Aligning LLMs to Fully Utilize the Cross-file Context in Repository-level Code Completion

Jia Li ♂*
Peking University
Beijing, China
jia_li@mail.tsinghua.edu.cn

Hao Zhu*
Peking University
Beijing, China
zhuhao@stu.pku.edu.cn

Huanyu Liu*
Peking University
Beijing, China
huanyuliu@stu.pku.edu.cn

Xianjie Shi
Peking University
Beijing, China
2100013180@stu.pku.edu.cn

He Zong
aiXcoder
Beijing, China
zonghe@aixcoder.com

Yihong Dong, Kechi Zhang
Peking University
Beijing, China
{dongyh, zhangkechi}@[stu.]pku.edu.cn

Siyuan Jiang
aiXcoder
Beijing, China
jiangsiyuan@aixcoder.com

Zhi Jin, Ge Li†
Peking University
Beijing, China
{zhijin, lige}@pku.edu.cn

***Abstract*—Large Language Models (LLMs) have shown promising results in repository-level code completion, which completes code based on the in-file and cross-file context of a repository. The cross-file context typically contains different types of information (*e.g.,* relevant APIs and similar code) and is lengthy. In this paper, we found that LLMs struggle to fully utilize the information in the cross-file context. We hypothesize that one of the root causes of the limitation is the misalignment between pre-training (*i.e.,* relying on nearby context) and repo-level code completion (*i.e.,* frequently attending to long-range cross-file context).**

To address the above misalignment, we propose Code Long-context Alignment - CoLA, a purely data-driven approach to explicitly teach LLMs to focus on the cross-file context. Specifically, CoLA constructs a large-scale repo-level code completion dataset - CoLA-132K, where each sample contains the long cross-file context (up to 128K tokens) and requires generating context-aware code (*i.e.,* cross-file API invocations and code spans similar to cross-file context). Through a two-stage training pipeline upon CoLA-132K, LLMs learn the capability of finding relevant information in the cross-file context, thus aligning LLMs with repo-level code completion. We apply CoLA to multiple popular LLMs (*e.g.,* aiXcoder-7B) and extensive experiments on CoLA-132K and a public benchmark - CrossCodeEval. Our experiments yield the following results. ❶ *Effectiveness.* CoLA substantially improves the performance of multiple LLMs in repo-level code completion. For example, it improves aiXcoder-7B by up to 19.7% in exact match. ❷ *Generalizability.* The capability learned by CoLA can generalize to new languages (*i.e.,* languages not in training data). ❸ *Enhanced Context Utilization Capability.* We design two probing experiments, which show CoLA improves the capability of LLMs in utilizing the information (*i.e.,* relevant APIs and similar code) in cross-file context. Our datasets and model weights are released in [1].

I. Introduction

Repository-level (Repo-level) code completion aims to complete an unfinished code file based on the in-file context and cross-file context within the current repository [2]–[4]. The cross-file context typically contains different types of information across files (*e.g.,* relevant APIs [4] and similar code [2], [3]) and is lengthy (up to thousands of tokens). In recent years, Large Language Models (LLMs) have achieved State-Of-The-Art (SOTA) performance in repo-level code completion [5]–[7]. To process the long cross-file context, the context windows of LLMs are often large, *e.g.,* 16,384 tokens.

However, we found that existing LLMs struggle to fully utilize information within the cross-file context in repo-level code completion. Our motivation stems from two observations. First, recent studies [8], [9] revealed that LLMs fail to pass simple probing tasks such as Needle-in-the-Haystack, which asks LLMs to recall specific text segments in long contexts. Second, we conducted a preliminary experiment and found similar problems in repo-level code completion. For example, LLMs fail to utilize relevant APIs and similar code presented in the cross-file context. Section II shows two examples and detailed analyses. However, recent studies [2], [3], [10] focus on how to extract relevant context and ignore this severe limitation. Consequently, a pressing research question arises: *How to make LLMs fully utilize the information in the cross-file context?*

We hypothesize that one of the root causes of the above limitation is the misalignment between pre-training and repo-level code completion. The pre-training data typically contains extensive files (*e.g.,* over 300 million files [11]) from different sources, and the correlation between files is often sparse. In auto-regressive pre-training, the loss on predicting the next token is more likely to be influenced by a few nearby pre-tokens rather than long-distance tokens [12], [13]. This training may lead LLMs to focus mainly on the nearby context and ignore the long-distance context. In contrast, in repo-level code completion, the cross-file context is typically lengthy, and LLMs need to frequently attend to the long-distance context. As a result, the misalignment between pre-training and repo-level code completion limits the capabilities of LLMs in utilizing the cross-file context.

