# RepoShapley: Shapley-Enhanced Context Filtering for Repository-Level Code Completion

**Yu Huo[♠*], Kun Zeng[♣*], Siyu Zhang[♡], Yuquan Lu[♣], Cheng Yang[◇], Yifu Guo[♣], Xiaoying Tang[♠♯‡†]**

[♠]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen
[♯]Shenzhen Future Network of Intelligence Institute (FNii-Shenzhen)
[‡]Guangdong Provincial Key Laboratory of Future Networks of Intelligence, CUHK(SZ)
[♡]University of California, San Diego [♣]Sun Yat-sen University [◇]Hangzhou Dianzi University

**Email:** yuhuo@link.cuhk.edu.cn, tangxiaoying@cuhk.edu.cn

**GitHub:** https://github.com/yuhuo03/RepoShapley

## Abstract

Repository-level code completion benefits from retrieval-augmented generation (RAG). However, controlling cross-file evidence is difficult because chunk utility is often interaction-dependent: some snippets help only when paired with complementary context, while others harm decoding when they conflict. We propose RepoShapley, a coalition-aware context filtering framework supervised by Shapley-style marginal contributions. Our offline labeling module, **ChunkShapley**, estimates signed per-chunk effects via teacher-forced probing, feeds them into a lightweight surrogate game that captures saturation and interference, computes exact Shapley values for small retrieval sets, and selects a decoding-optimal coalition through bounded post-verification with the frozen generator. The verified <KEEP>/<DROP> decisions and retrieval triggers are then distilled into a single model via discrete control tokens. Experiments across benchmarks and backbones show that RepoShapley improves completion quality while reducing harmful context and unnecessary retrieval.

## 1 Introduction

Large language models have demonstrated strong reasoning, coding, and generation capabilities (Brown et al., 2020; Wei et al., 2022; Chen et al., 2021; Liu et al., 2026). Yet repository-level code completion must resolve non-local dependencies such as project-specific APIs, shared contracts, and invariants (Jimenez et al., 2024; Ding et al., 2024b). Retrieval-Augmented Generation (RAG) injects cross-file evidence into Code LMs (Lewis et al., 2020; Kang et al., 2024; Shrivastava et al., 2023; Bairi et al., 2023), but retrieval control remains difficult under fixed context budgets: the model must filter redundant or misleading chunks from a noisy candidate pool (Ding et al., 2024a; Zhang et al., 2023; Wei et al., 2025; Liu et al., 2024a; Yoran et al., 2024).

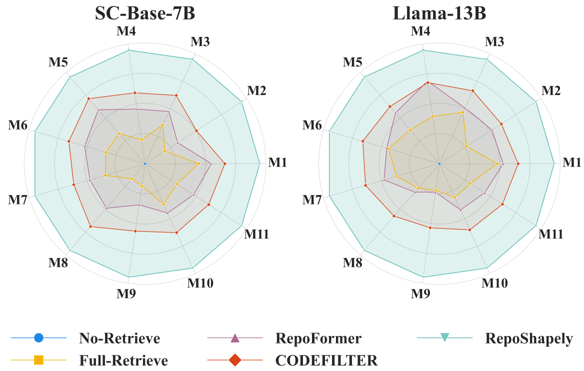


Figure 1: Performance radar charts on StarCoder-Base-7B and CodeLlama-13B. The plots display relative improvements over the No-Retrieve baseline (center). RepoShapley achieves the best performance among compared methods across 11 tested metrics; see Table 1.

The core difficulty is that chunk utility is often interaction-dependent. A snippet may appear uninformative in isolation yet become decisive when paired with complementary context, such as an interface declaration together with its implementation. Conversely, a plausible chunk can degrade generation when it co-occurs with conflicting evidence, such as deprecated versus updated APIs (Shi et al., 2023; Xu et al., 2024). Therefore, methods that score candidates independently can misestimate the utility of the multi-chunk context that is actually consumed at test time (Khandelwal et al., 2020; Yan et al., 2024; Bertsch et al., 2025).

To address this, we adopt a coalition-first approach. Retrieval control should be supervised by signals that reflect how a chunk behaves within a set, rather than in isolation. We introduce RepoShapley, a framework that learns to filter context using Shapley-style marginal contributions.

Our approach has two stages. First, we propose **ChunkShapley**, an offline labeling pipeline for interaction-aware supervision. Considering that

[*]Equal contribution
[†]Corresponding author



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 10390–10412

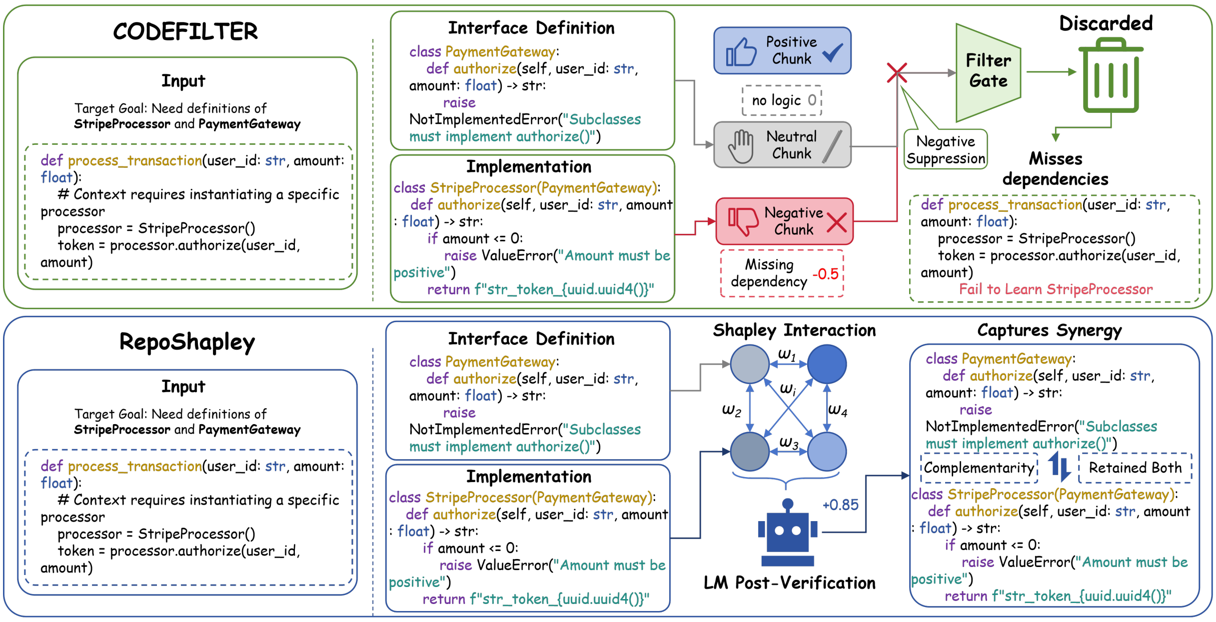


Figure 2: Under the same input context and the exact same retrieved candidate chunks, CODEFILTER makes decisions from independent per-chunk signals and can break under interaction effects, whereas REPOSHAPLEY performs coalition-aware filtering that more reliably removes high-score noise while preserving complementary evidence.

computing Shapley values directly with the generator is prohibitive, we introduce a structured logistic surrogate that can capture saturation and conflict efficiently. We then apply a verification step to ground the selected coalitions in the generator's actual decoding behavior. Second, we distill the resulting coalition-derived labels into a single generator via discrete control tokens, which we call REPOSHAPLEY. This distillation enables efficient, interaction-aware retrieval control at inference time. As shown in Figure 1, REPOSHAPLEY achieves the best performance under our evaluated settings, supporting our motivation that coalition-aware supervision is crucial for difficult cross-file completion. Our contributions are as follows:

- **Coalition-aware supervision for context filtering.** We formulate context selection as a cooperative game and use Shapley marginal contributions to capture complementarity and conflict beyond independent scoring.
- **ChunkShapley: Practical Shapley labeling for chunk filtering.** We combine single-chunk probing with a structured surrogate utility to compute exact Shapley values on small retrieval sets ($K{=}10$). We further select a verified coalition from a bounded candidate pool under decoding-time metrics.
- **REPOSHAPLEY: distillation for online retrieval control.** We distill verified keep and drop decisions into discrete control tokens, enabling a single model to decide when to retrieve and which chunks to keep.

## 2 Related Work

**Repository-Level RAG and Retrieval Control.** RAG mitigates non-local dependencies in code completion by retrieving cross-file evidence (Lewis et al., 2020; Izacard and Grave, 2021; Parvez et al., 2021; Guu et al., 2020; Jiang et al., 2023; Mallen et al., 2023; Yao and Fujita, 2024). Recent work improves context quality via iterative retrieval (Gao et al., 2023; Zhang et al., 2023; Shrivastava et al., 2023; Zhang et al., 2025), structure-aware indexing, including dataflow or call graphs (Cheng et al., 2024; Liu et al., 2024c), and dedicated benchmarks (Ding et al., 2023; Liu et al., 2024b; Li et al., 2025; Wang et al., 2025; Yang et al., 2025). In parallel, retrieval control has received increasing attention, focusing on when to retrieve and what to retain under a fixed context budget. RepoFormer (Wu et al., 2024) triggers retrieval through self-evaluation, while CODEFILTER (Li et al., 2025) filters chunks using independent likelihood-based signals. However, these controllers largely assess chunks in isolation. As a result, they do not explicitly account for combinatorial interactions such as complementarity between interfaces and implementation. In contrast, we cast context filtering as a coalition scoring problem to model such interdependencies.

**Shapley Values in RAG and Supervision.** Shapley values (Shapley, 1953) provide an axiomatic notion of marginal contribution and have been widely used in interpretability, including SHAP-style formulations (Lundberg and Lee, 2017; Ghorbani and Zou, 2019; Sundararajan et al., 2017). In RAG, prior work applies Shapley-style analysis to attribute outputs to retrieved documents (Nematov et al., 2025; Ye and Yoganarasimhan, 2025) or to estimate token-level importance (Asai et al., 2024; Xiao et al., 2025). Our use differs along three axes. First, whereas SHAP and Data Shapley (Ghorbani and Zou, 2019) perform *post-hoc* attribution on a frozen model, we use Shapley marginalization to *construct supervision* for an active retrieval controller. Second, document-level Shapley in RAG (Nematov et al., 2025; Ye and Yoganarasimhan, 2025) treats each retrieved passage as an independent player; we instead operate at the *chunk* level within a single repository and explicitly model coalition effects such as saturation and conflict. Third, TokenShapley (Xiao et al., 2025) attributes importance to individual tokens, whereas our formulation attributes importance to *subsets* of retrieved chunks, capturing inter-chunk synergies that token-level analysis cannot express. We then distill the resulting coalition reasoning into a token-level policy, enabling practical retrieval decisions during generation.

# 3 Methodology

## 3.1 Repository-level Retrieval-Augmented Code Completion

Repository-level code completion requires grounding generation in cross-file information such as project-specific APIs, shared utilities, and type or contract conventions. RAG addresses this by retrieving candidate snippets from the repository. However, retrieved evidence is often interaction-heavy: a snippet may be useful only when paired with complementary context, and seemingly relevant snippets can degrade generation when they introduce conflicting implementations.

**Problem setup.** Given a repository $\mathcal{R}$ and a target file, each instance is represented as $(X_{\text{in}}, X_{\text{out}}, Y)$. Here $X_{\text{in}} = (X_p, X_s)$ is the in-file context in fill-in-the-middle (FIM) format with prefix $X_p$ and suffix $X_s$, $X_{\text{out}}$ denotes a cross-file pool constructed from other files in $\mathcal{R}$, and $Y$ is the ground-truth missing span between $X_p$ and $X_s$ (Zhang et al., 2023; Wu et al., 2024).

**Retrieval and generation.** A retriever $R$ queries $X_{\text{out}}$ with $X_{\text{in}}$ and returns top-$K$ candidate chunks $X_{\text{cc}} = R(X_{\text{in}}, X_{\text{out}}) = \{cc_1, \ldots, cc_K\}$. A generator $G_\theta$ then predicts the completion $\hat{Y}$ conditioned on $X_{\text{in}}$ and a selected subset $X_S \subseteq X_{\text{cc}}$. Hence, the key problem is to estimate chunk utility and retain the subset that best supports generating $Y$.

## 3.2 Interaction-aware Chunk Attribution via Shapley Values

**Why independent chunk scoring is insufficient.** Retrieved code snippets rarely contribute independently. A chunk can be uninformative on its own but become essential when paired with complementary context such as an interface and its implementation. Conversely, a seemingly relevant snippet may reduce generation quality when it conflicts with other retrieved evidence. As a result, per-chunk scores computed in isolation can be a poor proxy for the utility of the multi-chunk context used at test time.

**Subset utility as a cooperative game.** We therefore evaluate chunks at the set level. Given top-$K$ candidates, we treat each chunk as a player and any subset as a coalition. Let $D = \{1, \ldots, K\}$ index candidates and $S \subseteq D$ denote a coalition, with $X_S = \{cc_i : i \in S\}$. We define the coalition value as the normalized teacher-forced log-likelihood gain on the ground-truth completion:

$$
\begin{aligned}
v(S \mid X_{\text{in}}, Y) &= \ell(X_{\text{in}}, X_S) - \ell(X_{\text{in}}) \\
\ell(C) &= \frac{1}{|Y|} \log p_\theta(Y \mid C).
\end{aligned}
$$

where $\log p_\theta(Y \mid C) = \sum_{t=1}^{|Y|} \log p_\theta(y_t \mid y_{<t}, C)$. By construction, $v(\emptyset \mid X_{\text{in}}, Y) = 0$, and $v(S)$ can be negative when retrieved context decreases model likelihood. Appendix C.4 compares log-likelihood with metric-based utilities (EM/ES) and shows log-likelihood yields the best downstream performance.

