{i-1}\big). \tag{4}$$

Additionally, this incremental value is inherently capped by the candidate's conditional entropy:

$$0 \leq I\big(c_{(i)};\, C, Q \mid S_{i-1}\big) \leq H\big(c_{(i)} \mid S_{i-1}\big) \leq H\big(c_{(i)}\big). \tag{5}$$

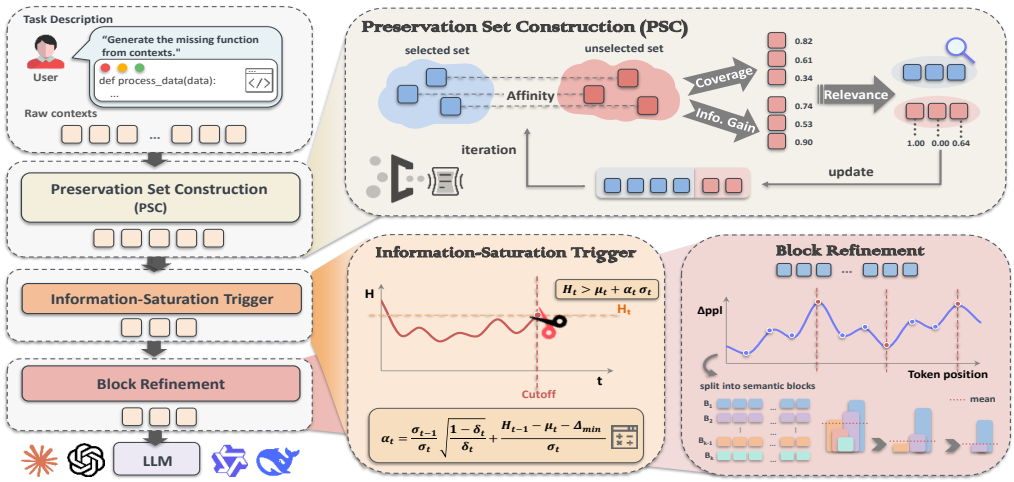

Figure 1: Overview of COPS. COPS constructs the Context Optimal Preservation Sets through a principled coarse-to-fine pipeline.

When $c_{(i)}$ introduces novel information absent from the current set, this conditional entropy upper bound is tight; it weakens as the fragment grows redundant with previously retained fragments. This implies that under a fixed token budget, mutual information maximization prioritizes candidates semantically distinct from existing selections, making **information gain** a prerequisite for constructing efficient, non-redundant subsets.

4 METHODS

In this section, we present COPS, a training-free, plug-and-play, and model-agnostic framework that operationalizes the first-principles formulation of Section 3. The overall architecture is illustrated in Figure 1. COPS follows a principled coarse-to-fine pipeline: (i) a coarse-grained stage that recalls candidates by AMI-based relevance and re-scores them via greedy preservation-set construction; (ii) an information-saturation trigger that dynamically decides when to stop expanding the retained set; and (iii) a fine-grained stage that refines each retained function at the semantic-block level.

4.1 APPROXIMATED MUTUAL INFORMATION AS A UNIFIED MEASURE

All three principles identified in Section 3 require estimating mutual information involving code fragments, which is intractable in its exact form. Following the information-theoretic view of perplexity, we adopt Approximated Mutual Information (AMI) as a unified, computable proxy. Given an instruction $Q = \{q_1, \ldots, q_N\}$, its unconditional and conditional perplexities under the model are

$$\mathrm{PPL}(Q) = \exp\Big(-\frac{1}{N}\sum_{i=1}^{N}\log P(q_i \mid q_{<i})\Big), \qquad \mathrm{PPL}(Q \mid C) = \exp\Big(-\frac{1}{N}\sum_{i=1}^{N}\log P(q_i \mid q_{<i}, C)\Big). \tag{6}$$

The AMI of a context fragment C with respect to the query Q is defined as the perplexity reduction after introducing C:

$$\mathrm{AMI}(C, Q) = \mathrm{PPL}(Q) - \mathrm{PPL}(Q \mid C). \tag{7}$$

Since perplexity is an exponential transformation of conditional cross-entropy, $\mathrm{AMI}(C, Q)$ is a monotonic transformation of the conditional mutual information $I(C; Q) = H(Q) - H(Q \mid C)$. A higher AMI indicates that the fragment substantially reduces the uncertainty of generating the query, capturing deep semantic and dependency relations rather than surface-level similarity. Analogously, we define the *inter-fragment affinity* $a(c_i \to c_j) = \mathrm{AMI}(c_i, c_j) = \mathrm{PPL}(c_j) - \mathrm{PPL}(c_j \mid c_i)$, which quantifies how much information fragment c_i provides about fragment c_j.

4.2 COARSE-GRAINED STAGE: PRESERVATION-SET CONSTRUCTION

AMI Recall. Computing the full preservation-set objective over all n fragments is unnecessary, since most fragments are irrelevant to the query. We therefore first compute $\mathrm{AMI}(c_i, Q)$ for every candidate and retain the top $p'\%$ fragments as the recalled set C_{recall}, which preserves high-relevance candidates at negligible cost.

Greedy Construction of the Preservation Set. For each candidate fragment c_i in the recalled set, given the currently selected set S, we denote the remaining fragments as $U = C_{\mathrm{recall}} \setminus S$ and instantiate the three principles with AMI. The *task relevance* score is $r_i = \mathrm{AMI}(c_i, Q)$. The *dependency coverage* score quantifies the novel information c_i provides about as-yet-uncovered fragments beyond the current set's baseline coverage:

$$\mathrm{cov}_i(U) = \sum_{c_j \in U} \max\Big(0,\ a(c_i \to c_j) - \max_{c_k \in S} a(c_j \to c_k)\Big). \tag{8}$$

Following min–max normalization, denoted $\tilde{r}_i$ and $\tilde{\mathrm{cov}}_i(U)$ respectively, the two scores are aggregated into an information preservation score:

$$g_i^{\mathrm{info}} = \tilde{r}_i + \lambda\, \tilde{\mathrm{cov}}_i(U). \tag{9}$$

Following the information gain principle derived from Equation (5), we further incorporate a *information gain* score $d_i(S) = \max\big(0,\ a(S \cup \{c_i\} \to Q) - a(S \to Q)\big)$, where $\tilde{d}_i(S)$ is the min–max-normalized $d_i(S)$. The preservation set is then expanded greedily:

$$S_{t+1} = S_t \cup \Big\{ \arg \max_{c_i \in C_{\mathrm{recall}} \setminus S_t} \big[g_i^{\mathrm{info}} + \alpha\, \tilde{d}_i(S_t) \big] \Big\}, \tag{10}$$

where α and λ are balance factors. The construction is initialized with the fragment of highest relevance, $S_1 = \{\arg\max_i r_i\}$, and expanded until the token budget yields the top-k candidates c_{coar}. Each candidate's final *coarse score* is its greedy selection value $g_i^{\mathrm{info}} + \alpha\, \tilde{d}_i$ at the time of selection, which determines the admission order in the next stage.

4.3 INFORMATION-SATURATION TRIGGER: DECIDING WHEN TO STOP

The coarse-grained stage produces an ordered candidate list c_{coar} of size k, but committing to all k fragments would ignore the diminishing marginal information gain revealed by Equation (4). We therefore introduce an adaptive re-selection mechanism that monitors the global system perplexity as fragments are iteratively admitted, and terminates expansion at the information-saturation point.