Based on this hypothesis, this paper proposes **Co**de **L**ong-context **A**lignment - CoLA, which explicitly teaches LLMs that the crucial information can be presented in the cross-file context, not just in the nearby context. CoLA is a purely data-

* Equal contribution. † Corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00125

driven solution that constructs a repo-level code completion dataset - CoLA-132K. Each sample in the dataset is equipped with the long cross-file context (up to 128K tokens), and asks LLMs to generate context-aware code. In this paper, we focus on two types of crucial information (*i.e.,* relevant APIs and similar code) in the cross-file context, which are widely used in related works [2], [3], [10]. Thus, CoLA asks LLMs to generate two types of context-aware code, *i.e.,* cross-file API invocations and code spans similar to cross-file context. Intuitively, to predict both types of code, LLMs need to master the capability of finding relevant information in the cross-file context, thus aligning LLMs with repo-level code completion.

To achieve the above goal, we construct CoLA-132K from 2,000 high-quality open-source repositories, obtaining 120,000 training samples and 12,000 testing samples. Besides, CoLA-132K covers four popular programming languages (*i.e.,* Python, Java, C++, and Go). We further propose a two-stage pipeline to train LLMs on CoLA-132K. The training pipeline consists of a supervised fine-tuning stage and a reinforcement learning stage. More details are in Section III.

To highlight the effectiveness of CoLA, we apply CoLA to multiple popular LLMs, including aiXcoder-7B [5], DeepSeek-Coder-6.7B [6], and Code Llama-7B [7]. The model trained from aiXcoder-7B is named aiXcoder-7B-v2. The training details are in Section IV.

To evaluate CoLA, we conduct a large-scale study on CoLA-132K (test set) and a public benchmark - CrossCodeEval [14]. Evaluation metrics are Exact Match (EM) and BLEU [15]. Our study yields the following results. ❶ *Effectiveness.* CoLA substantially improves the performance of multiple LLMs in repo-level code completion. For example, it improves aiXcoder-7B by up to 19.7% in EM. aiXcoder-7B-v2 even surpasses larger models (*e.g.,* DeepSeek-Coder-33B). We also exclude the impact of additional training data. ❷ *Generalizability.* From the perspective of languages, the capability learned by CoLA can generalize to new languages (*i.e.,* languages not in CoLA-132K), such as C# and TypeScript. From the perspective of models, CoLA is model-agnostic and effectively improves multiple LLMs, *e.g.,* DeepSeek-Coder and Code Llama. ❸ *Enhanced Context Utilization Capability.* CoLA significantly improves the capability of LLMs in utilizing two types of information (*e.g.,* APIs and similar code) within the cross-file contexts. For example, in more than 90% of test samples, aiXcoder-7B-v2 can accurately locate and invoke the relevant APIs. ❹ *Ablation Study.* The results of the ablation study show that two training stages and two types of context-aware code contribute to CoLA.

The key contributions of this paper are listed as follows:

- We find that LLMs struggle to fully utilize the cross-file context in repo-level code completion. To address this limitation, we propose **Co**de **L**ong-context **A**lignment - CoLA, which is a data-driven solution and explicitly trains LLMs to find relevant information in the cross-file context.
- To support CoLA, we collect and release a repo-level code completion dataset - CoLA-132K. It asks LLMs to generate context-aware code (*i.e.,* cross-file API invocations and code spans similar to the cross-file context).
- We apply CoLA to multiple popular LLMs and conduct extensive experiments. Results show that CoLA substantially improves the performance of LLMs on repo-level code completion.

```
def assert_shape(x: torch.Tensor, shape: Tuple[int, ...]):
  """ Helper function to assert the shape of a tensor.
  Args:
  x: Input tensor.
  shape: Expected shape of the input tensor.
  """
  assert x.shape == shape, f"expected shape {shape}, got {x.shape}"
. . . . .
def _hash_naive(x):
  # Compute the hash of a integer x using the naive algorithm
  x = ((x >> 16) ^ x) * 0x45D9F3B
  x = ((x >> 16) ^ x) * 0x45D9F3B
   = (x >> 16) ^ x
  return x
. . . . .
def process_data(data):
  # Calculate the hash value of the input data
  hashed_data = []
  for i in range(len(data)):
```