**Shapley attribution.** We quantify interaction-aware chunk contributions using the Shapley value (Shapley, 1953), which is defined as the average marginal gain of chunk $i$ over all coalitions:

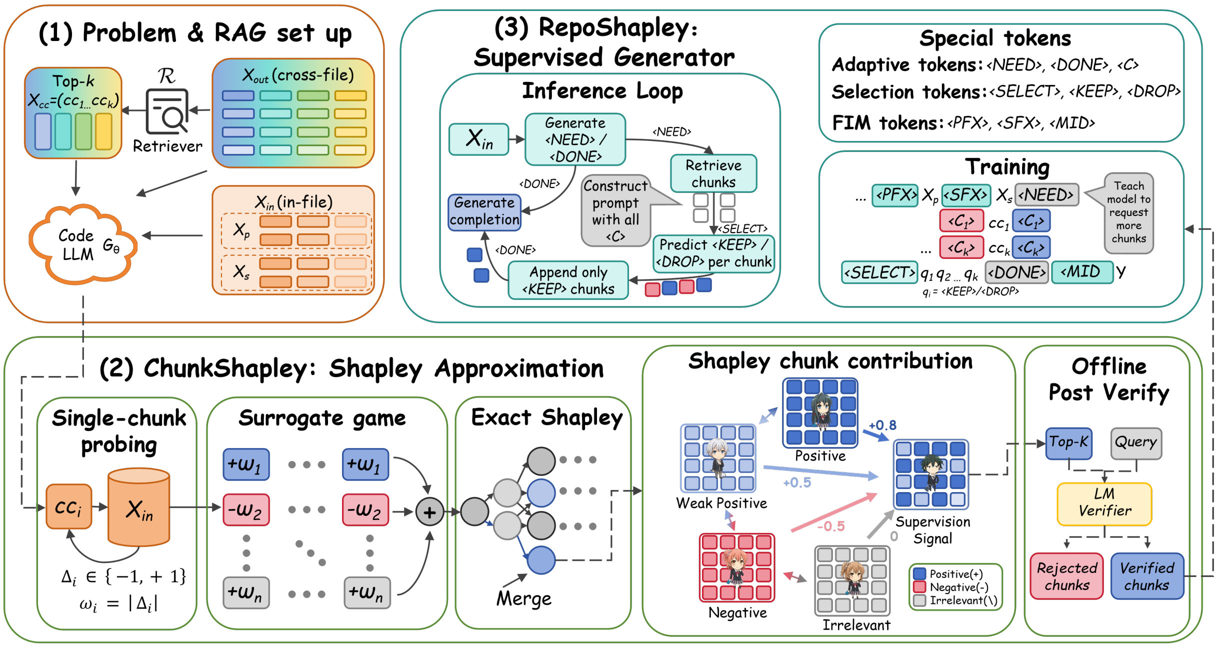


Figure 3: The overall framework of REPOSHAPLEY. The pipeline consists of two phases: (2) An offline ChunkShapley module that estimates the interaction-aware contribution of each chunk; and (3) An online Shapley-supervised Generator trained to control retrieval and filter contexts based on the estimated Shapley values.

$$\phi_i = \sum_{S\subseteq D\setminus\{i\}} \frac{|S|!\,(K-|S|-1)!}{K!}\,\Delta v_i(S)$$
$$\Delta v_i(S) = v(S\cup\{i\}\mid X_{\text{in}},Y) - v(S\mid X_{\text{in}},Y).$$

Intuitively, $\phi_i > 0$ indicates that chunk $i$ is helpful on average across different co-occurring contexts, while $\phi_i \leq 0$ suggests redundancy or harm under interactions. Shapley values satisfy *efficiency*: $\sum_{i\in D}\phi_i = v(D \mid X_{\text{in}},Y)$, allowing negative attributions when some chunks reduce coalition utility.

### 3.3 ChunkShapley: Practical Shapley Labeling for Chunk Filtering

Exact Shapley computation under the true coalition utility $v(\cdot)$ is impractical, as it would require evaluating the generator on exponentially many subsets. We therefore propose **ChunkShapley**, an *offline* labeling pipeline that (a) probes each chunk once to obtain a signed effect, (b) defines a lightweight surrogate game to approximate interaction patterns, (c) computes exact Shapley values under the surrogate by enumerating all $2^K$ coalitions, which is inexpensive since $v_{\text{sur}}(\cdot)$ is closed-form, and (d) performs bounded post-verification with the frozen generator to ground final keep and drop labels in decoding-time behavior. Algorithmic details are deferred to Appendix Algorithm 2.

**(a) Single-chunk probing.** We first compute a per-instance baseline score using teacher forcing and probe each candidate in isolation. Let $\ell(C)$ denote the normalized teacher-forced log-likelihood. For each retrieved chunk $cc_i$, we define its single-chunk effect

$$\Delta_i = \ell(X_{\text{in}},\{cc_i\}) - \ell(X_{\text{in}})$$
$$y_i = \text{sign}(\Delta_i), \quad \omega_i = |\Delta_i|.$$

To ensure consistent likelihood estimation under a limited context window, we preserve the full target span $Y$ and apply **left-truncation** only to the input context (i.e., $X_{\text{in}}$ and retrieved chunks).

**(b) Logistic surrogate game.** While ranking by $\Delta_i$ captures individual relevance, it ignores coalition dynamics. To model interactions efficiently, we define a one-dimensional surrogate utility. Given $(y_i,\omega_i)$, we aggregate coalition $S$ via a weighted vote:

$$g(S) = \sum_{i\in S}\omega_i y_i, \qquad v_{\text{sur}}(S) = \sigma(\beta\, g(S)) - \sigma(0).$$

where $\sigma(\cdot)$ is the sigmoid and $\beta > 0$ controls the saturation scale. This surrogate is not meant to match the full combinatorial utility; it targets two dominant effects for filtering. The sigmoid yields diminishing returns: when $|g(S)|$ is large, $\sigma'(\beta g(S)) \approx 0$, so additional similarly-signed evidence contributes little, capturing redundancy under a fixed budget. Conflicts are expressed by negative votes ($y_i = -1$), which reduce $g(S)$ and can suppress $v_{\text{sur}}(S)$ even when some chunks are individually helpful. Subtracting $\sigma(0)$ ensures $v_{\text{sur}}(\varnothing) = 0$ and keeps utilities centered. The surrogate remains lightweight for exhaustive subset evaluation, while any residual mismatch to decoding-time behavior is addressed by verification.

**(c) Exact Shapley values under the surrogate.** We compute Shapley values using the subset form under the surrogate utility:

$$\phi_i = \frac{1}{K} \sum_{S \subseteq D \setminus \{i\}} \frac{v_{\text{sur}}(S \cup \{i\}) - v_{\text{sur}}(S)}{\binom{K-1}{|S|}}.$$

Since our $v_{\text{sur}}(S)$ is closed-form, evaluating all $2^K$ subsets is computationally negligible for small retrieval sizes ($K \leq 10$). This allows us to obtain *exact* Shapley values under $v_{\text{sur}}$, avoiding the variance of sampling approximations.

In contrast, computing interactions using the heavy generator $G_\theta$ would require exponentially many coalition evaluations and is intractable. Therefore, we use $\phi_i$ under the surrogate as a proposal signal and rely on post-verification to finalize the decision.

**(d) Post-verification via a bounded candidate pool.** Because decoding quality is non-monotonic in context, positive attributions alone do not guarantee improved greedy decoding. Since the surrogate is only a proxy, we verify a small candidate pool with the frozen generator and select the coalition that maximizes decoding-time quality. This step is used only for offline label construction with access to $Y$; inference never uses $Y$.

Let $\pi_\phi$ and $\pi_\Delta$ be indices sorted by $\phi_i$ and $\Delta_i$. We build a de-duplicated set $\mathcal{C}$ containing: (i) Shapley prefixes $\{\pi_\phi[1{:}n]\}_{n=1}^{N_v}$, (ii) short $\Delta$ prefixes as a strong single-chunk baseline, and (iii) size-2/3 combinations among top-$L$ chunks by $\Delta$ to explicitly probe local synergies. For each $S \in \mathcal{C}$, we decode with the frozen generator and choose

$$S^\star = \arg\max_{S \in \mathcal{C}} \left(\text{ES}(\hat{Y}_S, Y),\ \text{EM}(\hat{Y}_S, Y)\right)$$

using lexicographic maximization (ES first, EM as tie-break). We then treat $S^\star$ as the teacher keep/drop labels for distillation.

**Verified labels for retrieval triggering.** The post-verification step also yields an oracle decision on whether retrieval is necessary. Let $\hat{Y}_\emptyset$ be the decoding result using only in-file context $X_{\text{in}}$, and let $\hat{Y}_{S^\star}$ be the decoding result using the verification-selected coalition $S^\star$. Let $\Delta_{\text{ES}} = \text{ES}(\hat{Y}_{S^\star}, Y) - \text{ES}(\hat{Y}_\emptyset, Y)$. We define the retrieval-control label as

$$r^\star = \begin{cases} \texttt{<DONE>}, & \text{if } \Delta_{\text{ES}} \leq \epsilon \\ \texttt{<NEED>}, & \text{otherwise.} \end{cases}$$

where $\epsilon$ is a small margin tuned on the validation set (default $\epsilon = 0$ unless stated otherwise). This label is used only for offline supervision; inference never accesses $Y$.

### 3.4 RepoShapley: Distilling ChunkShapley into Signal Tokens

While ChunkShapley provides robust coalition-aware supervision, the pipeline is too computationally intensive for online use. We therefore propose RepoShapley, which *distills* verified coalition decisions into discrete control tokens, enabling a single generator to efficiently decide *when* to retrieve and *which* chunks to retain at inference time.

**Signal tokens and verified labels.** We introduce retrieval-control tokens $\mathcal{T}_R = \{\texttt{<NEED>}, \texttt{<DONE>}\}$ to decide whether cross-file evidence is required, and candidate-selection tokens $\mathcal{T}_S = \{\texttt{<KEEP>}, \texttt{<DROP>}\}$ to indicate which retrieved chunks should be retained.

Step (d) outputs a *verification-selected coalition* $S^\star$ by evaluating a small set of Shapley-proposed candidate coalitions $\mathcal{C}$ using the frozen generator under decoding-time constraints, to match decoding-time behavior. We treat $S^\star$ as the teacher keep/drop label set and distill it into token-level supervision by assigning, for each retrieved chunk $cc_i$,

$$Q(cc_i) = \begin{cases} \texttt{<KEEP>} & \text{if } i \in S^\star \\ \texttt{<DROP>} & \text{otherwise.} \end{cases}$$

In this way, surrogate Shapley signals are used only to propose promising coalitions, while the student model learns to imitate the *verified* coalition-level behavior encoded by $S^\star$, turning combinatorial subset selection into a single-shot, controllable generation policy at inference time.

**Training: two-format verbalized supervision.** Following the standard separation of evidence selection and completion generation in retrieval-augmented code modeling, we train a single model with two serialized views of each instance. Format-1 supervises *selection*: given the in-file context and retrieved candidates, the model emits a keep/drop decision token for each chunk. Format-2 supervises *generation*: the model produces the missing span conditioned only on the kept evidence. Both formats reuse the same control tokens and share all parameters, enabling the model to learn selection and generation within a unified autoregressive interface.

**Format-1: Selection.** Given the in-file context and the retrieved candidate list, the model predicts a length-$K$ decision sequence $q_{1:K} \in \{\texttt{<KEEP>}, \texttt{<DROP>}\}^K$ under a dedicated <SELECT> marker. Let $[X_p]$ and $[X_s]$ denote tokenized FIM prefix and suffix, and let $\mathrm{Pack}(X_{\mathrm{cc}})$ be the deterministic serialization of retrieved candidates $X_{\mathrm{cc}} = \{cc_1, \ldots, cc_K\}$:

$$\mathrm{Pack}(X_{\mathrm{cc}}) = \texttt{<C\_1>}[cc_1]\texttt{</C\_1>} \cdots \texttt{<C\_K>}[cc_K]\texttt{</C\_K>}.$$

The Format-1 sequence is

$$\mathsf{F1}: \quad \texttt{<PFX>}[X_p]\ \texttt{<SFX>}[X_s]\ \texttt{<NEED>}\ \mathrm{Pack}(X_{\mathrm{cc}})\ \texttt{<SELECT>}\ q_1\ q_2\ \cdots\ q_K\ \texttt{<DONE>}.$$

We supervise $q_i$ using the *verified teacher coalition* $S^\star$: $q_i^\star = \texttt{<KEEP>}$ if $i \in S^\star$ and <DROP> otherwise.