Specifically, we maintain an index set V and its corresponding global perplexity sequence $\mathcal{P}$. Initially, V contains the indices of the first v_0 fragments (we set $v_0 = 2$), and $\mathcal{P}$ records their perplexity values. At the t-th iteration, we add the t-th fragment into V, compute the system perplexity $H_t = \mathrm{PPL}(Q \mid V)$ under the current context–query combination, and append it to $\mathcal{P}$. Meanwhile, we compute the adaptive threshold τ_t from the statistics of all perplexity values in $\mathcal{P}$:

$$\alpha_t = \max\Big(0, \frac{\sigma_{t-1}}{\sigma_t}\sqrt{\frac{1-\delta_t}{\delta_t}} + \frac{H_{t-1} - \mu_t - \Delta_{\min}}{\sigma_t}\Big), \qquad \tau_t = \mu_t + \alpha_t\, \sigma_t, \tag{11}$$

Here, $\delta_t \in (0, 1)$ denotes the per-step false-truncation risk budget, and $\sqrt{(1-\delta_t)/\delta_t}$ is the confidence margin derived from Cantelli's one-sided inequality. A smaller δ_t produces a larger confidence margin, making the trigger more conservative and reducing the probability of prematurely discarding a useful context fragment. The variance ratio σ_{t-1}/σ_t implements a heteroscedasticity correction that rescales the canonical Cantelli confidence margin to account for volatility in the perplexity sequence. The quantities μ_t and σ_t denote the mean and standard deviation of the perplexity sequence at the t-th expansion step. $\Delta_{\min}$ represents the minimum practically useful perplexity reduction expected from an additional context fragment. See detailed theoretical analysis in Appendix A.

If $H_t > \tau_t$, the newly added fragment is regarded as redundant, triggering the discard mechanism that terminates the iteration. Otherwise, the fragment is retained until $|V| = k$. The final selected m fragments ($m \le k$) replace c_{coar} as inputs for fine-grained compression.

4.4 Fine-Grained Stage: Semantic Block Refinement

In the fine-grained stage, we perform a block-level decomposition of each retained function. We identify internal anchors within functions based on perplexity and split each function into several basic semantic blocks, where each block represents a relatively independent semantic unit. Assume a retained function is divided into b semantic blocks, denoted as $\{B_j\}_{j=1}^{b}$. We calculate the AMI score between each block and the query Q as its importance score $R_j = \mathrm{AMI}(B_j, Q)$. We design a hyperparameter-free iterative selection mechanism. Blocks with scores higher than the mean score of the current candidate set have positive contributions and should be preferentially retained, while the remaining blocks are considered to have negative contributions and tend to be discarded.

Let L denote the index set of current candidate semantic blocks (initialized as $L = \{1, \ldots, b\}$), with corresponding scores $\{R_j\}_{j \in L}$. Each iteration consists of the following steps: **Step 1**: calculate the mean score of the current candidate set, $\bar{R} = \frac{1}{|L|} \sum_{j \in L} R_j$; **Step 2**: for each $j \in L$, calculate its positive advantage relative to the current mean, $R'_j = \max(R_j - \bar{R}, 0)$; **Step 3**: construct a new positive-advantage index set $S = \{j \in L \mid R'_j > 0\}$. If $S = L$, all current candidate blocks already have positive advantages, and the iteration terminates. Otherwise, update $L \leftarrow S$ and $\{R_j\}_{j \in L} \leftarrow \{R'_j\}_{j \in L}$, then return to Step 1.

After termination, we denote the final stable positive-advantage index set as S^*. The retained semantic block set is defined as $B^* = \{B_j \mid j \in S^*\}$, with the number of retained blocks $b' = |S^*|$. Since each iteration strictly shrinks the candidate set or terminates, the procedure converges in at most b rounds. These semantic blocks are used as the final compressed context input for downstream tasks.

5 Experiments

5.1 Settings

Baselines and Models. To comprehensively evaluate our method's effectiveness, we compare COPS against the following baselines: (1) **No Compression**: the full code context; (2) **No Context**: only the task instruction; (3) **Retrieval-Based Methods**: RAG employing the UniXCoder-base embedding model (Guo et al., 2022), CodeRAG (Zhang et al., 2025) and AIRCoder (Shi et al., 2026); (4) **Text Compression Methods**: Selective-Context (Li et al., 2023b) and LongLLMLingua (Jiang et al., 2024); (5) **Code Compression Methods**: DietCode (Zhang et al., 2022) and LongCodeZip (Shi et al., 2025). All approaches are tested on multiple LLMs, including DeepSeek-Coder-6.7B and Qwen2.5-Coder-7B. We adopt their instruction-tuned variants from Hugging Face and further extend evaluation to leading closed-source models, namely GPT-4o (Hurst et al., 2024) and Claude-3.7-Sonnet (Anthropic, 2025). Further details are provided in Appendix B.

Benchmarks. We evaluate COPS on long-code context benchmarks spanning three representative tasks: code completion, code summarization, and code question answering. For long code completion, we adopt Long Code Completion (Guo et al., 2023). For summarization, we employ Long Module Summarization (Bogomolov et al., 2024). RepoQA (Liu et al., 2024) serves as a multilingual QA benchmark encompassing 600 tests across 60 repositories and 6 programming languages. Long Code QA (Rando et al., 2025) targets QA over ultra-long code contexts supporting up to 1M tokens, designed to examine method scalability under extreme context lengths.See Appendix D.

Evaluation Metrics. Consistent with prior studies (Shi et al., 2025; Ma et al., 2025), we report the compression ratio and ACU (Average Capability per Unit of Computation) across all tasks. ACU captures post-compression information density. To simplify the calculation of ACU, the original context length is normalized to 1k tokens. Here, #Tokens denotes the token count of the compressed context, and #Performance indicates the task-specific evaluation score:

$$\mathrm{Ratio} = \frac{\|C_{\mathrm{original}}\|}{\|C_{\mathrm{compressed}}\|}, \qquad \mathrm{ACU} = \frac{\#\mathrm{Performance}}{\#\mathrm{Tokens}}. \tag{12}$$

For code completion, we employ Exact Match (EM) and Edit Similarity (ES) (Guo et al., 2023) to assess performance. In code summarization, following Bogomolov et al. (2024), we adopt GPT-4o-mini (Hurst et al., 2024) as a judge to compare the generated and reference summaries, yielding

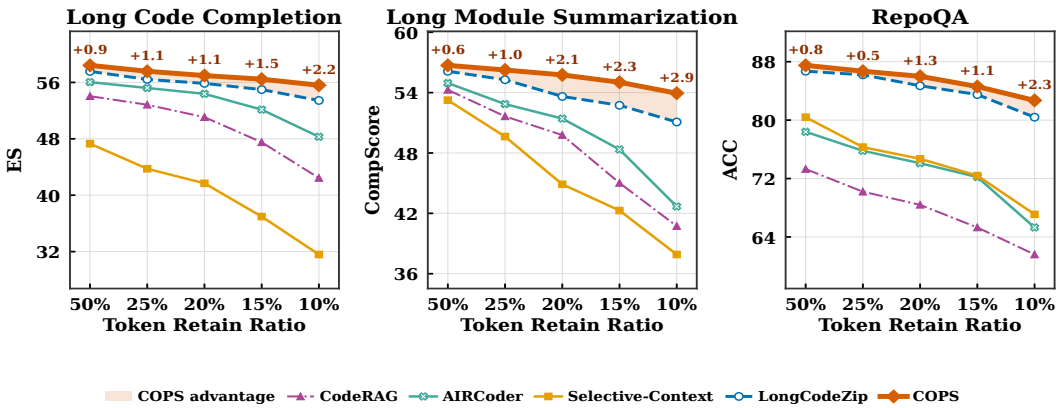

Figure 2: Performance vs. token retain ratio on three benchmarks. The shaded band and '+' labels mark COPS's margin over the strongest baseline LongCodeZip at every ratio.

Table 1: Performance comparison across different models and methods on three benchmarks.