(a) In-file + cross-file context (about 16K tokens)

```
Qwen2.5-Coder-7B: hashed_value += data[i] * 31 ** i
aiXcoder-7B: hashed_value += data[i] * i
aiXcoder-7B-v2 (Ours): hashed_value += _hash_naive(data[i])
Ground-truth (Human): hashed_value += _hash_naive(data[i])
```

(b) Outputs of LLMs and ground truth

Fig. 1: Motivating Example ❶. LLMs fail to invoke a relevant API in the cross-file context, leading to suboptimal completions.

II. Motivating Examples

In this section, we conduct a case study to show our motivation, *i.e.,* existing LLMs struggle to fully utilize the information within the cross-file context.

Motivating Example ❶: LLMs fail to invoke the relevant API in the cross-file context, leading to suboptimal results. Figure 1 shows a real-world example of repo-level code completion. In this example, LLMs need to complete an unfinished Python function - `process_data`, which calculates the hash value of input data. There is a relevant API - `_hash_naive` in the cross-file context, which provides a developer-customized approach to computing the hash value. Figure 1 (b) shows the outputs of existing LLMs and the ground-truth code. Qwen2.5-Coder-7B [16] and aiXcoder-7B [5] are two popular LLMs for code and achieve SOTA results in code completion. We can see that human developers invoke the API - `_hash_naive` to compute hash values. However, Qwen2.5-Coder-7B and aiXcoder-7B struggle to invoke the relevant API and output suboptimal completions.

Motivating Example ❷: LLMs struggle to utilize similar code in the cross-file context, outputting wrong results. Figure 2 shows a real-world example of repo-level code completion. LLMs require completing an unfinished function - `get_user_data_storage_path`, which returns the path of the user's data storage. The cross-file context contains a function - `get_user_logging_config_path` with a

```
def is_in_bounds(points: torch.Tensor) -> torch.BoolTensor:
  """
  Check if the points are within the bounds of [-1, 1] in all
dimensions.
  """
  return (points.abs() <= 1.0).all(-1, keepdim=True)
. . . . .
def get_user_logging_config_file():
  logging_config_path = f"{constants.USER_FOLDER}/logging_config.txt"
  if not os.path.exists(logging_config_path):
    os.makedirs(logging_config_path)
  return logging_config_path
. . . . .
def get_user_data_storage_path():
  try:
```

(a) In-file + cross-file context (about 16K tokens)

Qwen2.5-Coder-7B: `data_storage_path = "/user/data/data_storage.db"`

aiXcoder-7B: `data_storage_path = os.environ['DATA_STORAGE_PATH']`

aiXcoder-7B-v2 (Ours):

`data_storage_path = f"{constants.USER_FOLDER}/data_storage.db"`

Ground truth (Human):

`data_storage_path = f"{constants.USER_FOLDER}/data_storage.db"`

(b) Outputs of LLMs and ground truth

Fig. 2: Motivating Example ❷. LLMs struggle to utilize similar code in the cross-file context, outputting wrong results.

similar purpose, which returns the path of the user's logging config file. The similar function shows how to access the user's folder (*i.e.,* `constants.USER_FOLDER`) and is helpful to complete code. Figure 2 (b) shows the outputs of existing LLMs and the ground-truth code. Human developers refer to the similar function and write the correct path to the user's data storage. However, both LLMs do not utilize the information in the similar function and output wrong completions.

The above two examples show that existing LLMs struggle to fully utilize the cross-file context in repo-level code completion. Recent studies [8], [9] also found similar phenomena in natural language understanding tasks.