**Format-2: Generation.** To teach the model how to complete code *given filtered evidence*, we construct a generation format that includes only the chunks in $S^\star$ and then decodes the target span in FIM mode:

$$\mathsf{F2}: \quad \texttt{<PFX>}[X_p]\ \texttt{<SFX>}[X_s]\ \texttt{<NEED>}\ \mathrm{Pack}(C_{S^\star})\ \texttt{<DONE>}\ \texttt{<MID>}\ [Y].$$

**No-retrieval format.** If retrieval is unnecessary ($r^\star = \texttt{<DONE>}$), we drop the cross-file block and the selection head:

$$\texttt{<PFX>}[X_p]\ \texttt{<SFX>}[X_s]\ \texttt{<DONE>}\ \texttt{<MID>}\ [Y].$$

This indicates that in-file context suffices.

*Remark.* We reuse <DONE> both as the retrieval decision token and as a block delimiter; the two usages are unambiguous from their fixed positions in the sequence. The retrieval-control token (<NEED>/<DONE>) is learned with teacher forcing as a next-token target (counted in $\mathcal{L}_R$), rather than provided as an oracle input.

---

**Algorithm 1:** RepoShapley Inference Process

**Input:** Generator $G$, Retriever $R$, Cross-file pool $X_{\mathrm{out}}$, In-file context $X_{\mathrm{in}} = (X_p, X_s)$;
Token sets $\mathcal{T}_R = \{\texttt{<NEED>}, \texttt{<DONE>}\}$, $\mathcal{T}_S = \{\texttt{<KEEP>}, \texttt{<DROP>}\}$; threshold $t_c$.
**Output:** Completed code $\hat{Y}$.

```
X ← (<PFX>, X_p, <SFX>, X_s)
r ← Select(Softmax_{T_R}(G(· | X)), t_c)
if r = <DONE> then
    X ← append(X, <MID>)
    return Ŷ ← G(X)
end
X_cc ← R(X_in, X_out)
X_sel ←
    X ⊕ <NEED> ⊕ Pack(X_cc) ⊕ <SELECT>
(q_1, ..., q_K) ← G(X_sel)
Ŝ ← {i ∈ {1, ..., K} : q_i = <KEEP>}
X ← X ⊕ <NEED> ⊕ Pack(C_Ŝ) ⊕ <DONE>
X ← append(X, <MID>)
return Ŷ ← G(X)
```

---

**Objectives with masked contexts.** We mask all in-file and cross-file *content* tokens in the loss and compute gradients only on *generated targets* (control tokens, selection tokens, and the completion $Y$). Let $r^\star \in \mathcal{T}_R = \{\texttt{<NEED>}, \texttt{<DONE>}\}$ be the retrieval-control label. For retrieval-needed instances (Formats F1/F2), $r^\star = \texttt{<NEED>}$; for no-retrieval instances, $r^\star = \texttt{<DONE>}$. For Format-1, we optimize retrieval triggering and selection:

$$\begin{aligned}
\mathcal{L}_R^{\mathsf{F1}} &= -\log P_G\big(r^\star \mid X_{\mathrm{in}}\big) \\
\mathcal{L}_S^{\mathsf{F1}} &= -\sum_{i \in \mathcal{J}} \log P_G\big(q_i^\star \mid X_{\mathrm{in}}, X_{\mathrm{cc}}, r^\star, q_{<i}^\star\big) \\
\mathcal{L}^{\mathsf{F1}} &= \lambda_R \mathcal{L}_R^{\mathsf{F1}} + \lambda_S \mathcal{L}_S^{\mathsf{F1}}
\end{aligned}$$

where $\mathcal{J} = \{1, \ldots, K\}$ for retrieval-needed instances and $\mathcal{J} = \emptyset$ for no-retrieval instances.

For Format-2, we optimize retrieval triggering and generation conditioned on the verified filtered context. Let $X_{S^\star} = \mathrm{Pack}(C_{S^\star})$ denote the serialized filtered evidence.

$$\begin{aligned}
\mathcal{L}_R^{\mathsf{F2}} &= -\log P_G\big(r^\star \mid X_{\mathrm{in}}\big) \\
\mathcal{L}_Y^{\mathsf{F2}} &= -\sum_{t=1}^{T} \log P_G\big(y_t \mid y_{<t}, X_{\mathrm{in}}, X_{S^\star}, r^\star\big) \\
\mathcal{L}^{\mathsf{F2}} &= \lambda_R \mathcal{L}_R^{\mathsf{F2}} + \mathcal{L}_Y^{\mathsf{F2}}.
\end{aligned}$$

Table 1: Code completion performance in the Infilling setting.

| Model | Strategy | RepoEval | | | | | | CCLongEval | | | CCEval | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Line | | API | | Function | | Chunk | | Func | Line | |
| | | EM (M1) | ES (M2) | EM (M3) | ES (M4) | UT (M5) | ES (M6) | EM (M7) | ES (M8) | ES (M9) | EM (M10) | ES (M11) |
| **SC-Base-1B** | No-Retrieve | 43.14 | 67.39 | 38.03 | 66.81 | 21.67 | 47.29 | 30.62 | 60.54 | 47.16 | 18.72 | 42.85 |
| | Full-Retrieve | 52.27 | 73.13 | 44.18 | 69.09 | 25.61 | 55.93 | 37.49 | 64.04 | 50.72 | 22.38 | 47.26 |
| | RepoFormer | 54.71 | 76.52 | 45.73 | 72.41 | 28.46 | 57.69 | 41.93 | 70.21 | 54.37 | 25.42 | 49.18 |
| | CODEFILTER | 57.19 | 78.84 | 48.37 | 75.66 | 31.13 | 59.91 | 44.52 | 72.48 | 56.59 | 27.81 | 52.03 |
| | REPOSHAPLEY | **61.34**+4.15 | **82.78**+3.94 | **53.62**+5.25 | **79.53**+3.87 | **35.84**+4.71 | **64.39**+4.48 | **48.57**+4.05 | **77.52**+5.04 | **61.18**+4.59 | **32.26**+4.45 | **56.37**+4.34 |
| **SC-Base-3B** | No-Retrieve | 48.12 | 72.38 | 40.17 | 68.91 | 24.93 | 51.52 | 36.16 | 65.19 | 49.63 | 21.82 | 45.58 |
| | Full-Retrieve | 57.84 | 77.21 | 48.83 | 72.68 | 30.58 | 58.16 | 42.61 | 68.29 | 53.84 | 25.92 | 50.31 |
| | RepoFormer | 58.59 | 79.16 | 49.82 | 74.63 | 32.89 | 60.62 | 46.38 | 72.11 | 56.39 | 28.85 | 52.16 |
| | CODEFILTER | 61.21 | 81.09 | 51.97 | 77.62 | 35.18 | 63.26 | 49.62 | 74.58 | 58.51 | 30.84 | 55.29 |
| | REPOSHAPLEY | **64.93**+3.72 | **85.27**+4.18 | **56.38**+4.41 | **81.72**+4.10 | **39.91**+4.73 | **68.16**+4.90 | **53.52**+3.90 | **78.83**+4.25 | **62.84**+4.33 | **35.79**+4.95 | **59.41**+4.12 |
| **SC-Base-7B** | No-Retrieve | 51.62 | 75.51 | 43.83 | 71.29 | 25.62 | 52.71 | 38.91 | 66.62 | 52.84 | 23.37 | 48.01 |
| | Full-Retrieve | 58.26 | 77.79 | 50.38 | 75.01 | 32.26 | 60.21 | 44.62 | 69.19 | 55.16 | 28.51 | 52.91 |
| | RepoFormer | 59.83 | 79.26 | 51.31 | 77.46 | 35.71 | 61.19 | 46.84 | 74.16 | 57.11 | 29.62 | 55.49 |
| | CODEFILTER | 61.49 | 81.41 | 53.62 | 79.29 | 37.79 | 63.41 | 49.16 | 77.26 | 59.84 | 32.11 | 57.84 |
| | REPOSHAPLEY | **65.81**+4.32 | **86.59**+5.18 | **58.79**+5.17 | **84.11**+4.82 | **41.84**+4.05 | **68.16**+4.75 | **54.73**+5.57 | **81.29**+4.03 | **64.62**+4.78 | **36.59**+4.48 | **62.91**+5.07 |
| **Llama-7B** | No-Retrieve | 51.89 | 73.42 | 41.53 | 66.98 | 24.81 | 44.56 | 37.21 | 65.16 | 50.37 | 18.16 | 43.34 |
| | Full-Retrieve | 60.18 | 78.91 | 48.76 | 73.16 | 29.93 | 52.21 | 45.41 | 69.37 | 52.11 | 23.41 | 47.46 |
| | RepoFormer | 60.52 | 79.36 | 49.31 | 75.91 | 33.19 | 52.64 | 46.84 | 69.56 | 52.16 | 24.26 | 48.31 |
| | CODEFILTER | 63.76 | 82.31 | 52.62 | 78.54 | 32.84 | 54.49 | 50.76 | 74.91 | 54.74 | 27.16 | 51.68 |
| | REPOSHAPLEY | **68.31**+4.55 | **86.76**+4.45 | **57.28**+4.66 | **83.11**+4.57 | **37.56**+4.72 | **59.14**+4.65 | **55.21**+4.45 | **79.19**+4.28 | **59.41**+4.67 | **31.23**+4.07 | **55.24**+3.56 |
| **Llama-13B** | No-Retrieve | 53.81 | 74.84 | 42.19 | 67.96 | 26.31 | 47.14 | 41.91 | 67.46 | 52.71 | 20.97 | 45.88 |
| | Full-Retrieve | 61.41 | 79.29 | 49.81 | 77.41 | 31.69 | 54.21 | 47.36 | 70.61 | 55.24 | 25.53 | 50.20 |
| | RepoFormer | 62.19 | 81.51 | 50.46 | 79.21 | 34.39 | 54.44 | 48.96 | 71.11 | 55.41 | 27.20 | 52.20 |
| | CODEFILTER | 64.16 | 82.71 | 53.01 | 78.99 | 35.41 | 57.76 | 51.31 | 74.19 | 58.81 | 29.91 | 54.69 |
| | REPOSHAPLEY | **68.89**+4.73 | **87.11**+4.40 | **57.66**+4.65 | **83.41**+4.42 | **40.11**+4.70 | **62.39**+4.63 | **55.91**+4.60 | **78.59**+4.40 | **63.51**+4.70 | **35.05**+5.14 | **59.37**+4.68 |

Here $\mathcal{L}_R$ is implemented as the cross-entropy on the next-token prediction at the retrieval-control position (i.e., immediately after `<SFX>`$[X_s]$), rather than a separate classifier.

During training, we either (i) include both formats for each instance, or (ii) sample one format per instance with a fixed mixing ratio. The final objective is the expectation over the chosen format:

$$\mathcal{L} = \mathbb{E}_{\mathsf{F}\sim\pi}\left[\mathcal{L}^{\mathsf{F}}\right], \quad \mathsf{F} \in \{\mathsf{F1}, \mathsf{F2}\}.$$

**Inference.** At inference time, REPOSHAPLEY makes retrieval decisions in one autoregressive rollout. Given the in-file context, the model first predicts a retrieval-control token $r \in \mathcal{T}_R =$ <NEED>, <DONE>. If $r =$ <DONE>, it directly performs FIM decoding to generate the completion.

If $r =$ `<NEED>`, we retrieve $K$ cross-file candidates $X_{\text{cc}} = cc_1, \ldots, cc_K$ and serialize them as $\text{Pack}(X_{\text{cc}})$. Conditioned on this packed block, the model outputs a length-$K$ selection sequence under `<SELECT>`, where $(q_1, \ldots, q_K) \in \mathcal{T}_S^K$ and $\mathcal{T}_S =$ `<KEEP>`, `<DROP>`. We then keep only chunks with $q_i =$ `<KEEP>`, append them to the prompt, and generate $\hat{Y}$ via FIM decoding after emitting `<MID>`. Alg. 1 provides the full procedure. We use $\oplus$ to denote token sequence concatenation.

## 4 Experiments

### 4.1 Experimental Setup

**Dataset.** We curate 290k Python repositories from The Stack (Kocetkov et al., 2023) after strict quality filtering (LOC constraints, AST parsing, and deduplication; Appendix A.2). Following (Wu et al., 2024), we sample 7.5k repositories to construct 50k labeled instances: for each instance, we retrieve top-10 cross-file chunks using Jaccard similarity (Jaccard, 1912) and assign supervision derived from ChunkShapley. During data labeling, we discard instances whose verification-selected coalition $S^\star$ fails to reach a minimum completion quality, i.e., $\text{ES}(\hat{Y}_{S^\star}, Y) < \tau_{\text{es}}$, to ensure supervision reliability. We split repositories into disjoint 95%/5% train/validation pools before instance construction, so validation instances come from repositories unseen during training.

**Models and Training.** We fine-tune StarCoder-Base (SCB-1B/3B/7B) (Li et al., 2023) and CodeLlama (Llama-7B/13B) (Roziere et al., 2023) for 2 epochs using a learning rate of $2 \times 10^{-5}$ with linear decay and 5% warm-up. We set $\lambda_R = \lambda_S = 2.0$, max sequence length to 4096. With a global batch size of 512 on 8 NVIDIA H100 (80GB), training takes on average 2.2/6.5/15.4 hours for SCB-1B/3B/7B and 15.8/28.6 hours for Llama-7B/13B, respectively. Details are shown in Appendix B.