Models	Methods	Long Code Completion				Long Module Summarization			RepoQA		
		ES	EM	Ratio	ACU	CompScore	Ratio	ACU	Avg Acc	Ratio	ACU
Qwen2.5-Coder-7B	No Compression	56.36	31.80	1.0×	0.09	56.00	1.0×	0.05	86.0	1.0×	0.08
	No Context	38.14	9.60	–	–	6.13	–	–	0.0	–	–
	RAG	50.81	24.60	2.8×	0.21	53.50	1.7×	0.09	67.5	3.7×	0.24
	CodeRAG	53.87	29.20	3.1×	0.25	54.27	2.0×	0.11	70.2	3.9×	0.27
	AIRCoder	55.39	29.80	3.2×	0.27	54.58	2.2×	0.12	75.8	4.0×	0.30
	LongLLMLingua	23.88	9.00	3.2×	0.10	46.72	1.5×	0.07	71.3	4.3×	0.30
	Selective-Context	45.37	16.80	2.6×	0.16	53.29	1.9×	0.10	75.7	4.6×	0.34
	DietCode	43.91	13.20	3.4×	0.19	35.67	2.1×	0.07	26.0	3.7×	0.09
	LongCodeZip	57.28	32.00	3.7×	0.32	56.19	1.7×	0.09	86.2	4.1×	0.35
	COPS (Ours)	**56.94**	**31.40**	**6.4×**	**0.55**	**56.26**	**3.8×**	**0.21**	**85.5**	**5.4×**	**0.45**
DeepSeek-Coder-6.7B	No Compression	57.14	34.40	1.0×	0.09	19.09	1.0×	0.02	38.3	1.0×	0.04
	No Context	41.29	13.20	–	–	2.49	–	–	0.0	–	–
	RAG	58.48	31.60	4.2×	0.37	22.95	2.1×	0.05	55.5	3.5×	0.19
	CodeRAG	56.71	30.80	4.0×	0.34	23.21	2.2×	0.05	57.2	3.8×	0.21
	AIRCoder	59.07	32.80	4.3×	0.39	24.79	2.4×	0.06	68.7	3.9×	0.26
	LongLLMLingua	54.09	26.40	4.8×	0.38	21.62	1.7×	0.04	59.1	3.0×	0.17
	Selective-Context	54.24	26.20	4.1×	0.32	21.73	2.4×	0.05	58.3	3.9×	0.22
	DietCode	51.57	20.20	3.4×	0.24	17.35	2.1×	0.04	20.5	3.7×	0.07
	LongCodeZip	60.41	35.00	4.3×	0.40	27.89	2.3×	0.06	74.6	4.5×	0.33
	COPS (Ours)	**60.16**	**34.20**	**6.8×**	**0.63**	**27.65**	**4.2×**	**0.11**	**74.2**	**5.9×**	**0.43**

the CompScore (See Appendix B). CompScore ranges from 0 to 100, where 50 indicates neutral preference. For RepoQA, we evaluate achieving a BLEU (Papineni et al., 2002) similarity rate above 0.8 (Liu et al., 2024). For Long Code QA, we adopt Accuracy as the metric.

5.2 Main Results

We evaluate COPS under two complementary settings: comparison of compression ratio and information density under fixed budget and comparison of performance under same compression ratio.

Fixed Budget, Unconstrained Compression. Table 1 reports results under a fixed token budget, where each method compresses as aggressively as possible, thereby probing the attainable Pareto frontier between compression ratio and downstream quality. COPS consistently pushes this frontier furthest across all tasks and both backbones. On Qwen2.5-Coder-7B, it reaches 6.4× compression on code completion with only a 0.34-point ES drop relative to LongCodeZip, while improving ACU by 72%. The margin widens under DeepSeek-Coder-6.7B, and it achieves 6.8× compression while attaining an ACU of 0.63. Gains are most pronounced on summarization, where COPS attains 3.8 and 4.2× compression—roughly doubling the ratio of the strongest baseline—at comparable CompScore, confirming that our first-principles successfully retains the structural scaffolding that single-criterion relevance scoring discards. On RepoQA, COPS reaches up to 5.9× compression with the nearly best overall accuracy. These results show that our preservation-set construction keeps

the most relevant fragments while demoting redundant, low-gain ones to later admission positions, and that the information-saturation trigger pinpoints the exact point where additional context ceases to be informative, discarding it thereafter.

Identical Retention Ratios. Figure 2 compares COPS with the strongest baselines at matched retention ratios. COPS outperforms at every ratio and task, with the advantage widening as the budget tightens (*e.g.*, +2.9 CompScore on Long Module Summarization at 10%). This trend is precisely what the information-gain principle predicts: when the budget is scarce, explicitly penalizing redundancy among selected fragments preserves more non-redundant evidence than static single-criterion scoring. Under loose budgets, even a suboptimal selection can retain sufficient evidence; under tight budgets, only a selection that respects first-principles level interactions survives.

5.3 Transferability Analysis

Table 3 decouples the compression model from the generation model on code completion. Two observations emerge. First, every compressed variant outperforms the no-compression baseline across all compressor–generator pairs (*e.g.*, 57.79 vs. 56.75 for DeepSeek-Coder-6.7B), confirming that long code contexts contain substantial distractive redundancy and raising information density is itself beneficial. Second, COPS transfers across model and model scales: a Qwen-1.5B compressor attains an average ES of 58.41, within 0.3 performance degradation of the 7B compressor (*i.e.*, 58.71). The selection criterion is model-agnostic by construction. Smaller compressors reduce latency and memory without sacrificing selection quality.

Table 2: Ablation study results on Long Code Completion and RepoQA using Qwen2.5-Coder-7B.

Methods	**Long Code Completion**			**RepoQA**	
	ES	EM	ACU	ACC	ACU
COPS (full)	**56.94**	**31.40**	**0.55**	**85.5**	**0.45**
Coarse-grained stage:					
w/ Relevance only	56.22	30.40	0.53	84.6	0.45
w/ Coverage only	55.84	30.20	0.52	84.1	0.44
w/ Information Gain only	56.07	30.40	0.53	84.3	0.44
w/o Relevance	56.17	30.40	0.53	84.4	0.45
w/o Coverage	56.41	30.60	0.54	85.1	0.45
w/o Information Gain	56.35	30.40	0.54	84.9	0.45
Saturation trigger:					
w/o Saturation trigger	56.39	31.80	0.41	85.8	0.37
Fine-grained stage:					
w/o Block Refinement	56.06	30.60	0.46	85.3	0.38
w/ Top-K (K=3)	56.12	30.20	0.47	84.5	0.42
w/ Top-K (K=5)	56.18	30.40	0.43	84.7	0.39
w/ Top-K (K=10)	56.57	30.40	0.40	85.2	0.37
w/ Top-K (K=20)	56.33	30.60	0.36	85.3	0.35

5.4 Ablation Study

To understand the contribution of each component in COPS, we conduct ablation studies on the long code completion and question answering task using Qwen2.5-Coder-7B, keeping the total token budget and other hyperparameters consistent with the full method. See Table 2.

Coarse-grained first principles. Relevance-only ranking wastes budget on mutually redundant fragments (*i.e.*, ACU drops to 0.53), while coverage-only and gain-only variants discard query-critical evidence, hurting ES and EM. Removing any one principle consistently underperforms the full model, confirming that task relevance, dependency coverage, and information gain are complementary rather than interchangeable—as the chain-rule decomposition predicts.

Information-saturation trigger. Disabling the trigger collapses the compression ratio from 6.4× to 4.8× and ACU from 0.55 to 0.41, without quality gain. This validates the argument of Section 4.3: a fixed budget is insensitive to the diminishing marginal utility, whereas the Cantelli-calibrated stopping rule adapts the effective budget to sample-specific information density.