As stated in Section I, we hypothesize that one root cause of the above limitation is the misalignment between pre-training and repo-level code completion. To address this misalignment, our idea is to train LLMs to generate context-aware code and push LLMs to actively utilize the information in the cross-file context. For example, similar to Figure 1, we train LLMs to predict a cross-file API invocation. To minimize the generation loss, LLMs must learn to find relevant APIs from the lengthy cross-file context, thus naturally aligning to repo-level code completion. Similarly, we train LLMs to generate code similar to the cross-file context and teach LLMs to refer to similar code in the cross-file context. We further propose a two-stage training pipeline to implement the alignment. We apply CoLA to a SOTA LLM - aiXcoder-7B and present aiXcoder-7B-v2. Figure 1 & 2 (b) show the outputs of aiXcoder-7B-v2. aiXcoder-7B-v2 successfully utilizes the information in the cross-file context and outputs correct completions.

III. CoLA

In this section, we propose **Co**de **L**ong-context **A**lignment - CoLA for aligning LLMs to repo-level code completion. Because CoLA is purely a data-driven solution, we first present a dataset - CoLA-132K and describe a two-stage training pipeline.

A. *Dataset* - CoLA-*132K*

The idea of CoLA is to train LLMs to generate context-aware code and push LLMs to utilize information in the cross-file context. Considering existing repo-level code completion datasets (*e.g.,* RepoEval [2]) are small-scale and lack context-aware code, we construct a new dataset - CoLA-132K.

1) Data Statistics: CoLA-132K is a multilingual repo-level code completion dataset, covering four popular programming languages (*i.e.,* Python, Java, C++, and Go). It is collected from 2,000 high-quality open-source repositories and consists of 120,000 training and 12,000 testing samples. CoLA-132K has two features:

- **Long Cross-file Context.** Besides the in-file context, each sample is equipped with the long cross-file context. The average length of the cross-file context is about 12,000 tokens, and the maximum length is over 128,000 tokens. For comparison, a popular repo-level code completion benchmark - CrossCodeEval's context average 3390 tokens [14].
- **Context-aware Code Completion.** To explicitly train LLMs to utilize the cross-file context, the code to be completed is highly context-aware. According to related works [2], [3], [10], we focus on two types of crucial information (*i.e.,* relevant APIs and similar code) in the cross-file context and ask LLMs to generate two types of context-aware code: (1) cross-file API invocations; (2) code spans similar to cross-file context. Figure 3 shows some code examples in CoLA-132K. Intuitively, to predict both types of code, LLMs need to master the capability of finding relevant information in the cross-file context, thus aligning LLMs with repo-level code completion.

2) Data Collection Pipeline: The data collection pipeline of CoLA-132K consists of four stages:

Stage 1: Code Repository Crawling. In this stage, we crawl code repositories from GitHub. We design the following rules to ensure the quality and diversity of repositories:

- *Multilingual Data*. Python, Java, C++, and Go are four popular programming languages in modern software development [17]. Thus, we prioritize repositories in these languages to establish a representative corpus.
- *License Compliance*. Repositories must adopt permissive open-source licenses (*e.g.,* MIT) to avoid legal issues. We use GHArchive [18] to cross-check the license type and ensure compliance with permissive standards.
- *Popularity*. We retain popular repositories that have at least 10 stars and updates within the last two years. Then we sort repositories based on the number of stars, commit frequency, and the presence of test files. The lowest 10% of repositories are removed as they are considered low-quality.

To avoid data leakage, we exclude repositories in existing evaluation benchmarks (*i.e.,* CrossCodeEval in our experiments). For the test set of CoLA-132K, we only consider repositories created after March 2024, since the training data of the latest studied LLM is cut off at March 2024.

```
. . . . .
</s><s># the file path is: model.py
# the code file is written by Python
def convert_to_custom_text_state_dict(state_dict: dict):
</s><s># the file path is: model.py
# the code file is written by Python
def convert_weights_to_lp(model: nn.Module, dtype=torch.float16):
. . . . .
def load_checkpoint(model, checkpoint_path, model_key, strict):
state_dict = load_state_dict(checkpoint_path,
model_key=model_key, is_openai=False)
# detect old format and make compatible with new format
if 'positional_embedding' in state_dict and not hasattr(model,
'positional_embedding'):
```

Cross-file + In-file Context

The Code to be Complete

```
state_dict = convert_to_custom_text_state_dict(state_dict)
```