**Benchmarks and Metrics.** We evaluate on three repository-level code completion benchmarks: RepoEval (Zhang et al., 2023), CrossCodeEval (Ding et al., 2023), and CrossCodeLongEval (Wu et al., 2024). Together they cover line, API, chunk, and function-level completion tasks under realistic

Table 2: Component Ablation of REPOSHAPLEY on RepoEval. We investigate components in (A) Labeling and (B) Distillation. Baseline is SC-Base-1B.

| Method / Variant | RepoEval-Line | | RepoEval-API | | Latency |
|---|---|---|---|---|---|
| | EM | ES | EM | ES | (ms/req) |
| RepoFormer | 54.71 | 76.52 | 45.73 | 72.41 | 661 |
| CODEFILTER | 57.19 | 78.84 | 48.37 | 75.66 | 947 |
| **REPOSHAPLEY** | **61.34** | **82.78** | **53.62** | **79.53** | 1053 |
| ***A. Labeling Strategy*** | | | | | |
| 1. w/o Post-verification | 38.50 | 54.44 | 36.15 | 55.81 | – |
| 2. $\Delta$-only labeling | 58.45 | 77.12 | 48.46 | 75.26 | – |
| 3. Linear Surrogate | 59.92 | 76.41 | 50.73 | 77.09 | – |
| 4. Uniform Weights | 60.18 | 80.97 | 51.82 | 77.38 | – |
| ***B. Distillation*** | | | | | |
| 5. Format-1 only | 5.56 | 8.27 | 2.34 | 5.66 | 523 |
| 6. Format-2 only | 59.88 | 79.11 | 52.12 | 77.49 | 830 |
| 7. No Trigger | 61.26 | 81.33 | 52.15 | 78.81 | 1462 |

cross-file dependencies. We consider two prompting settings: **Infilling** (FIM with $X_{\text{in}} = (X_p, X_s)$) and **Left-to-right** (prefix-only with $X_{\text{in}} = X_p$). Following prior work (Wu et al., 2024), we report Exact Match (EM) and Edit Similarity (ES) for non-function tasks, and unit-test pass rate (UT) for function tasks. Metric formulations are shown in Appendix A.1.

**Baselines.** We compare REPOSHAPLEY against: (1) **No-Retrieve** (in-file only); (2) **Full-Retrieve** (Zhang et al., 2023) (top-10 sparse retrieval); (3) **RepoFormer** (Wu et al., 2024) (selective retrieval); and (4) **CODEFILTER** (Li et al., 2025) (likelihood-based filtering). CODEFILTER serves as the primary baseline to highlight the benefit of interaction-aware supervision.

## 4.2 Main Results

Tables 1 and 4 show that REPOSHAPLEY consistently improves repository-level infilling across benchmarks and backbones, validating our core hypothesis that supervision derived from evidence coalitions better reflects interaction-heavy retrieval.

First, interaction-blind filtering remains brittle. While adaptive controllers generally outperform Full-Retrieve, methods trained from per-chunk labels (CODEFILTER) can still overfit to isolated similarity and fail to account for complementarity and conflict that only appear when multiple chunks are concatenated. This gap is most visible on harder settings that require resolving non-local dependencies such as Function, where selecting the right combination of evidence matters more.

Second, coalition-aware supervision yields the strongest gains on difficult tasks. On SC-Base-7B, REPOSHAPLEY improves RepoEval API from 53.62/79.29 to **58.79/84.11** (EM/ES) and raises RepoEval Function unit-test pass rate from 37.79 to **41.84**, outperforming CODEFILTER by clear margins. These improvements align with our motivation: modeling evidence interactions helps retain complementary context while suppressing conflicting or redundant chunks.

Finally, the gains generalize beyond RepoEval. REPOSHAPLEY also delivers consistent improvements on long-context and chunk-level benchmarks, on CCLongEval Chunk it improves ES from 77.26 to **81.29** on SC-Base-7B and from 74.19 to **78.59** on Llama-13B, indicating that the learned keep and drop policy transfers across evaluation granularities and context regimes.

Although REPOSHAPLEY introduces additional computation during offline labeling, its inference-time overhead remains modest. As shown in Table 2, REPOSHAPLEY runs at 1053 ms/req, which is comparable to CODEFILTER (947 ms) and within the same runtime scale as RepoFormer (661 ms).

Table 3: Retriever comparison on SC-Base-7B (RepoEval-Line, Infilling).

| Method | Jaccard | | UniXcoder | |
|---|---|---|---|---|
| | EM | ES | EM | ES |
| Full-Retrieve | 58.26 | 77.79 | 62.71 | 80.45 |
| REPOSHAPLEY | **65.81** | **86.59** | **68.30** | **88.14** |

**Robustness to retriever choice.** To verify that gains are not tied to sparse retrieval, we replace Jaccard with UniXcoder (Guo et al., 2022) as a dense retriever on SC-Base-7B (RepoEval-Line). As shown in Table 3, REPOSHAPLEY improves over Full-Retrieve under both retrievers, and the absolute gains are comparable (+8.80 ES with Jaccard vs. +7.69 ES with UniXcoder), confirming that coalition-aware filtering generalizes across retrieval paradigms. Appendix C.7 further shows that REPOSHAPLEY remains effective as an external selective-RAG policy for stronger modern code LMs, and Appendix C.8 validates cross-lingual generalization on Java, C#, and TypeScript.

## 4.3 Ablation Study & Analysis

We study how each component of REPOSHAPLEY affects performance by ablating (A) the offline labeling pipeline and (B) the online distillation strategy on StarCoderBase-1B with RepoEval (Table 2). **Coalition-aware labeling matters.** Ablations in

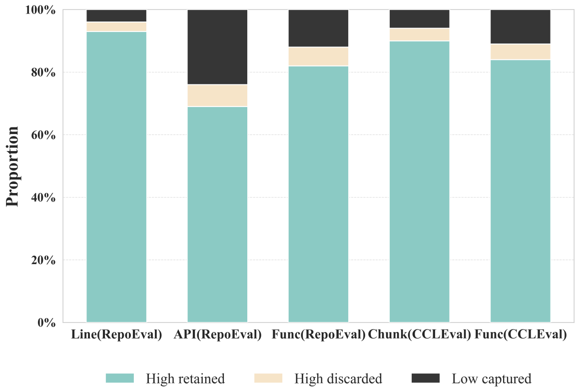


Figure 4: Breakdown of chunk selection decisions across benchmarks. **High retained**: consensus. **High discarded** and **Low captured**: corrections by RepoShapley.

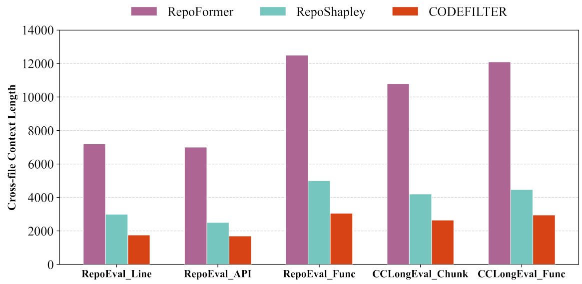


Figure 5: Retained cross-file context lengths. RepoFormer keeps the most tokens, CODEFILTER is most aggressive, and REPOSHAPLEY balances pruning with coverage.

Part A show that modeling interactions is necessary for reliable filtering. Using Shapley signs without post-verification (Row 1) causes a large drop, indicating that signed marginal effects alone are not stable under prerequisite dependencies. Replacing coalition-based attribution with single-chunk probing (Row 2) also hurts performance, suggesting that independent scores miss synergy among chunks. Simplifying the surrogate utility by removing the sigmoid (Row 3) or using uniform weights (Row 4) further degrades results, supporting our design for capturing saturation and conflict effects.

**Distillation and triggering improve inference.** Part B shows that the training design is essential. Training with selection-only signals (Row 5) fails to produce usable code, while generation-only training (Row 6) lags behind the full model due to residual noise. Removing the trigger (Row 7) yields similar accuracy but increases latency, confirming that the learned trigger reduces unnecessary retrieval while maintaining generation quality.

**Sensitivity to surrogate scale $\beta$.** We vary $\beta$ on RepoEval-Line (SC-Base-1B) and find that ES is robust for $\beta \in [0.5, 2.0]$, peaking at $\beta$=1.0 (82.78%) and degrading at both extremes (78.5% at $\beta$=0.1, 80.4% at $\beta$=5.0). Small $\beta$ under-separates positive and negative chunks, while large $\beta$ saturates the sigmoid. We adopt $\beta$=1.0 as the default.

**Selection behavior analysis.** To understand what RepoShapley actually keeps and discards, we partition retrieved candidates by combining retriever score (high vs. low) with the keep/drop decision. As shown in Figure 4, on relatively local tasks (RepoEval Line, CCLEval Chunk), RepoShapley agrees with the retriever on most chunks (90% high retained) while selectively correcting a small fraction. As task complexity increases, the corrections become more substantial: on RepoEval API, 7% of high-score chunks are discarded and 24% of kept chunks come from the low-score tail, confirming that interaction-aware filtering recovers individually under-ranked but coalition-critical evidence.

**Context length distribution.** Figure 5 compares the number of cross-file tokens retained after filtering. RepoFormer retains the longest contexts (exceeding 12k tokens on function tasks), carrying substantial redundancy. CODEFILTER prunes most aggressively but may remove weak yet necessary dependencies. RepoShapley lies between the two, shortening contexts relative to RepoFormer while keeping complementary chunks that appear low-signal in isolation, achieving a favorable trade-off between token overhead and semantic coverage.

## 5 Conclusion

In this work, we study repository-level retrieval control under strong evidence interaction effects. We propose ChunkShapley, an offline labeling pipeline that estimates signed single-chunk effects, uses a structured surrogate game to capture saturation and conflict, and applies bounded post-verification to align selected coalitions with the generator's decoding behavior. The resulting Shapley-style labels are then distilled into REPOSHAPLEY, which performs retrieval triggering and chunk selection through discrete control tokens at inference time. Across benchmarks, backbones, retrievers, and prompting settings, REPOSHAPLEY consistently improves completion quality while reducing harmful or unnecessary context, showing that coalition-aware supervision is an effective foundation for repository-level code completion.

## Limitations

Our method has several limitations. First, the surrogate utility is built from single-chunk probes and may miss higher-order interactions where multiple individually weak chunks become useful only jointly; bounded post-verification mitigates but cannot fully guarantee recovery. Second, offline labeling enumerates $2^K$ subsets under the surrogate game ($K{=}10$), so the cost grows exponentially and limits scalability to larger retrieval budgets. Third, our framework uses a fixed chunking granularity, which affects the player set: finer granularity increases cost, while coarser granularity may mask intra-chunk interactions. Fourth, the verification stage is tied to greedy decoding with ES/EM; coalition choices may vary under different decoding strategies or objectives. Finally, the labeling pipeline requires multiple teacher-forced forward passes and decoding runs, increasing offline computation. Dataset licenses and code availability details are provided in Appendix E.

## Acknowledgments

This work is supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant No. 2025A1515012968, in part by the Shenzhen Science and Technology Program under Grant No. JCYJ20240813113502004, in part by the National Natural Science Foundation of China under Grant No. 62001412, in part by Shenzhen Stability Science Program 2023, in part by the Guangdong Provincial Key Laboratory of Future Networks of Intelligence (Grant No. 2022B1212010001), and in part by the Shenzhen Key Lab of Crowd Intelligence Empowered Low-Carbon Energy Network (Grant No. ZDSYS20220606100601002).

## References

Anthropic. 2025. System card addendum: Claude Opus 4.1. Technical report, Anthropic. Accessed: 2026-04-18.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D C, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. 2023. Codeplan: Repository-level coding using LLMs and planning. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Amanda Bertsch, Maor Ivgi, Emily Xiao, Uri Alon, Jonathan Berant, Matthew R Gormley, and Graham Neubig. 2025. In-context learning with long-context models: An in-depth exploration. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12119–12149.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Wei Cheng, Yuhan Wu, and Wei Hu. 2024. Dataflow-guided retrieval augmentation for repository-level code completion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7957–7977.

Yangruibo Ding, Jinjun Peng, Marcus J. Min, Gail Kaiser, Junfeng Yang, and Baishakhi Ray. 2024a. Semcoder: Training code language models with comprehensive semantics reasoning. In *Advances in Neural Information Processing Systems*, volume 37, pages 60275–60308. Curran Associates, Inc.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and 1 others. 2023. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36:46701–46723.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2024b. Cocomic: Code completion by jointly modeling in-file and cross-file context. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 3433–3445.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pages 2242–2251. PMLR.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th conference of the european chapter of the association for computational linguistics: main volume*, pages 874–880.

Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Mintong Kang, Nezihe Merve Gürel, Ning Yu, Dawn Song, and Bo Li. 2024. C-RAG: Certified generation risks for retrieval-augmented language models. In *Forty-first International Conference on Machine Learning*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*.

Denis Kocetkov, Raymond Li, Loubna Ben allal, Jia LI, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro Von Werra, and Harm de Vries. 2023. The stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, and 1 others. 2023. Starcoder: May the source be with you! *Transactions on machine learning research*.

Yanzhou Li, Shangqing Liu, Kangjie Chen, Tianwei Zhang, and Yang Liu. 2025. Impact-driven context filtering for cross-file code completion. In *Second Conference on Language Modeling*.