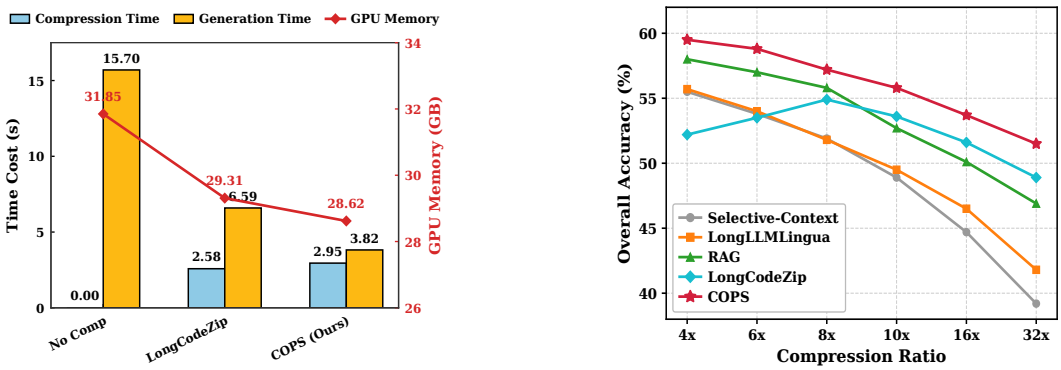

Figure 3: (a) Comparison of time cost and GPU memory consumption on long code completion. (b) Scaling performance across different compression ratios on Long Code QA.

Fine-grained block refinement. Removing block refinement lowers ES by 0.88 and ACU by 17%, and demonstrating that removing redundant segments within functions counterintuitively improves downstream task performance. Fixed top-K alternatives ($K = 3, 5, 10, 20$) uniformly underperform the positive-advantage iteration, and their ACU degrades monotonically as K increases (*i.e.*, from 0.47 to 0.36), since larger K reintroduces low-contribution blocks. The positive-advantage iteration adaptively prunes blocks without a manual cardinality hyperparameter.

Table 3: Cross-model transferability results (ES) on long code completion.

Comp Model	DS-6.7B	Qwen-7B	Avg. ES
No Compression	57.14	56.36	56.75
No Context	41.29	38.14	39.72
DS-6.7B	59.16	56.41	57.79
Qwen-0.5B	59.81	55.92	57.87
Qwen-1.5B	60.27	56.54	58.41
Qwen-3B	60.29	56.35	58.32
Qwen-7B	**60.47**	**56.94**	**58.71**

5.5 Efficiency and Scalability

Figure 3(a) reports wall-clock time and peak GPU memory on code completion using Qwen2.5-Coder-7B. First, we compute the AMI for each context fragment in parallel via a single forward pass, enabling the efficient recall and filtering of the vast majority of irrelevant contexts; greedy construction and the saturation trigger operate on precomputed scores and the top-k candidate contexts, where $k \ll n$, adding merely 2.95 s of compression overhead. End-to-end latency drops to 6.77 s (*i.e.*, 15.70 s without compression), and peak memory decreases from 31.85 GB to 28.62 GB.

Figure 3(b) conducts compression pressure test on Long Code QA with contexts up to 1M tokens. As the compression ratio increases, COPS consistently outperforms all baselines, with the performance gap continuing to widen; even at 32× compression, the accuracy retained by COPS remains comparable to that achieved by certain baselines at only 6×, demonstrating that the principled subset-selection formulation scales gracefully to extreme context lengths where other static-ranking approaches degrade sharply. Therefore, these results demonstrate the robustness of our method.

6 Conclusion

In this work, we rethinked long-context code compression from a first-principles perspective and formulated it as constructing Context Optimal Preservation Sets. We proposed COPS, a training-free, plug-and-play framework that constructs greedy preservation-set, an information-saturation trigger and positive-advantage semantic block refinement. Extensive experiments on code completion, summarization, and question answering demonstrate that COPS achieves up to 6.8× compression with minimal performance loss, This work establishes COPS as a critical criterion for efficient and effective long-context modeling in LLMs code intelligence.

AI USE STATEMENT

In this work, we used generative AI tools for refining and polishing the text of the manuscript, improving the overall clarity, precision, and readability of our methodology descriptions and mathematical derivations. We have not used generative AI tools for creating figures, or synthesizing data, and generating experimental results or formulating core theoretical proofs are not applicable to this work. Additionally, we used generative AI tools for generating a portion of code. We have reviewed all AI-assisted work. Specifically, we carefully checked all LLM-polished text to ensure it accurately reflects our original research and does not introduce any factual errors, and LLM-generated code was verified and tested for correctness by 2 authors. All core ideas, experimental designs, and results are the original work of the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Anthropic. Claude 3.7 Sonnet and Claude Code. `https://www.anthropic.com/news/claude-3-7-sonnet`, February 2025.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Egor Bogomolov, Aleksandra Eliseeva, Timur Galimzyanov, Evgeniy Glukhov, Anton Shapkin, Maria Tigina, Yaroslav Golubev, Alexander Kovrigin, Arie Van Deursen, Maliheh Izadi, et al. Long code arena: a set of benchmarks for long-context code models. *arXiv preprint arXiv:2406.11612*, 2024.

Jialiang Chen, Kaifa Zhao, Jie Liu, Chao Peng, Jierui Liu, Hang Zhu, Pengfei Gao, Ping Yang, and Shuiguang Deng. Coreqa: uncovering potentials of language models in code repository question answering. *arXiv preprint arXiv:2501.03447*, 2025a.

Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, et al. Swe-exp: Experience-driven software issue resolution. *arXiv preprint arXiv:2507.23361*, 2025b.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xrag: Extreme context compression for retrieval-augmented generation with one token. volume 37, pp. 109487–109516, 2024.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36:46701–46723, 2023.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 7212–7225, 2022.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pp. 12098–12107. PMLR, 2023.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 1658–1677, 2024.

Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering*, pp. 1646–1656, 2023.

Jia Li, Hao Zhu, Huanyu Liu, Xianjie Shi, He Zong, Yihong Dong, Kechi Zhang, Siyuan Jiang, Zhi Jin, and Ge Li. aixcoder-7b-v2: Training llms to fully utilize the long context in repository-level code completion. *arXiv preprint arXiv:2503.15301*, 2025a.

Jia Li, Hao Zhu, Huanyu Liu, Xianjie Shi, He Zong, Yihong Dong, Kechi Zhang, Siyuan Jiang, Zhi Jin, and Ge Li. Aligning llms to fully utilize the cross-file context in repository-level code completion. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1477–1489. IEEE, 2025b.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023a.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 6342–6353, 2023b.

Jiawei Liu, Jia Le Tian, Vijay Daita, Yuxiang Wei, Yifeng Ding, Yuhan Katherine Wang, Jun Yang, and Lingming Zhang. Repoqa: Evaluating long context code understanding. *arXiv preprint arXiv:2406.06025*, 2024.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. Reacc: A retrieval-augmented code completion framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6227–6240, 2022.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. Cot-valve: Length-compressible chain-of-thought tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6025–6035, 2025.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

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, 2002.

Wei Peng, Yan Shi, Yi Wang, Xinyu Zhang, Beijun Shen, and Xiaodong Gu. Swe-qa: Can language models answer repository-level code questions? In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8230–8245, 2026.

Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. Longcodebench: Evaluating coding llms at 1m context windows. *arXiv preprint arXiv:2505.07897*, 2025.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

ByteDance Seed, Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, et al. Seed-coder: Let the code model curate data for itself. *arXiv preprint arXiv:2506.03524*, 2025.