Haoyue Liu, Zhichao Wang, Yongxin Guo, Haoran Shou, and Xiaoying Tang. 2026. Adaptive prompt structure factorization: A framework for self-discovering and optimizing compositional prompt programs. *Preprint*, arXiv:2604.06699.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024b. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*.

Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. 2024c. Graphcoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 570–581.

Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822.

Ikhtiyor Nematov, Tarik Kalai, Elizaveta Kuzmenko, Gabriele Fugagnoli, Dimitris Sacharidis, Katja Hose, and Tomer Sagi. 2025. Source attribution in retrieval-augmented generation. *arXiv preprint arXiv:2507.04480*.

OpenAI. 2025. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/. Accessed: 2026-04-18.

Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734.

Qwen Team. 2025. Qwen2.5-Max: Exploring the intelligence of large-scale MoE model. https://qwenlm.github.io/blog/qwen2.5-max/. Accessed: 2026-04-18.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Lloyd S Shapley. 1953. A value for n-person games. *Contributions to the Theory of Games*, 2:307–317.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Disha Shrivastava, Denis Kocetkov, Harm De Vries, Dzmitry Bahdanau, and Torsten Scholak. 2023. Repofusion: Training code models to understand your repository. *arXiv preprint arXiv:2306.10998*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025. CodeRAG-bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214, Albuquerque, New Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Zhepei Wei, Wei-Lin Chen, and Yu Meng. 2025. InstructRAG: Instructing retrieval-augmented generation via self-synthesized rationales. In *The Thirteenth International Conference on Learning Representations*.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: selective retrieval for repository-level code completion. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Yingtai Xiao, Yuqing Zhu, Sirat Samyoun, Wanrong Zhang, Jiachen T Wang, and Jian Du. 2025. Tokenshapley: Token level context attribution with shapley value. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3882–3894.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*.

Zezhou Yang, Ting Peng, Cuiyun Gao, Chaozheng Wang, Hailiang Huang, and Yuetang Deng. 2025. A deep dive into retrieval-augmented generation for code completion: Experience on wechat. *arXiv preprint arXiv:2507.18515*.

Chengyuan Yao and Satoshi Fujita. 2024. Adaptive control of retrieval-augmented generation for large language models through reflective tags. *Electronics*, 13(23):4643.

Zikun Ye and Hema Yoganarasimhan. 2025. Fair document valuation in llm summaries via shapley values. *arXiv preprint arXiv:2505.23842*.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *International Conference on Learning Representations*, volume 2024, pages 29862–29883.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Sheng Zhang, Yifan Ding, Shuquan Lian, Shun Song, and Hui Li. 2025. 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*, pages 23278–23288, Suzhou, China. Association for Computational Linguistics.

# A Details of dataset construction

## A.1 Metrics Formulation

We evaluate code completion quality using Exact Match (EM), Edit Similarity (ES), and Unit Tests (UT). Let $\hat{Y}$ be the generated code and $Y$ be the ground truth:

$$
\begin{aligned}
\text{EM} &= \mathbf{1}(\hat{Y} = Y) \\
\text{ES} &= [1 - \frac{\mathcal{D}(\hat{Y}, Y)}{\max(|\hat{Y}|, |Y|)}] \times 100\% \\
\text{UT} &= \mathbf{1}(\text{Pass}(\hat{Y}))
\end{aligned}
$$

where $\mathbf{1}(\cdot)$ is the indicator function, $\mathcal{D}(\cdot)$ denotes the Levenshtein distance, and $\text{Pass}(\cdot)$ returns true if the code passes all unit tests.

## A.2 Data Collection and Preprocessing

**File-level filtering.** We begin with conservative file hygiene to reduce retrieval noise and stabilize likelihood-based labeling. We keep only `.py` files and discard files with fewer than 10 non-empty lines. To remove minified/generated blobs that distort sparse retrieval, we drop files whose maximum line length exceeds 300 characters or whose average line length exceeds 120 characters (computed after trimming trailing whitespace). We further filter out non-code payloads by requiring alphanumeric density $\geq 0.35$ (ratio of letters/digits over all characters). Finally, we exclude vendored or generated directories by path keywords, including `vendor/`, `third_party/`, `site-packages/`, `dist/`, `build/`, `.venv/`, and `migrations/`. All statistics are computed on UTF-8 decoded text (with a permissive fallback that drops undecodable bytes).

**Repository-level filtering.** We retain repositories with sufficient structure for cross-file interactions by requiring at least 8 remaining Python files and total non-empty LOC between 300 and 50,000 after file-level filtering. To avoid duplicate-heavy projects where top-$K$ retrieval collapses to repeated copies, we estimate the near-duplicate file ratio using **SimHash**. Specifically, for each repository we compute SimHash fingerprints over a normalized UTF-8 text representation of each file (whitespace-collapsed, with trailing whitespace removed), and perform pairwise checks on up to the first 200 files (`max_files_for_dup_check`= 200). We mark two files as near-duplicates if their SimHash Hamming distance is at most 3 (`simhash_hamming_threshold`= 3). Repositories with more than 30% near-duplicate files are discarded (`max_dup_ratio`= 0.3).

We also enforce syntactic integrity by parsing a sampled subset of files with Python `ast.parse`. Concretely, we uniformly sample up to 20 files per repository (`ast_sample_k`= 20) from the remaining Python files after file-level filtering, and compute the parse success rate on this sample. Repositories with parse success rate $< 70\%$ are removed (`min_ast_parse_rate`= 0.7). For reproducibility, both the duplicate-check subsampling (when applicable) and AST sampling use a fixed random seed of 13 (`seed`= 13).

## A.3 Data labeling.

We chunk the cross-file pool and construct retrieval queries for each target span $Y$. During offline label construction, we use two candidate-retrieval variants in equal proportion: a context-only query from the in-file context and an oracle-assisted query that also includes $Y$. The oracle-assisted query is used only to retrieve candidate chunks before ChunkShapley scoring; $Y$ is never included in the model input, inference-time retrieval query, or benchmark evaluation. For each query, we retrieve the top-$K$ candidate chunks and distill coalition-aware decisions into chunk-wise <KEEP>/<DROP> labels and a retrieval-control token. Specifically, given $X_{\text{in}} = (X_p, X_s)$ and retrieved candidates $X_{\text{cc}} = (cc_1, \ldots, cc_K)$, we run our offline **ChunkShapley** pipeline to obtain a verification-selected coalition $S^\star \subseteq \{1, \ldots, K\}$. We first compute a teacher-forced baseline log-likelihood $\ell(\emptyset)$ and probe each chunk in isolation, $\Delta_i = \ell(\{i\}) - \ell(\emptyset)$, yielding a signed vote $y_i = \text{sign}(\Delta_i)$ and weight $\omega_i = |\Delta_i|$. We then define a lightweight surrogate game $v_{\text{sur}}(S) = \sigma(\beta \sum_{i \in S} \omega_i y_i) - \sigma(0)$, where $\sigma(\cdot)$ is the sigmoid function, and compute exact surrogate Shapley values by enumerating all $2^K$ coalitions (tractable for small $K$ and performed offline). Finally, we verify a bounded set of Shapley-proposed coalitions using the frozen generator under decoding-time constraints and select $S^\star$ that maximizes completion quality (lexicographically by ES then EM). We treat $S^\star$ as the teacher subset and assign labels: $Q(cc_i) =$ <KEEP> if $i \in S^\star$ and <DROP> otherwise.

To supervise retrieval triggering, we assign the retrieval-control token $r^\star \in \{$<NEED>, <DONE>$\}$ by comparing the completion quality with and with-

Table 4: Code completion performance in the Left-to-Right setting.

| Model | Strategy | RepoEval | | | | | | CCLongEval | | | CCEval | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Line | | API | | Function | | Chunk | | Func | Line | |
| | | EM | ES | EM | ES | UT | ES | EM | ES | ES | EM | ES |
| SC-Base-1B | No-Retrieve | 33.42 | 57.88 | 28.54 | 57.36 | 16.55 | 40.21 | 22.45 | 53.05 | 39.88 | 16.54 | 54.47 |
| | Full-Retrieve | 44.52 | 66.21 | 36.95 | 64.77 | 21.30 | 48.55 | 31.12 | 63.49 | 45.36 | 20.12 | 58.21 |
| | RepoFormer | 46.12 | 68.33 | 37.44 | 66.12 | 23.45 | 50.12 | 32.55 | 65.12 | 46.88 | 22.45 | 60.33 |
| | CODEFILTER | 48.88 | 70.15 | 39.85 | 69.11 | 24.12 | 51.55 | 34.15 | 66.88 | 48.22 | 24.88 | 62.55 |
| | **REPOSHAPLEY** | **54.21** +5.33 | **76.45** +6.30 | **45.66** +5.81 | **74.88** +5.77 | **29.85** +5.73 | **57.22** +5.67 | **40.55** +6.40 | **72.44** +5.56 | **54.12** +5.90 | **30.12** +5.24 | **68.95** +6.40 |
| SC-Base-3B | No-Retrieve | 35.82 | 60.12 | 29.55 | 58.45 | 20.05 | 38.95 | 25.44 | 53.45 | 44.82 | 18.22 | 57.51 |
| | Full-Retrieve | 50.45 | 70.88 | 40.66 | 67.89 | 26.12 | 48.66 | 36.15 | 59.88 | 45.75 | 23.45 | 62.12 |
| | RepoFormer | 50.11 | 71.95 | 41.02 | 69.88 | 27.55 | 50.12 | 36.75 | 61.95 | 47.12 | 25.66 | 63.88 |
| | CODEFILTER | 53.12 | 73.66 | 43.15 | 73.12 | 27.88 | 51.22 | 38.05 | 61.55 | 48.88 | 27.45 | 65.12 |
| | **REPOSHAPLEY** | **59.45** +6.33 | **79.11** +5.45 | **48.88** +5.73 | **79.55** +6.43 | **33.45** +5.57 | **57.88** +6.66 | **44.22** +6.17 | **67.12** +5.57 | **54.66** +5.78 | **33.15** +5.70 | **70.44** +5.32 |
| SC-Base-7B | No-Retrieve | 38.15 | 62.12 | 31.45 | 59.88 | 21.88 | 39.95 | 29.45 | 58.55 | 53.45 | 19.68 | 59.00 |
| | Full-Retrieve | 51.22 | 71.55 | 42.45 | 68.33 | 28.15 | 50.45 | 41.55 | 64.88 | 48.75 | 24.55 | 63.45 |
| | RepoFormer | 50.88 | 70.45 | 40.88 | 72.66 | 28.05 | 48.55 | 41.05 | 64.75 | 48.15 | 26.88 | 65.12 |
| | CODEFILTER | 53.95 | 74.22 | 44.55 | 72.15 | 29.05 | 52.33 | 42.12 | 66.88 | 57.88 | 28.55 | 67.55 |
| | **REPOSHAPLEY** | **59.88** +5.93 | **80.55** +6.33 | **50.12** +5.57 | **78.45** +6.30 | **34.66** +5.61 | **58.12** +5.79 | **48.45** +6.33 | **72.15** +5.27 | **63.45** +5.57 | **34.12** +5.57 | **73.22** +5.67 |
| Llama-7B | No-Retrieve | 39.55 | 64.12 | 30.88 | 60.22 | 22.45 | 42.55 | 30.12 | 58.12 | 45.45 | 20.88 | 60.12 |
| | Full-Retrieve | 52.45 | 70.88 | 43.15 | 68.75 | 26.88 | 50.12 | 41.22 | 63.88 | 52.66 | 25.44 | 64.55 |
| | RepoFormer | 51.12 | 71.45 | 40.88 | 70.88 | 28.66 | 50.05 | 39.45 | 62.95 | 50.45 | 27.12 | 65.88 |
| | CODEFILTER | 53.66 | 73.12 | 43.88 | 73.55 | 29.88 | 50.88 | 41.88 | 65.45 | 53.55 | 29.45 | 67.88 |
| | **REPOSHAPLEY** | **59.12** +5.46 | **78.66** +5.54 | **49.55** +5.67 | **79.12** +5.57 | **35.12** +5.24 | **56.45** +5.57 | **47.22** +5.34 | **71.05** +5.60 | **59.12** +5.57 | **35.66** +6.21 | **73.45** +5.57 |
| Llama-13B | No-Retrieve | 41.55 | 65.12 | 31.22 | 60.66 | 24.12 | 43.55 | 31.55 | 57.66 | 46.12 | 21.88 | 61.45 |
| | Full-Retrieve | 54.22 | 74.45 | 44.66 | 71.88 | 28.95 | 51.45 | 43.22 | 68.12 | 50.22 | 26.55 | 66.12 |
| | RepoFormer | 52.45 | 71.55 | 43.95 | 71.66 | 28.66 | 51.12 | 43.55 | 67.88 | 52.45 | 28.12 | 67.45 |
| | CODEFILTER | 55.33 | 75.12 | 45.12 | 74.95 | 30.12 | 52.45 | 44.22 | 67.95 | 57.12 | 30.88 | 69.55 |
| | **REPOSHAPLEY** | **61.12** +5.79 | **81.45** +6.33 | **51.55** +6.43 | **80.66** +5.71 | **36.45** +6.33 | **58.22** +5.77 | **50.12** +5.90 | **73.45** +5.50 | **62.88** +5.76 | **36.95** +6.07 | **75.12** +5.57 |

Table 5: Default labeling and inference settings used unless otherwise specified. ES thresholds are reported in ES points.

| Setting | Symbol | Default |
|---|---|---|
| Retrieved chunks | $K$ | 10 |
| Chunk/query window | $w$ | 512 tokens |
| Chunk stride | $s$ | 256 tokens |
| Shapley prefix count | $N_v$ | 10 |
| Verification scope | $L$ | 3 |
| Surrogate scale | $\beta$ | 1.0 |
| Minimum verified ES | $\tau_{\text{es}}$ | 50 |
| Retrieval-gain margin | $\epsilon$ | 0 |
| Trigger threshold | $t_c$ | 0.5 |
| Labeling query mix | – | 50/50 ctx./oracle |
| Inference query | – | ctx.-only |
| Train/validation split | – | repo-level 95/5 |

out cross-file evidence. If the verified coalition provides negligible gain over the in-file-only completion ($\text{ES}(\hat{Y}_{S^\star}, Y) - \text{ES}(\hat{Y}_\emptyset, Y) \leq \epsilon$), we set $r^\star =$ <DONE>; otherwise $r^\star =$ <NEED>. To ensure that retained evidence is meaningful, we filter instances by requiring the verification-selected coalition to achieve $\text{ES} \geq \tau_{\text{es}}$. Alg. 3 summarizes the cross-file labeling procedure.

### A.4 Labeling Algorithm Details

Algorithm 3 outlines the process of deriving supervision signals from raw code repositories. First, top-$K$ candidate chunks $X_{\text{cc}}$ are retrieved based on the query window $Q$. We then utilize CHUNKSHAPLEY to identify the optimal chunk subset $S^\star$ that maximizes generation quality relative to the ground truth $Y$.

The labeling logic follows three specific criteria:

1. **Quality Control:** Instances are discarded if the optimal subset's performance falls below a minimum threshold $\tau_{\text{es}}$, ensuring training data quality.

2. **Retrieval Label ($r^\star$):** We measure the performance gain of using external context ($S^\star$) versus the closed-book baseline ($\emptyset$). If the gain is negligible ($\leq \epsilon$), the retrieval label is set to <DONE>; otherwise, it is <NEED>.

3. **Selection Label ($q_i^\star$):** Individual chunks are labeled as <KEEP> if they belong to the optimal subset $S^\star$, and <DROP> otherwise.

## B Hyperparameter Optimization

We tune training hyperparameters using **StarCoderBase-1B** as a proxy model to reduce search cost. Unless otherwise specified, all other settings follow the main experimental setup (e.g., data split, prompt formats, max sequence length, and batching).

**Search space.** We conduct a grid search on the following space: learning rate $\in \{1 \times 10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}\}$, loss weight $\lambda \in \{0.2, 1.0, 2.0, 5.0\}$, training epochs $\in \{1, 2, 5\}$, and warmup steps $\in \{50, 100\}$. Here $\lambda$ is applied

**Algorithm 2: ChunkShapley**: Surrogate Shapley Attribution with Bounded Verification

**Input:** In-file context $X_{\text{in}}$; ground-truth completion $Y$; retrieved chunks $X_{\text{cc}} = (cc_1, \ldots, cc_K)$; Frozen generator $G_\theta$; surrogate scale $\beta$; verification params $(N_v, L)$.
**Output:** Verification-selected coalition $S^\star \subseteq \{1, \ldots, K\}$; decoded completion $\hat{Y}_{S^\star}$; surrogate Shapley scores $\{\phi_i\}_{i=1}^K$.

$\ell(\emptyset) \leftarrow \frac{1}{|Y|} \log p_\theta(Y \mid X_{\text{in}})$
**for** $i \leftarrow 1$ **to** $K$ **do**
  $\ell(\{i\}) \leftarrow \frac{1}{|Y|} \log p_\theta(Y \mid X_{\text{in}}, \{cc_i\})$
  $\Delta_i \leftarrow \ell(\{i\}) - \ell(\emptyset)$
  $y_i \leftarrow \text{sign}(\Delta_i)$; $\omega_i \leftarrow |\Delta_i|$
**end**
**foreach** $S \subseteq \{1, \ldots, K\}$ **do**
  $g(S) \leftarrow \sum_{j \in S} \omega_j y_j$
  $v_{\text{sur}}(S) \leftarrow \sigma(\beta\, g(S)) - \sigma(0)$
**end**
**for** $i \leftarrow 1$ **to** $K$ **do**
  $\phi_i \leftarrow 0$
  **foreach** $S \subseteq \{1, \ldots, K\} \setminus \{i\}$ **do**
    $w(S) \leftarrow \frac{|S|!\,(K-|S|-1)!}{K!}$
    $\phi_i \leftarrow \phi_i + w(S)\big(v_{\text{sur}}(S \cup \{i\}) - v_{\text{sur}}(S)\big)$
  **end**
**end**
$\pi_\phi \leftarrow \text{argsort}(\{\phi_i\}, \text{desc})$; $\pi_\Delta \leftarrow \text{argsort}(\{\Delta_i\}, \text{desc})$
$\mathcal{C} \leftarrow \text{BuildPool}(\pi_\phi, \pi_\Delta; N_v, L)$
**foreach** $S \in \mathcal{C}$ **do**
  $\hat{Y}_S \leftarrow \text{Decode}(G_\theta \mid X_{\text{in}}, X_S)$
  Compute $\text{ES}(\hat{Y}_S, Y)$ and $\text{EM}(\hat{Y}_S, Y)$
**end**
$S^\star \leftarrow \arg\max_{S \in \mathcal{C}} \big(\text{ES}(\hat{Y}_S, Y),\ \text{EM}(\hat{Y}_S, Y)\big)$
**return** $S^\star$, $\hat{Y}_{S^\star}$, $\{\phi_i\}_{i=1}^K$

to the retrieval-control and selection losses, i.e., $\lambda_R = \lambda_S = \lambda$ (while the generation loss uses unit weight).

**Selection criterion.** For each configuration, we evaluate code completion performance on the validation split using the same metrics as in the main experiments. We select the best hyperparameters by maximizing the validation completion quality (with ES as the primary criterion and EM as a tie-breaker).

**Final configuration.** The selected hyperparameters are: learning rate $2 \times 10^{-5}$, $\lambda_R = \lambda_S = 2.0$, epochs $= 2$, warmup steps $= 50$. We reuse this configuration for all backbones in our experiments for consistency.

**Algorithm 3:** RepoShapley Cross-file Labeling via ChunkShapley

**Input:** Repository cross-file pool $X_{\text{out}}$; in-file context $X_{\text{in}} = (X_p, X_s)$; target span $Y$; Retriever $R$; frozen generator $G$; chunk window $w$; stride $s$; retrieve budget $K$; verification params $(N_v, L)$ (as in Alg. 2); thresholds $\tau_{\text{es}}$ and $\epsilon$; query mode $m \in \{\text{ctx}, \text{oracle}\}$.
**Output:** Labeled instance: retrieval label $r^\star \in \{$<NEED>, <DONE>$\}$ and selection labels $(q_1^\star, \ldots, q_K^\star)$ with $q_i^\star \in \{$<KEEP>, <DROP>$\}$

$Q_{\text{ctx}} \leftarrow \text{ContextWindow}(X_{\text{in}}; w)$
**if** $m = \text{oracle}$ **then**
  $Q \leftarrow Q_{\text{ctx}} \oplus Y$  `// labeling only`
**else**
  $Q \leftarrow Q_{\text{ctx}}$  `// inference matched`
**end**
$\widetilde{X}_{\text{out}} \leftarrow \text{chunkize}(X_{\text{out}}; w, s)$
$X_{\text{cc}} \leftarrow R(Q, \widetilde{X}_{\text{out}})[1{:}K]$
$(S^\star, \hat{Y}_{S^\star}, \{\phi_i\}_{i=1}^K) \leftarrow$ **ChunkShapley**$(X_{\text{in}}, Y, X_{\text{cc}}, G;\ N_v, L)$
$\hat{Y}_\emptyset \leftarrow G(X_{\text{in}})$
**if** $\text{ES}(\hat{Y}_{S^\star}, Y) < \tau_{es}$ **then**
  **return discard instance**
**end**
**if** $\text{ES}(\hat{Y}_{S^\star}, Y) - \text{ES}(\hat{Y}_\emptyset, Y) \le \epsilon$ **then**
  $r^\star \leftarrow$ <DONE>
**else**
  $r^\star \leftarrow$ <NEED>
**end**
**for** $i \leftarrow 1$ **to** $K$ **do**
  **if** $i \in S^\star$ **then**
    $q_i^\star \leftarrow$ <KEEP>
  **else**
    $q_i^\star \leftarrow$ <DROP>
  **end**
**end**
**return** $r^\star, (q_1^\star, \ldots, q_K^\star)$

# C Detailed Experiments

## C.1 Ablation on Verification Scope ($L$)

Table 7 shows that expanding the verification scope from $L = 0$ to $L = 3$ brings the largest gains: moving beyond prefix-only verification substantially improves both EM and ES, and performance increases steadily up to the default $L = 3$. In contrast, further enlarging the scope ($L > 3$) yields only marginal improvements, despite a rapidly growing candidate pool and offline labeling cost. Therefore, we adopt $L = 3$ as a practical default that captures most of the benefit of combinatorial probing.

## C.2 Oracle Analysis

To validate the theoretical superiority of Shapley-based valuation over independent likelihood probing (as used in CODEFILTER), we conducted an Or-

Table 6: Hyperparameter search space and selected values (tuned on StarCoderBase-1B).

| Hyperparameter | Search space | Selected |
|---|---|---|
| Learning rate | {1e−5, 2e−5, 5e−5} | 2e−5 |
| $\lambda$ ($\lambda_R = \lambda_S$) | {0.2, 1.0, 2.0, 5.0} | 2.0 |
| Epochs | {1, 2, 5} | 2 |
| Warmup steps | {50, 100} | 50 |

acle study. We calculated the best possible Edit Similarity (ES) achievable if the model perfectly selected chunks according to the respective valuation methods (selecting top-$K$ chunks with score $> 0$).

As shown in Table 8, the Shapley-based oracle outperforms the CODEFILTER oracle by **10.45** percentage points. This confirms that modelling chunk interactions, like synergy and conflict, is critical for repository-level code completion, as independent probing fails to identify chunks that are only useful when combined, for instance interface definitions and implementations.

### C.3 Sensitivity Analysis on Retrieval Budget $K$

Table 9 investigates the trade-off between completion performance and inference latency by varying the retrieval budget $K$ (i.e., the number of candidate chunks processed by ChunkShapley) on SC-Base-1B. Increasing $K$ expands the search space for complementary evidence, potentially capturing more synergistic interactions. However, since our method involves exact Shapley estimation via subset enumeration, the computational cost grows exponentially with $K$. Specifically, a small budget ($K = 7$) yields low latency but fails to retrieve sufficient complementary pairs, resulting in suboptimal accuracy (52.16% EM). Conversely, increasing $K$ beyond 10 yields *diminishing returns*; for instance, expanding to $K = 13$ marginally improves ES by 0.51% but causes latency to explode to over 3.5 seconds due to the combinatorial complexity of the surrogate game ($2^{13}$ subsets), rendering it impractical. Crucially, $K = 10$ achieves the optimal balance; we adopt it as the default setting to balance interaction coverage with inference efficiency.

### C.4 Ablation Study on Coalition Utility Functions

In **ChunkShapley**, the choice of the characteristic function $v(S)$ is critical, as it defines the "value" distributed among retrieved chunks. We hypothesize that while task-specific metrics (like Exact Match) align perfectly with the final objective, they provide sparse and noisy signals for attribution. To validate the effectiveness of our Log-likelihood-based utility, we conduct an ablation study comparing it against task-metric-based utilities.

**Experimental Setup.** We compare three definitions of coalition utility $v(S)$:

**Log-likelihood Utility (Ours):** We use the normalized token-level log-probability gain under teacher forcing. This provides a continuous, dense signal reflecting the model's confidence:

$$v_{\log}(S) = \frac{1}{|Y|}\sum_{t=1}^{|Y|}\Big(\log p_\theta(y_t \mid y_{<t}, X_{\text{in}}, X_S) - \log p_\theta(y_t \mid y_{<t}, X_{\text{in}}, \emptyset)\Big).$$

**Exact Match (EM) Utility:** We define utility as the binary gain in obtaining a perfect prediction. This signal is discrete ($\in \{-1, 0, 1\}$) and highly sparse:

$$v_{\text{EM}}(S) = \mathbf{1}[\text{EM}(\hat{Y}_S, Y) = 1] - \mathbf{1}[\text{EM}(\hat{Y}_\emptyset, Y) = 1].$$

where $\hat{Y}_S$ is the greedy decoding result given context $S$.

**Edit Similarity (ES) Utility:** We define utility based on the improvement in surface-level similarity. While continuous (0-100), ES is derived from discrete decoding steps and is non-differentiable:

$$v_{\text{ES}}(S) = \text{ES}(\hat{Y}_S, Y) - \text{ES}(\hat{Y}_\emptyset, Y) \tag{1}$$

For all variants, we compute the exact Shapley values using the respective $v(S)$, select the optimal subset $S^\star$ using the verification strategy, and train the corresponding REPOSHAPLEY model.