Chuanqi Shi, Miao Gao, and Zhiqiang Gao. Aircoder: Adaptive integration of multi-dimensional retrieval for repository-level code completion. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25458–25470, 2026.

Yuling Shi, Yichun Qian, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. Longcodezip: Compress long context for code language models. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 141–153. IEEE, 2025.

Jan Strich, Florian Schneider, Irina Nikishina, and Chris Biemann. On improving repository-level code qa for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 209–244, 2024.

Yan Wang, Xiaoning Li, Tien N Nguyen, Shaohua Wang, Chao Ni, and Ling Ding. Natural is the best: Model-agnostic code simplification for pre-trained large language models. volume 1, pp. 586–608. ACM New York, NY, USA, 2024.

Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. Codet5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 1069–1088, 2023.

Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin, Weishan Zhao, Li Zhu, Hari Sundaram, et al. Codescope: An execution-based multilingual multitask multidimensional benchmark for evaluating llms on code understanding and generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5511–5558, 2024.

Sheng Zhang, Yifan Ding, Shuquan Lian, Shun Song, and Hui Li. Coderag: Finding relevant and necessary knowledge for retrieval-augmented repository-level code completion. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 23289–23299, 2025.

Zhaowei Zhang, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. Diet code is healthy: Simplifying programs for pre-trained models of code. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1073–1084, 2022.

A Theoretical Analysis and Proofs

A.1 Theoretical Analysis on Coverage

Proposition 1 (Submodularity of AMI Coverage). *The coverage function* $F(S) = \sum_{c_j \in C_{\text{recall}}} \max_{s \in S} a(s \to c_j)$ *is monotone submodular. Consequently, greedy maximization of* F *under a cardinality constraint achieves a* $(1 - 1/e)$ *approximation to the optimum.*

Proof. For any $A \subseteq B \subseteq C_{\text{recall}}$ and $x \in C_{\text{recall}} \setminus B$, define

$$m_A(c_j) = \max_{s \in A} a(s \to c_j), \qquad m_B(c_j) = \max_{s \in B} a(s \to c_j). \tag{13}$$

Since $A \subseteq B$, we have $m_A(c_j) \le m_B(c_j)$ for every $c_j \in C_{\text{recall}}$. The marginal gain of adding x to A for fragment c_j is

$$\Delta_x(c_j \mid A) = \max\big(a(x \to c_j) - m_A(c_j),\, 0\big), \tag{14}$$

and likewise $\Delta_x(c_j \mid B) = \max\big(a(x \to c_j) - m_B(c_j),\, 0\big)$. Since $m_A(c_j) \le m_B(c_j)$, it follows that $\Delta_x(c_j \mid A) \ge \Delta_x(c_j \mid B)$ for every c_j. Summing over all fragments,

$$F(A \cup \{x\}) - F(A) \ge F(B \cup \{x\}) - F(B), \tag{15}$$

which is the submodularity condition. Monotonicity follows directly from the non-negativity of the marginal gains. The $(1 - 1/e)$ approximation guarantee then follows from the classical result for greedy maximization of monotone submodular functions under a cardinality constraint.

A.2 False-Truncation Analysis for Information-Saturation Trigger

We provide a probabilistic analysis of the information-saturation trigger. The goal is to characterize the probability that the trigger incorrectly terminates context expansion when the newly admitted fragment still provides a non-negligible marginal information gain.

Let V_{t-1} be the retained context set before the t-th selection step, and let c_t denote the candidate fragment considered at this step. We define

$$H_t = \text{PPL}(Q \mid V_{t-1} \cup \{c_t\}) \tag{16}$$

as the predictive perplexity of the query after appending c_t. Let

$$\mathcal{F}_{t-1} = \sigma(H_1, \ldots, H_{t-1}, V_{t-1}, c_1, \ldots, c_{t-1}) \tag{17}$$

denote the sigma-algebra containing all information available before H_t is evaluated.

The conditional mean and variance of the current perplexity are

$$m_t = \mathbb{E}[H_t \mid \mathcal{F}_{t-1}], \qquad v_t = \text{Var}(H_t \mid \mathcal{F}_{t-1}). \tag{18}$$

The expected perplexity reduction induced by c_t is

$$\Delta_t^{\text{PPL}} = H_{t-1} - m_t. \tag{19}$$

Assumption 1 (Minimum Useful Gain). *A candidate fragment is useful if its expected perplexity reduction is at least* $\Delta_{\min} > 0$*:*

$$\Delta_t^{\text{PPL}} \ge \Delta_{\min}. \tag{20}$$

Equivalently,

$$m_t \le H_{t-1} - \Delta_{\min}. \tag{21}$$

Assumption 2 (History-Calibrated Variance Bound). *Let* σ_{t-1}^2 *be a history-based uncertainty estimate constructed from* $\mathcal{F}_{t-1}$*. The conditional variance of the current perplexity satisfies*

$$v_t \le \sigma_{t-1}^2. \tag{22}$$

The stopping threshold is constructed before observing H_t:

$$\tau_t = \mu_t + \alpha_t \sigma_t, \tag{23}$$

where μ_t, σ_t, and α_t are $\mathcal{F}_{t-1}$-measurable. The trigger terminates context expansion when

$$H_t > \tau_t. \tag{24}$$

The corresponding false-truncation event is

$$\mathcal{E}_t^{\text{FT}} = \{H_t > \tau_t\} \cap \{m_t \le H_{t-1} - \Delta_{\min}\}. \tag{25}$$

Proposition 2 (False-Truncation Bound). *Under Assumptions 1 and 2, suppose that*

$$d_t = \tau_t - H_{t-1} + \Delta_{\min} = \mu_t + \alpha_t \sigma_t - H_{t-1} + \Delta_{\min} > 0. \tag{26}$$

Then, the conditional false-truncation probability satisfies

$$\Pr\big(\mathcal{E}_t^{\mathrm{FT}} \mid \mathcal{F}_{t-1}\big) \le \frac{v_t}{v_t + d_t^2} \le \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + d_t^2}. \tag{27}$$

Equivalently,

$$\Pr\big(\mathcal{E}_t^{\mathrm{FT}} \mid \mathcal{F}_{t-1}\big) \le \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + [\mu_t + \alpha_t \sigma_t - H_{t-1} + \Delta_{\min}]^2}. \tag{28}$$

Proof. Condition on $\mathcal{F}_{t-1}$. By Assumption 1,

$$m_t \le H_{t-1} - \Delta_{\min}. \tag{29}$$

If the trigger activates, then $H_t > \tau_t$. Hence,

$$\begin{aligned} H_t - m_t &> \tau_t - m_t \\ &\ge \tau_t - (H_{t-1} - \Delta_{\min}) \\ &= \mu_t + \alpha_t \sigma_t - H_{t-1} + \Delta_{\min} \\ &= d_t. \end{aligned} \tag{30}$$

Therefore,

$$\mathcal{E}_t^{\mathrm{FT}} \subseteq \{H_t - m_t > d_t\}. \tag{31}$$

Cantelli's one-sided inequality states that, for any random variable X with finite mean and variance, and any $a > 0$,

$$\Pr(X - \mathbb{E}[X] \ge a) \le \frac{\mathrm{Var}(X)}{\mathrm{Var}(X) + a^2}. \tag{32}$$

Applying Equation (32) conditionally to $X = H_t$, with conditional mean m_t, conditional variance v_t, and $a = d_t > 0$, gives

$$\Pr(H_t - m_t \ge d_t \mid \mathcal{F}_{t-1}) \le \frac{v_t}{v_t + d_t^2}. \tag{33}$$

Combining Equations (31) and (33), we obtain

$$\Pr\big(\mathcal{E}_t^{\mathrm{FT}} \mid \mathcal{F}_{t-1}\big) \le \frac{v_t}{v_t + d_t^2}. \tag{34}$$

Finally, Assumption 2 gives $v_t \le \sigma_t^2$. Since

$$f(v) = \frac{v}{v + d_t^2} \tag{35}$$

is monotonically increasing for $v \ge 0$, it follows that

$$\frac{v_t}{v_t + d_t^2} \le \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + d_t^2}. \tag{36}$$

Substituting the definition of d_t from Equation (26) proves the result.

Corollary 1 (Confidence-Calibrated Trigger). *Let $\delta_t \in (0, 1)$ be a desired upper bound on the per-step false-truncation probability. Under the conditions of Proposition 2, it is sufficient to choose α_t:*

$$\frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + [\mu_t + \alpha_t \sigma_t - H_{t-1} + \Delta_{\min}]^2} \le \delta_t. \tag{37}$$

In particular, a sufficient condition is

$$\alpha_t \ge \frac{H_{t-1} - \mu_t - \Delta_{\min}}{\sigma_t} + \frac{\sigma_{t-1}}{\sigma_t} \sqrt{\frac{1 - \delta_t}{\delta_t}}. \tag{38}$$

Proof. Let

$$d_t = \mu_t + \alpha_t \sigma_t - H_{t-1} + \Delta_{\min}. \tag{39}$$

To ensure Equation (37), it suffices that

$$\frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + d_t^2} \le \delta_t. \tag{40}$$

Rearranging gives

$$d_t^2 \ge \sigma_{t-1}^2 \frac{1-\delta_t}{\delta_t}. \tag{41}$$

Since $d_t > 0$, taking square roots yields

$$d_t \ge \sigma_{t-1} \sqrt{\frac{1-\delta_t}{\delta_t}}. \tag{42}$$

Substituting the definition of d_t and solving for α_t gives Equation (38).

Corollary 2 (Global False-Truncation Control)**.** *Suppose the saturation trigger performs at most T expansion tests. If the per-step false-truncation probabilities satisfy*

$$\Pr\big(\mathcal{E}_t^{\mathrm{FT}} \mid \mathcal{F}_{t-1}\big) \le \delta_t, \qquad \sum_{t=1}^{T} \delta_t \le \delta_{\mathrm{global}}, \tag{43}$$

then the probability of at least one false truncation during the entire expansion process is bounded by

$$\Pr\left(\bigcup_{t=1}^{T} \mathcal{E}_t^{\mathrm{FT}}\right) \le \delta_{\mathrm{global}}. \tag{44}$$

Proof. By the union bound,

$$\Pr\left(\bigcup_{t=1}^{T} \mathcal{E}_t^{\mathrm{FT}}\right) \le \sum_{t=1}^{T} \Pr\big(\mathcal{E}_t^{\mathrm{FT}}\big). \tag{45}$$

Applying the per-step bounds in Equation (43) gives

$$\Pr\left(\bigcup_{t=1}^{T} \mathcal{E}_t^{\mathrm{FT}}\right) \le \sum_{t=1}^{T} \delta_t \le \delta_{\mathrm{global}}. \tag{46}$$

B Implementation Details

Unless otherwise specified, the compression model is identical to the generation model. Regarding the experiments conducted under a fixed budget, token budgets follow task requirements: $B = 2\text{k}$ for code completion and RepoQA, which target local logic and function-level reproduction, and $B = 5\text{k}$ for summarization, which demands module-level coverage. The recall ratio is $p' = 40\%$, and the balance factors are $\lambda = 0.4$ and $\alpha = 0.5$. For the saturation trigger we set the minimum useful gain $\Delta_{\min} = 0.1$ (Assumption 1) and schedule the per-step false-truncation risk budget as $\delta_t = [0.01, 0.10, 0.20, 0.40, 0.50, 0.60, 0.75, 0.80, 0.90, 0.95]$: early expansions are shielded by a large Cantelli confidence margin while later steps tolerate a higher truncation risk as the marginal gain decays, and δ_t stays at 0.95 once the schedule is exhausted. To bound the cost of exact scoring, inter-fragment affinities are evaluated only for the top `max_affinity_candidates` $= 8$ fragments by coarse relevance, and the exact information-gain term is computed only for the top `max_gain_candidates` $= 8$ fragments by g^{info}; the remaining candidates reuse cached estimates. All experiments run on an Intel Xeon Platinum 8558 CPU with a single NVIDIA A100 80G GPU.

Table 4: Performance comparison for Seed-Coder-8B on three benchmarks.

Models	Methods	Long Code Completion				Long Module Summarization			RepoQA		
		ES	EM	Ratio	ACU	CompScore	Ratio	ACU	Avg Acc	Ratio	ACU
Seed-Coder-8B	No Compression	64.04	40.20	1.0×	0.10	44.95	1.0×	0.04	69.0	1.0×	0.07
	No Context	41.88	13.60	–	–	17.42	–	–	0.0	–	–
	RAG	58.51	32.40	2.8×	0.25	42.54	3.0×	0.12	61.7	3.9×	0.23
	LongLLMLingua	54.84	26.40	4.2×	0.33	49.73	2.4×	0.12	71.2	5.1×	0.35
	Selective-Context	56.92	29.00	4.2×	0.35	50.16	2.9×	0.14	66.5	4.4×	0.29
	DietCode	51.43	18.80	3.4×	0.23	44.74	2.1×	0.09	24.0	3.7×	0.09
	LongCodeZip	62.24	36.60	3.9×	0.38	55.04	3.5×	0.19	77.8	4.0×	0.30
	COPS (Ours)	**61.85**	**35.80**	**6.6×**	**0.63**	**54.98**	**4.5×**	**0.24**	**76.1**	**5.6×**	**0.42**

C Evaluation Metrics

For the code summarization task, we calculate the CompScore:

$$\text{CompScore} = \frac{1}{2} \times \Big[P(s_o \succ \hat{s}) + \big(1 - P(\hat{s} \succ s_o)\big) \Big], \tag{47}$$

where $P(s_o \succ \hat{s})$ denotes the probability that the evaluator prefers the generated summary s_o over the reference summary $\hat{s}$, and $P(\hat{s} \succ s_o)$ denotes the probability of the opposite preference. The CompScore ranges from 0 to 100, where 50 indicates equal preference.

D Datasets

We assess COPS on long-code context benchmarks spanning three representative tasks: code completion, code summarization, and code question answering. Prompts for each task are formulated following the settings in (Guo et al., 2023; Bogomolov et al., 2024; Liu et al., 2024; Rando et al., 2025). For long code completion, we adopt Long Code Completion (Guo et al., 2023). comprising 500 Python samples whose contexts surpass 5K tokens. For summarization, we employ Long Module Summarization (Bogomolov et al., 2024). which provides 216 examples drawn from 43 Python repositories; we retain 139 instances with context lengths exceeding 2,000 tokens. RepoQA (Liu et al., 2024) serves as a multilingual QA benchmark encompassing 600 tests across 60 repositories and 6 programming languages. Long Code QA (Rando et al., 2025) targets QA over ultra-long code contexts supporting up to 1M tokens, designed to examine method scalability under extreme context lengths.

E Comparison Results On Seed-Coder-8B

To further validate the generalization capability of COPS beyond the two primary backbones, we conducted additional experiments on the Seed-Coder-8B. These evaluations follow the fixed-budget setting of Section 5.2 across three benchmarks.