**Results and Analysis.** As shown in Table 11, using log-likelihood as the utility function yields the best performance. We attribute the inferiority of metric-based utilities ($v_{\text{EM}}, v_{\text{ES}}$) to the **sparsity and high variance** of the signal. In code completion, a chunk might significantly improve the model's understanding (providing the correct variable type) without immediately flipping the final prediction to an exact match. $v_{\log}$ captures this "partial credit," whereas $v_{\text{EM}}$ assigns zero value, leading to false negatives in attribution. Furthermore, generation-based metrics are sensitive to decoding

Table 7: **Ablation on Verification Scope ($L$).** Impact of the Top-$L$ range used for combinatorial probing in the post-verification stage. We report the average size of the candidate pool $|\mathcal{C}|$, offline labeling latency per instance, and performance on SC-Base-1B.

| **Top-$L$** | **Labeling Cost** | | **Performance** | |
|---|---|---|---|---|
| | **Avg. Pool Size** ($|\mathcal{C}|$) | **Train Time per Sample** (s) | **EM** (%) | **ES** (%) |
| 0 (Prefix Only) | 8.41 | 33 | 45.15 | 70.86 |
| 1 | 10.78 | 94 | 57.73 | 76.27 |
| 2 | 12.56 | 187 | 59.15 | 78.40 |
| **3**(Default) | 18.29 | 348 | 61.34 | 82.78 |
| 4 | 25.07 | 671 | 61.70 | 83.22 |
| 5 | 35.42 | 869 | 61.94 | 83.24 |
| 7 | 68.94 | 1528 | 62.13 | 83.59 |
| 10 (All) | 225 | 6823 | - | - |

Table 8: **Oracle Performance Comparison.** We report the mean Best ES score achievable by selecting contexts based on oracle labels. REPOSHAPLEY (Oracle) demonstrates a significantly higher theoretical upper bound.

| **Method** | **Oracle Best ES (%)** |
|---|---|
| Full-Retrieve | 71.52 |
| CODEFILTER (Oracle) | 85.23 |
| REPOSHAPLEY (ORACLE) | **95.68** |

Table 9: Sensitivity analysis of the retrieval budget ($K$) on SC-Base-1B in the Infilling setting. We report Line Completion accuracy (EM/ES) and average inference latency. $K = 10$ achieves the best trade-off between context coverage and computational cost.

| **Retrieval Size** | **RepoEval-Line** | | **Efficiency** |
|---|---|---|---|
| | **EM** | **ES** | **Latency (ms)** |
| 7 | 52.16 | 70.44 | 513 |
| 9 | 58.33 | 75.12 | 833 |
| **10 (Ours)** | **61.34** | **82.78** | **1053** |
| 11 | 61.37 | 82.40 | 1924 |
| 13 | 61.88 | 81.62 | 3539 |
| 20 | 61.76 | 79.92 | 18825 |

dynamics, where a small change in context might drastically alter the greedy path, causing $v_{\text{metric}}(S)$ to fluctuate wildly. In contrast, teacher-forced log-probabilities provide a smoother and more robust estimation of marginal contribution.

### C.5 Filtering effectiveness via selective drop and counterfactual inverse.

Following CODEFILTER (Li et al., 2025), we study whether attribution signals can reliably separate helpful from harmful retrieved context on RepoEval under the same Jaccard-based retriever. Table 12 reports two complementary interventions: **Selective (Drop)**, which removes chunks labeled as `<DROP>` and keeps the remaining context, and **Inverse (Keep)**, which retains only the dropped chunks as a counterfactual diagnostic.

REPOSHAPLEY gains markedly from selective filtering, improving both EM and ES compared to the CODEFILTER counterpart. In contrast, the inverse setting substantially degrades performance for both methods, confirming that the dropped chunks are predominantly low-utility (e.g., redundant or misleading) rather than accidentally filtered-out evidence. Notably, Figure 6 shows that CODEFILTER's decisions are prone to brittle single-chunk thresholds: when individual signals are weak, it tends to label very few chunks as positive, effectively collapsing the available context. REPOSHAPLEY instead maintains a stable selection set, consistent with interaction-aware supervision that removes toxic context while preserving the evidence required for accurate repository-level completion.

### C.6 Interaction-Aware Proposal.

We examine whether gains stem solely from verification by testing **Delta+Verify**, which replaces Shapley candidates with single-chunk rankings ($\Delta_i$) under the same verifier. Table 13 shows that REPOSHAPLEY achieves substantially higher verified ES. This confirms that $\Delta$ scores miss **synergistic chunks** (weak in isolation), creating a hard performance ceiling. In contrast, Shapley effectively

Table 10: Multilingual evaluation on CrossCodeEval (Line completion, ES / ID F1).

| Model | Strategy | Python | | Java | | C# | | TypeScript | |
|---|---|---|---|---|---|---|---|---|---|
| | | ES | F1 | ES | F1 | ES | F1 | ES | F1 |
| SC-Base-7B | Full-Retrieve | 52.74 | 43.42 | 58.78 | 49.53 | 63.72 | 55.64 | 51.28 | 42.29 |
| | **RepoShapley-M** | **62.91** | **53.30** | **66.20** | **58.29** | **71.83** | **64.23** | **60.23** | **51.63** |
| Llama-13B | Full-Retrieve | 50.20 | 42.27 | 60.23 | 50.44 | 62.80 | 55.86 | 48.06 | 41.24 |
| | **RepoShapley-M** | **59.37** | **51.39** | **67.11** | **57.17** | **70.95** | **63.20** | **56.98** | **50.80** |

Table 11: **Ablation study on Utility Functions.** We report the performance of **RepoShapley** when trained with Shapley labels derived from different utility definitions. *Log-likelihood* (Ours) significantly outperforms metric-based utilities due to the density and stability of the teacher-forcing signal.

| Utility Function | Signal Type | EM | ES |
|---|---|---|---|
| w/ $v_{\text{EM}}$ | Binary | 58.12 | 77.45 |
| w/ $v_{\text{ES}}$ | Discrete-Step | 59.03 | 78.10 |
| **w/ $v_{\text{log}}$ (Ours)** | **Continuous** | **60.50** | **79.07** |

Table 12: Impact of the filtering policy based on different attribution signals. **Selective** denotes removing chunks labeled as `<DROP>`, while **Inverse** retains only those chunks (to verify the toxicity of dropped content).

| Strategy | CODEFILTER | | RepoShapley | |
|---|---|---|---|---|
| | **EM** | **ES** | **EM** | **ES** |
| **Selective** (Drop) | 49.31 | 57.50 | **54.79**↑ | **78.62**↑ |
| Inverse (Keep) | 33.17 | 40.26 | 34.72 | 41.15 |

captures these high-potential interactive subsets, providing the verifier with a superior candidate pool.

### C.7 Strong Generation Models with RepoShapley Policy in Infilling

Table 14 reports results when we pair RepoShapley with a wide range of state-of-the-art code LMs under the infilling setting on RepoEval-Line and RepoEval-API. For each backbone, we compare a standard *Full-Retrieve* strategy against using RepoShapley as a selective RAG policy while keeping the same generation model.

Across all backbones, RepoShapley consistently improves both EM and ES on both benchmarks, indicating that coalition-aware evidence filtering is complementary to model scaling and re-

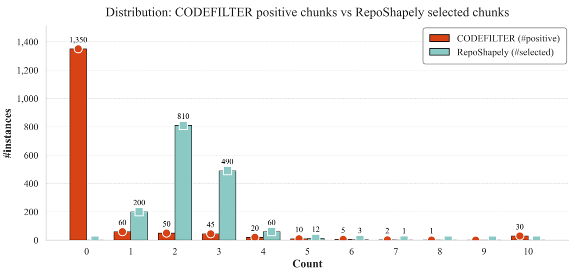


Figure 6: Distribution: `CODEFILTER` positive chunks vs. RepoShapley selected chunks

Table 13: **Impact of Proposal Mechanism.** Comparison between using Single-Chunk $\Delta$ vs. Coalition Shapley $\phi$ to generate candidates for the verification step.

| Proposal Method | Metric | Verified ES (%) |
|---|---|---|
| Delta ($\Delta$) + Verify | ES | 74.21 |
| **RepoShapley ($\phi$) + Verify** | **ES** | **82.78** |

mains effective for both open-source and closed-source generators. For closed-source models, we use the official APIs for generation, while RepoShapley is applied as an external policy to decide whether to retrieve and which chunks to keep.

### C.8 Multilingual Evaluation on CrossCodeEval

To evaluate cross-lingual generalization, we extend the labeling pipeline to Java, C#, and TypeScript, and perform multilingual mixture fine-tuning to train a single multilingual RepoShapley controller (denoted RepoShapley-M). We evaluate on CrossCodeEval (Ding et al., 2023) using StarCoderBase-7B and CodeLlama-13B. As shown in Table 10, RepoShapley-M consistently outperforms Full-Retrieve across all four languages, with ES gains ranging from +7.4 to +10.2 points. This confirms that coalition-aware filtering is not restricted to Python and transfers effectively across

Table 14: Accuracy of modern code LMs as the generation model and with REPOSHAPLEY as the policy model for selective RAG in Infilling setting. We compare DeepSeek (Guo et al., 2025), Qwen2.5-Max (Qwen Team, 2025), GLM (Zeng et al., 2025), GPT-5 (OpenAI, 2025), and Claude Opus 4.1 (Anthropic, 2025). **For closed-source models, we use the official APIs for generation.**

| **Model** | **Strategy** | **RepoEval-Line** | | **RepoEval-API** | |
|---|---|---|---|---|---|
| | | **EM** | **ES** | **EM** | **ES** |
| StarCoderBase-7B | Full-Retrieve | 58.26 | 77.79 | 50.38 | 75.01 |
| | **RepoShapley** | **65.81** | **86.59** | **58.79** | **84.11** |
| CodeLlama-13B | Full-Retrieve | 61.41 | 79.29 | 49.81 | 77.41 |
| | **RepoShapley** | **68.89** | **87.11** | **57.66** | **83.41** |
| DeepSeek-R1 | Full-Retrieve | 62.94 | 80.09 | 51.61 | 80.93 |
| | **RepoShapley** | **68.79** | **87.92** | **58.96** | **84.07** |
| Qwen2.5-Max | Full-Retrieve | 63.38 | 81.91 | 54.72 | 81.39 |
| | **RepoShapley** | **69.51** | **88.17** | **59.14** | **84.58** |
| GLM-4.5 | Full-Retrieve | 65.93 | 83.04 | 55.27 | 83.21 |
| | **RepoShapley** | **69.96** | **88.49** | **60.32** | **84.94** |
| GPT-5 | Full-Retrieve | 66.51 | 83.73 | 55.94 | 83.82 |
| | **RepoShapley** | **70.37** | **89.08** | **60.19** | **84.41** |
| Claude Opus 4.1 | Full-Retrieve | 68.23 | 84.38 | 57.56 | 85.61 |
| | **RepoShapley** | **71.09** | **89.13** | **61.74** | **85.92** |

languages.

### C.9 Latency-Accuracy Trade-off Analysis

Following RepoFormer (Wu et al., 2024), we visualize the latency-accuracy trade-off to evaluate the efficiency of REPOSHAPLEY across StarCoderBase-1B, 3B, and 7B as shown in Figure 7-9. By varying the retrieval triggering threshold $t_c$ during inference, we control the model's sensitivity to external evidence.

The results demonstrate that REPOSHAPLEY establishes a superior Pareto frontier compared to static retrieval strategies. We find that our model can improve accuracy while also reducing latency by skipping retrieval when the in-file context is already sufficient, and focusing retrieval on harder cases that truly need cross-file information. Consistent with prior observations, this efficiency gain is particularly pronounced in Line and API completion tasks, where avoiding the overhead of unnecessary retrieval significantly lowers average latency without compromising generation quality.

## D Case Study

In this section, we present a case study to illustrate how REPOSHAPLEY performs interaction-aware chunk selection for repository-level code completion in the FIM setting. The target file defines utilities for extracting event start/end markers from log search results and computing event durations. In this instance, the missing span lies inside `LogEventStats.run`: after parsing each end-marker timestamp `end` from `end_tag` results, the code should immediately register it into `EventCollection` via `add_event_end`. After registering each end marker, the routine proceeds to handle start markers and finally calls `calculate_event_deltas`. The repository contains many timestamp-related helpers; however, most retrieved evidence is only partially relevant or unrelated (e.g., YAML formatting, tests, Ceph helpers). Naively appending all retrieved contexts can distract the model into rewriting timestamp parsing rather than emitting the required event-collection logic.

**Instance (FIM).** Given the in-file prefix and suffix (Figure 10), the model must generate the missing span at <MID>. Concretely, the correct completion should insert an `add_event_end` call that uses the current `result`'s `event_id` and the parsed end timestamp. For compact presentation, Figure 10 splits the code across two columns. Therefore, the <SFX> panel shows the subsequent lines after the insertion point (not necessarily the immediate next line in the source file), while preserving the original indentation and control flow.

**Retrieved top-10 cross-file chunks.** We retrieve the top-10 candidates $\{c_1, \ldots, c_{10}\}$ from other files in the same repository (Figure 11). Most candidates are either unrelated or only partially relevant. Importantly, the most helpful timestamp utilities are split across multiple chunks ($c_1, c_8, c_9$), so that no single chunk alone fully specifies the needed behavior, making the evidence *interaction-heavy*.