The empirical results, summarized in Table 4, demonstrate the strong performance of COPS on Seed-Coder-8B. Under the 2k-token budget, our method compresses the completion context 6.6× while retaining an ES of 61.85, (*i.e.*, 96.6% of the score obtained with all tokens), and raises ACU from 0.38 (LongCodeZip at 3.9×) to 0.66; it matches the strongest baseline's CompScore (54.98 vs. 55.04) at a 4.5× rather than 3.5× ratio on Long Module Summarization. What is particularly noteworthy is that, on Long Module Summarization and RepoQA, the compressed context even surpasses the full-context upper bound (*i.e.*, 54.98 vs. 44.95 and 76.1 vs. 69.0), echoing the observation that long contexts carry distractive redundancy whose removal raises information density, as the mutual-information objective in Equation (1) prescribes. Without per-backbone tuning, it affirms the architecture-agnostic generalization of our framework.

F Performance Comparison under the Same Compression Ratio

Table 5 isolates information density from budget effects by fixing the token retention ratio and comparing task performance directly. COPS leads at every ratio on all three benchmarks, and the margin over the strongest baseline, LongCodeZip, grows as the budget tightens: on RepoQA from $+0.8$ accuracy points at 50% to $+2.3$ at 10%, and on long code completion from $+0.86$ ES at 50% to $+2.16$ at 10%. This is exactly the behavior predicted by the information-gain bound (Equation (5)): when few tokens remain, only a selection that accounts for redundancy within the retained set can keep non-duplicative evidence, whereas static scorers spend the budget on mutually overlapping fragments. The contrast is starkest for LongLLMLingua, whose ES falls from 26.79 to 11.72 as the ratio drops from 50% to 10%: token-level pruning destroys the syntactic scaffolding on which completion depends, while COPS removes whole low-advantage semantic blocks and keeps executable structure intact.

Table 5: Performance comparison under identical token retention ratios using Qwen2.5-Coder-7B.

Retain Ratio	Method	LCC (ES)	LMS (CompScore)	RepoQA (Acc)
50%	LongLLMLingua	26.79	44.87	76.8
	CodeRAG	54.05	54.27	73.3
	AIRCoder	56.04	54.95	78.4
	Selective-Context	47.31	53.25	80.4
	LongCodeZip	57.56	56.12	86.7
	COPS (Ours)	**58.42**	**56.71**	**87.5**
25%	LongLLMLingua	21.86	40.29	71.2
	CodeRAG	52.83	51.64	70.2
	AIRCoder	55.21	52.86	75.8
	Selective-Context	43.75	49.63	76.3
	LongCodeZip	56.43	55.28	86.2
	COPS (Ours)	**57.58**	**56.25**	**86.7**
20%	LongLLMLingua	19.47	38.33	68.6
	CodeRAG	51.07	49.78	68.4
	AIRCoder	54.38	51.42	74.1
	Selective-Context	41.69	44.89	74.7
	LongCodeZip	55.86	53.62	84.7
	COPS (Ours)	**56.97**	**55.76**	**86.0**
15%	LongLLMLingua	16.80	33.67	64.9
	CodeRAG	47.51	45.02	65.3
	AIRCoder	52.14	48.35	72.2
	Selective-Context	36.97	42.28	72.4
	LongCodeZip	54.98	52.74	83.5
	COPS (Ours)	**56.46**	**55.03**	**84.6**
10%	LongLLMLingua	11.72	28.58	59.5
	CodeRAG	42.44	40.73	61.6
	AIRCoder	48.29	42.68	65.3
	Selective-Context	31.59	37.91	67.1
	LongCodeZip	53.44	51.08	80.4
	COPS (Ours)	**55.60**	**53.94**	**82.7**

G Results with Closed-Source Models

To verify that the gains of COPS don't depend on a particular generator, we compress with Qwen2.5-Coder-7B and evaluate the outputs on Claude-3.7-Sonnet (Anthropic, 2025) and GPT-4o (Hurst et al., 2024); Table 6 summarizes the results. On Claude-3.7-Sonnet, COPS at 6.4$\times$ compression retains an ES of 65.77 against 66.24 for the full context, (*i.e.*, 99.3% of the uncompressed performance, and lifts ACU from 0.10 to 0.66). LongCodeZip reaches a comparable ES (65.87) only at

3.7× with ACU 0.38, so COPS roughly doubles the achievable compression at parity. The same pattern holds on GPT-4o (*i.e.*, 63.78 ES at 6.4×, ACU 0.63 against 0.10 uncompressed). On summarization, COPS matches the strongest baseline's CompScore (*i.e.*, 60.25 vs. 60.96 on Claude-3.7-Sonnet, 58.46 vs. 58.77 on GPT-4o) while compressing 3.8× against 1.7×, doubling ACU (*i.e.*, 0.22 vs. 0.10). These results demonstrate the practical applicability of our method in real-world settings, where small-model compression drastically reduces the inference costs of large closed-source models.

Table 6: Results with closed-source generators. Compression is performed by Qwen2.5-Coder-7B in all rows.

	Long Code Completion / Claude-3.7-Sonnet				Long Code Completion / GPT-4o				LMS / Claude-3.7-Sonnet			LMS / GPT-4o		
Method	ES	EM	Ratio	ACU	ES	EM	Ratio	ACU	CompScore	Ratio	ACU	CompScore	Ratio	ACU
No Compression	66.24	41.20	1.0×	0.10	65.13	40.80	1.0×	0.10	60.72	1.0×	0.06	58.42	1.0×	0.06
No Context	43.97	14.20	–	–	42.92	14.00	–	–	6.58	–	–	6.41	–	–
RAG	61.44	34.00	2.8×	0.26	60.03	33.20	2.8×	0.25	58.03	1.7×	0.10	55.85	1.7×	0.09
LongLLMLingua	57.58	27.80	3.2×	0.27	56.24	27.20	3.2×	0.26	50.86	1.5×	0.07	48.89	1.5×	0.07
Selective-Context	57.63	31.40	2.6×	0.23	55.78	29.80	2.6×	0.22	59.15	1.9×	0.12	57.26	1.9×	0.11
DietCode	54.00	19.80	3.4×	0.25	52.76	19.40	3.4×	0.24	38.82	2.1×	0.08	37.21	2.1×	0.08
LongCodeZip	65.87	40.00	3.7×	0.38	64.25	38.40	3.7×	0.37	60.96	1.7×	0.10	58.77	1.7×	0.10
COPS (Ours)	65.77	40.00	**6.4**×	**0.66**	63.78	37.40	**6.4**×	**0.63**	60.25	**3.8**×	**0.22**	58.46	**3.8**×	**0.22**

H Algorithm

Algorithm 1 details the iterative admission rule of the information-saturation trigger. Starting from the first u_0 candidates, the procedure appends candidates in coarse-score order and monitors the global perplexity $H_t = \mathrm{PPL}(q \mid \mathcal{U} \cup \{c_t\})$. After each admission the threshold is rebuilt from the history $\mathcal{P}$ as $\tau_t = \mu_t + \alpha_t \sigma_t$, with the coefficient α_t taken in the confidence-calibrated form of Corollary 1, where δ_t is read from the risk schedule of Appendix B and $\Delta_{\min}$ encodes Assumption 1. Expansion halts at the first t with $H_t > \tau_t$: since $H_{t-1} - H_t$ approximates the marginal information gain ΔI_t (Equation 4), the stopping condition fires precisely when the newest candidate's gain becomes statistically indistinguishable from perplexity noise, and Proposition 2 bounds the probability of discarding a still-useful fragment at each step.