**Why the kept coalition matters.** Chunks $c_1$, $c_8$, and $c_9$ form a coherent timestamp-handling subroutine. $c_1$ and $c_8$ provide compatible `datetime` parsing formats, and $c_9$ implements temporal filtering logic used by the surrounding utilities. Full-Retrieve is distracted by irrelevant utilities (e.g., $c_2, c_3$) and hallucinates redundant timestamp-parsing logic inside `run`, instead of emitting the required `add_event_end` registration.

**Generation comparison.** Figure 12 compares the generations. REPOSHAPLEY correctly inserts the `add_event_end` registration and then follows the existing start-marker logic, whereas Full-Retrieve is distracted and produces redundant parsing code.

**Failure modes.** While REPOSHAPLEY improves overall performance, we observe three recurring failure patterns. First, when all retrieved chunks are low-quality (e.g., the repository lacks relevant cross-file context), the coalition game has no good coalition to select, and the model may still keep noisy chunks rather than abstaining entirely. Second, when chunk interactions are highly non-monotone—for example, three chunks that are individually harmful but jointly beneficial—the surrogate game's logistic form may underestimate the coalition value, causing the verifier to miss the optimal subset. Third, in cases where the ground-truth completion depends on implicit project conventions not captured in any retrieved chunk, even a perfect coalition cannot help, and the model may hallucinate plausible but incorrect code. We observed such failures in approximately 8% of RepoEval instances where REPOSHAPLEY underperformed Full-Retrieve.

## E Dataset Licenses and Code Availability

**Training data.** Our training data is derived from the permissively licensed source-code subset of The Stack (Kocetkov et al., 2023). We follow the opt-out provisions specified by the dataset authors and do not redistribute raw source files.

**Evaluation benchmarks.** RepoEval (Zhang et al., 2023) is released under the MIT License. CrossCodeEval (Ding et al., 2023) is released under the Apache 2.0 License. CrossCodeLongEval (Wu et al., 2024) follows the same license as RepoEval.

**Models.** StarCoderBase (Li et al., 2023) is released under the BigCode OpenRAIL-M License. CodeLlama (Roziere et al., 2023) is released under the Llama 2 Community License.

**Code.** Our code is publicly available at https://github.com/yuhuo03/RepoShapley under the Apache-2.0 License, including data preprocessing scripts, training configurations, and evaluation pipelines for reproducing all experiments.

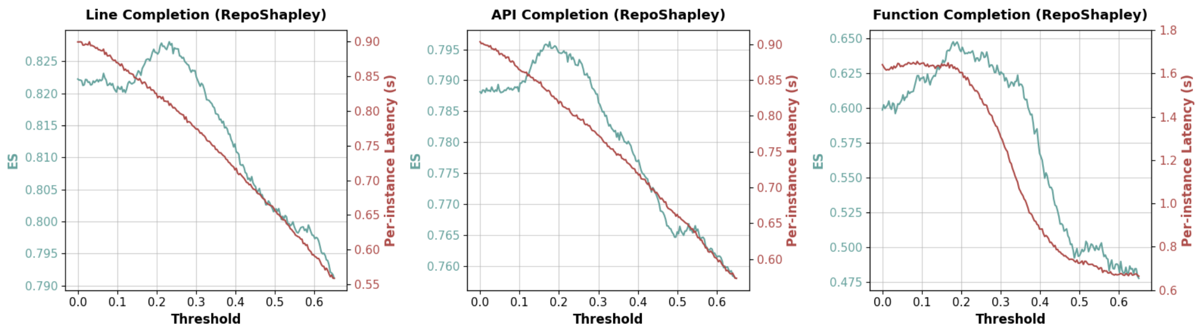


Figure 7: Latency-accuracy trade-off on SC-Base-1B.

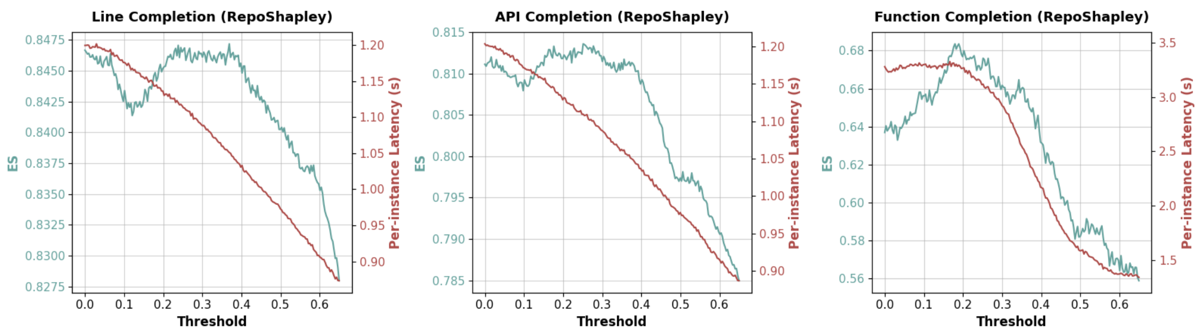


Figure 8: Latency-accuracy trade-off on SC-Base-3B.

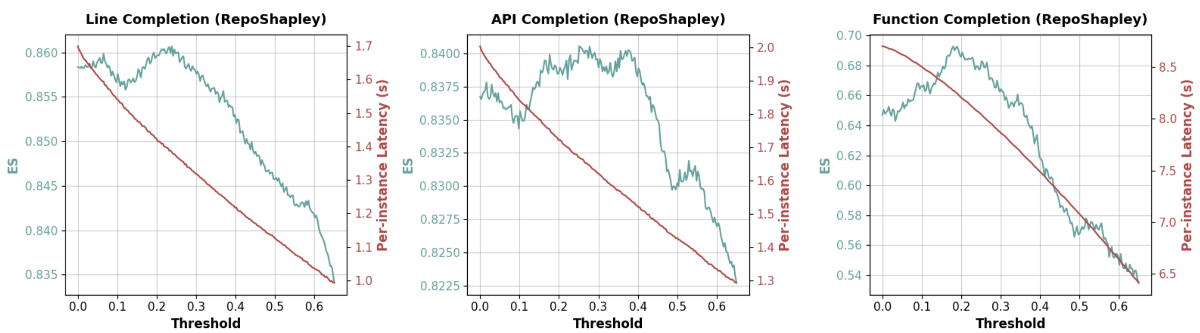


Figure 9: Latency-accuracy trade-off on SC-Base-7B.

<PFX>

```
import statistics
from datetime import datetime
class EventCollection(object):
    """Used to collect events found in
    logfiles..."""
    def __init__(self):
        self._events = {}
    def most_recent(self, items):
        return sorted(items, key=lambda e: e[
    "end"], reverse=True)[0]
    @property
    def complete_events(self):
        # ... (omitted for brevity if needed)
     ...
        return complete
    @property
    def incomplete_events(self):
        # ... (omitted for brevity) ...
        return incomplete
    def find_most_recent_start(self, event_id,
      end_ts):
        """ For a given event end marker,
    find the most recent start marker. """
        last = None
        for item in self._events[event_id].
    get("heads", []):
            start_ts = item["start"]
            if start_ts <= end_ts:
                if not last or start_ts >
    last["start"]:
                    last = item
        return last
    def add_event_end(self, event_id, end_ts):

        if event_id not in self._events:
            self._events[event_id] = {}
        if "tails" not in self._events[
    event_id]:
            self._events[event_id]["tails"] =
     [end_ts]
        else:
            self._events[event_id]["tails"].
    append(end_ts)
    def add_event_start(self, event_id,
    start_ts, metadata=None,
                        metadata_key=None):
        # ... logic to add start markers ...
        pass
    def calculate_event_deltas(self):
        # ... logic to calc deltas ...
        pass
```

<SFX>

```
class SearchResultIndices(object):
    # ... index definitions ...
    pass
class LogEventStats(object):
    """Used to identify events within logs...
    """
    def __init__(self, results,
    results_tag_prefix, custom_idxs=None):
        self.data = EventCollection()
        self.results = results
        self.results_tag_prefix =
    results_tag_prefix
        # ... init logic ...
    def run(self):
        """ Collect event start/end markers...
     """
        seq_idxs = self.log_seq_idxs
        end_tag = "{}-end".format(self.
    results_tag_prefix)
        for result in self.results.
    find_by_tag(end_tag):
            day = result.get(seq_idxs.day)
            secs = result.get(seq_idxs.secs)
            end = "{} {}".format(day, secs)
            end = datetime.strptime(end, "%Y
    -%m-%d %H:%M:%S.%f")
            start = "{} {}".format(day, secs)
            start = datetime.strptime(start,
    "%Y-%m-%d %H:%M:%S.%f")
            metadata = result.get(seq_idxs.
    metadata)
            meta_key = seq_idxs.metadata_key
            event_id = result.get(seq_idxs.
    event_id)
            self.data.add_event_start(
    event_id, start, metadata=metadata,

    metadata_key=meta_key)
        self.data.calculate_event_deltas()
    def get_top_n_events_sorted(self, max,
    reverse=True):
        # ... sorting logic ...
        return top_n_sorted
    def get_event_stats(self):
        # ... stats logic ...
        return stats
```

<NEED>

Figure 10: **FIM instance.** The missing span inserts the add_event_end registration after parsing each end marker; the subsequent start-marker handling logic continues in the suffix.

**Retrieved Candidates Pool & Selection Decisions**

<C_1> <KEEP>

```
ts_formats = ["%Y-%m-%d %H:%M:%S.%f", "%Y-%m-%d %H:%M:%S"]
# ... (timestamp parsing logic) ...
def filter_by_age(cls, results, result_age_hours):
    # ...
    current = datetime.strptime(current, "%Y-%m-%d %H:%M:%S"
      )
    # ...
```

<C_2> <DROP>

```
if message is not None:
    message = str(message).format(**fdict)
# ... message formatting utilities ...
@cached_yproperty_attr
def type(self):
    """ Name of core.issues.IssueTypeBase object ... """
    # ...
```

<C_3> <DROP>

```
data_file = os.path.join(dtmp, 'data.txt')
# ... YAML generation for testing ...
class MyEventHandler(events.YEventCheckerBase):
    def __init__(self):
        super().__init__(yaml_defs_group='mygroup'...)
```

<C_4> <DROP>

```
def _override_keys(cls): return ['raises']
# ... YPropertyOverride logic ...
def message_with_format_dict_applied(self, property=None):
    # ...
```

<C_5> <DROP>

```
s = FileSearcher()
# ... Unit tests for SequenceSearchDef ...
def test_sequence_searcher_eof(self):
    # ...
```

<C_6> <DROP>

```
# ... Duplicate or similar unit tests ...
self.assertEqual(len(sections), 2)
# ...
```

<C_7> <DROP>

```
""" Returns a dict of ceph versions info ... """
out = self.cli_cache['ceph_versions']
# ... Ceph daemon logic ...
```

<C_8> <KEEP>

```
class YPropertySearchBase(YPropertyMappedOverrideBase):
    @classmethod
    def get_datetime_from_result(cls, result):
        """ This attempts to create a datetime object... """
    ts = result.get(1)
    # ...
    ts_formats = ["%Y-%m-%d %H:%M:%S.%f", ...]
    # ...
```

<C_9> <KEEP>

```
for r in sorted(_results, key=lambda i: i[0], reverse=True):
    # ... filtering logic ...
def apply_constraints(self, results):
    # ...
    results = self.filter_by_age(results, result_age_hours)
    # ...
```

<C_10> <DROP>

```
relname = 'unknown'
# ... package version check ...
def _get_bind_interfaces(self, type):
    """ For the given config network type ... """
    # ...
```

Figure 11: **10-chunk retrieved pool and selection.** RepoShapley identifies and keeps $\{c_1, c_8, c_9\}$ (timestamp utilities split across chunks) while dropping unrelated evidence. (Content abbreviated for display).

**Ground Truth**

```
    self.data.add_event_end(
        result.get(seq_idxs.event_id),
        end)
start_tag = "{}-start".format(
    self.results_tag_prefix)
    for result in self.results.
      find_by_tag(
        start_tag):
    day = result.get(seq_idxs.day)
    secs = result.get(seq_idxs.secs)
```

**RepoShapley (Ours)**

```
    self.data.add_event_end(
        result.get(seq_idxs.event_id),
        end)
start_tag = "{}-start".format(
    self.results_tag_prefix)
    for result in self.results.
      find_by_tag(
        start_tag):
    day = result.get(seq_idxs.day)
    secs = result.get(seq_idxs.secs)
```

**Full-Retrieve Baseline**

```
    ts = "{} {}".format(day, secs)
    ts_formats = ["%Y-%m-%d..."]
    for format in ts_formats:
        try:
            end = datetime.strptime(ts,
      format)
            break
        except ValueError:
            continue
start_tag = "{}-start".format(
    self.results_tag_prefix)
    for result in self.results.
      find_by_tag(
        start_tag):
    day = result.get(seq_idxs.day)
    secs = result.get(seq_idxs.secs)
```

Figure 12: **Completion comparison.** RepoShapley inserts the correct control-flow logic. Full-Retrieve is distracted and hallucinates redundant parsing logic.