I Hyperparameter Sensitivity Analysis

Table 7 varies each hyperparameter around the default configuration of Appendix B on Long Code Completion using Qwen2.5-Coder-7B. Performance is flat across the tested ranges: ES moves within 0.59 points for $\lambda \in [0.2, 0.8]$ and within 0.42 for $\alpha \in [0.3, 0.7]$, confirming that the joint score does not rely on finely tuned weighting between the three principles. The recall ratio p' exhibits the expected recall–cost trade-off: at 20% the candidate pool occasionally drops dependency-critical fragments (ES 56.28), while 60% adds roughly 46% compression time for no measurable gain. The two candidate caps behave as pure efficiency knobs: halving them to 4 costs 0.39–0.47 ES because exact affinity and gain terms are skipped for promising candidates, whereas doubling them to 16 changes ES by at most 0.07 while increasing affinity computation time by about 65%. We therefore keep both caps at 8 as the accuracy–cost trade-off. Overall, these ablation studies demonstrate the robustness of our method to variations in key hyperparameters. The consistent performance across diverse benchmarks and base models validates our choice of hyperparameters as the default settings in the main experiments.

J Case Studies

We provide qualitative case studies illustrating the efficacy of COPS in preserving critical class-level semantics and removing redundant contexts for accurate code completion.

As illustrated in the Figure 4, our method selects only the top three context segments from the baseline's list, yet achieves improved rather than degraded performance. The rationale is as follows. We attribute it to the orthogonal complementarity of these three functions. Specifically, Idx 36 provides the core domain-specific logic and exception handling. Unlike generic file operations,

Algorithm 1 Information-Saturation Trigger Mechanism

Require: Ranked coarse candidate fragments $C_{\text{coar}} = \{c_1, c_2, \ldots, c_k\}$ (sorted by preservation score), task query Q, initial fragment count v_0, per-step false-truncation risk budget $\delta_t \in (0, 1)$, minimum useful perplexity reduction $\Delta_{\text{min}} > 0$

Ensure: Final retained context subset $U^* \subseteq C_{\text{coar}}$

Initialize retained set $V \leftarrow \{c_1, c_2, \ldots, c_{v_0}\}$
Initialize perplexity history sequence $\mathcal{P} \leftarrow \{\text{PPL}(Q \mid V)\}$
$t \leftarrow v_0$ {current expansion step index}
while $t < k$ **do**
  $t \leftarrow t + 1$
  Fetch the t-th candidate fragment $c_t \in C_{\text{coar}}$
  Compute global system perplexity after adding c_t:
    $H_t = \text{PPL}(Q \mid V \cup \{c_t\})$
  Estimate historical statistics from all prior perplexity values in $\mathcal{P}$:
    $\mu_t = \mathbb{E}[\mathcal{P}], \quad \sigma_t = \sqrt{\text{Var}[\mathcal{P}]}$
  Calibrate confidence coefficient via Cantelli's one-sided concentration bound:
    $\alpha_t = \max\left(0, \frac{H_{t-1} - \mu_t - \Delta_{\text{min}}}{\sigma_t} + \frac{\sigma_{t-1}}{\sigma_t}\sqrt{\frac{1-\delta_t}{\delta_t}}\right)$
  Compute adaptive saturation threshold:
    $\tau_t = \mu_t + \alpha_t \cdot \sigma_t$
  if $H_t > \tau_t$ **then**
    break {saturation triggered: marginal information gain is negligible}
  end if
  Update retained set and perplexity history:
    $V \leftarrow V \cup \{c_t\}$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{H_t\}$
end while
$U^* \leftarrow V$
return U^*

Table 7: Sensitivity of COPS to its main hyperparameters on long code completion using Qwen2.5-Coder-7B; defaults in bold.

Group	Variant	ES	EM	ACU
Coverage weight λ	0.2	56.41	30.40	0.53
	0.4	**56.94**	**31.00**	**0.55**
	0.6	56.72	30.80	0.54
	0.8	56.35	30.40	0.52
Gain weight α	0.3	56.52	30.60	0.53
	0.5	**56.94**	**31.00**	**0.55**
	0.7	56.61	30.60	0.54
Recall ratio p'	20%	56.28	30.20	0.54
	40%	**56.94**	**31.00**	**0.55**
	60%	56.87	30.80	0.55
Max affinity candidates	4	56.47	30.40	0.53
	8	**56.94**	**31.00**	**0.55**
	16	56.98	31.00	0.55
Max gain candidates	4	56.55	30.60	0.54
	8	**56.94**	**31.00**	**0.55**
	16	57.01	31.20	0.55

`mknod` necessitates a unique `try-except` block to capture system-specific `OSError` exceptions (e.g., `EPERM` or `EINVAL`) for unprivileged users, alongside the `stat.S_ISFIFO` validation. Idx 70 supplies the **structural scaffolding and metadata hints** for the `dir_fd` paradigm. It establishes the standard boilerplate for acquiring the file descriptor and wrapping the execution in a

try-finally block for resource cleanup; crucially, its trailing text explicitly contains the exact decorator (@unittest.skipUnless) required for the target function, serving as a direct structural prompt. Finally, Idx 35 acts as a **semantic anchor and logical closure**. It semantically justifies the hasattr(stat, 'S_IFIFO') condition present in the decorator from Idx 70 and reinforces the canonical paradigm for FIFO validation. During autoregressive generation, the attention mechanism effectively *stitches* these orthogonal components: extracting the decorator and dir_fd scaffolding from Idx 70, injecting the mknod-specific exception handling from Idx 36 into the try block with the dir_fd=f argument, and finalizing the validation logic in the else block reinforced by Idx 35. Other dir_fd test cases (e.g., Idx 65–67) are structurally homogenized with Idx 70 and thus introduce contextual noise rather than novel information. Consequently, this triad achieves maximal coverage of both structural and semantic requirements with minimal information entropy, enabling flawless code derivation.

Code to be Completed

```
def test_mknod_dir_fd(self):
    # Test using mknodat() to create a FIFO (the only use specified by POSIX).
    support.unlink(support.TESTFN)
    mode = stat.S_IFIFO | stat.S_IRUSR | stat.S_IWUSR
```

Ground Truth Completion

```
f = posix.open(posix.getcwd(), posix.O_RDONLY)
```

Selective Functions from Context

```
36
def test_mknod(self):
    # Test using mknod() to create a FIFO (the only use specified by POSIX)
    support.unlink(support.TESTFN)
    mode = stat.S_IFIFO | stat.S_IRUSR | stat.S_IWUSR
    ...
70
def test_mkdir_dir_fd(self):
    f = posix.open(posix.getcwd(), posix.O_RDONLY)
    try:
        posix.mkdir(support.TESTFN + 'dir', dir_fd=f)
        posix.stat(support.TESTFN + 'dir')  # should not raise exception
    finally:
        posix.close(f)
        support.rmtree(support.TESTFN + 'dir')
    ...

35
def test_mkfifo(self):
    support.unlink(support.TESTFN)
    posix.mkfifo(support.TESTFN, stat.S_IRUSR | stat.S_IWUSR)
    self.assertTrue(stat.S_ISFIFO(posix.stat(support.TESTFN).st_mode))
    ...
```

Analysis

```
def test_mknod_dir_fd(self):
    # Test using mknodat() to create a FIFO
    support.unlink(support.TESTFN)
    mode = stat.S_IFIFO | stat.S_IRUSR | stat.S_IWUSR
    #     To be completed:
    # 1. How to obtain dir_fd?
    # 2. How to call mknod with the dir_fd parameter?
    # 3. How to verify the result?
```

The baseline selects context index segments including [36, 70, 35, 66, 63, 68, 67, 69, 34, 65, 58, 41]. In contrast, our method only retains the Top-3 context segments [36, 70, 35]. This design not only drastically cuts redundant context but also shields the LLM from irrelevant contextual distractions, yielding substantial improvements in code completion performance. The ES metric rises sharply from merely 2.17 under the baseline to 100.00 (↑98), which verifies the effectiveness of our proposed method.

Figure 4: Code Completion Case Study: Our method achieves a 6.4× compression ratio while removing redundant context functions, Qwen2.5-Coder-7B as backbone.