(a) Cross-file API Invocation

```
. . . . .
</s><s># the file path is: fileLeak.py
# the code file is written by Python
for content in self.page404_content:
  for location_404 in self.location404:
      if location_404 in page.location_url:
        return True
  if not page.location_url.endswith(page.url.payload + "/200"):
    self.location_404_url.add(page.location_url)
    return True
. . . . .
if "/." in page.url.url and page.status_code == 200:
  if len(page.content) == 0:
    return True
```

Cross-file + In-file Context

The Code to be Complete

```
if not page.location_url.endswith(page.url.payload + "/"):
  self.location_404_url.add(page.location_url)
  return True
```

(b) Code span similar to cross-file context

Fig. 3: Two samples in CoLA-132K.

Stage 2: Repo-level Deduplication. Duplicate training data will lead to model overfitting [19]. Thus, we conduct the repo-level data deduplication, which consists of two steps:

- *Exact deduplication.* We remove repositories where all files have the same content.
- *Near deduplication.* Exact deduplication may cause false negatives, so we further perform near deduplication. We compute the MinHash [20] of all files with 256 permutations and use Locality Sensitive Hashing (LSH) [21] to identify clusters of duplicates across repositories. We further reduce these clusters by ensuring that each file in the original cluster is similar to at least one other file in the reduced cluster. We consider two files near-duplicate when their Jaccard similarity [22] exceeds a threshold. We follow related work [5] and empirically set the threshold to 0.85.

Stage 3: Dependency Graph Construction. We use SCIP [23] to parse these repositories. Repositories that fail to parse or exceed the time limit are excluded. The structured output of SCIP contains the locations of various code elements within the repository (*e.g.,* classes and functions) and the dependencies between these elements (*e.g.,* function invocations). This information can then support the subsequent stages.

Stage 4: Context-aware Code Extraction. In this stage, we extract the context-aware code as the ground-truth code (*i.e.,* code to be completed). As stated in Section III-A, we focus on extracting two types of context-aware code:

- *Cross-file API invocations.* Based on the dependency graphs built in stage 3, we identify code lines that contain cross-file API invocations. These lines are then extracted as the ground-truth code.
- *Code spans similar to cross-file context.* Following previous work [5], we sample structured spans from repositories, ensuring they meet the following criteria: non-empty, not comments. We only include code spans, whose maximum similarity with retrieved code snippets in the cross-file context exceeds a threshold. How to retrieve code snippets is described in Stage 5. We manually inspect some retrieved code candidates and empirically set the threshold to 0.3.

To avoid data leakage, we exclude the ground-truth code appearing in our evaluation benchmarks (*i.e.,* CrossCodeEval in our experiments).

Stage 5: Context Extraction. This stage is to extract the input context corresponding to the ground-truth code.

- *In-file context.* Following previous work [5], we locate the current file (*i.e.,* the source file of the ground-truth code) and extract the prefix and suffix as in-file context. The prefix refers to the code preceding the ground-truth code, while the suffix is the code following the ground-truth code.
- *Cross-file context:* Extracting cross-file contexts is an open problem in repo-level code completion. Following previous works [2]–[4], we adopt two mainstream strategies for extracting cross-file context:
  1) *Dependency-based strategy.* The motivation is that imported files often contain elements used by the current file, such as functions. Thus, based on the dependency graphs built in Stage 3, we extract external files imported by the current file as cross-file context.
  2) *Retrieval-based strategy.* The idea of this strategy is to retrieve similar code snippets across the repository as cross-file context. Following previous works [2], [3], we divide the entire repository into multiple code snippets by splitting it into code snippets of fixed size. The size is empirically set to 20 lines according to previous works [2], [3]. These code snippets form a retrieval corpus. From the end of the in-file prefix, we extract a code snippet that serves as the query. We then calculate the Jaccard similarity between the query and each snippet in the retrieval corpus at the token level, ranking the snippets based on similarity. Following previous work [2], we select the top 10 most similar code snippets as cross-file contexts.

B. A Two-Stage Training Pipeline

CoLA employs a two-stage training pipeline upon the dataset, including Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL). We split the training data into two parts in a 5:5 ratio for two stages. An overview of the training pipeline is shown in Algorithm 1.

1) Supervised Fine-Tuning (SFT, Line 1-6 in Algorithm 1): SFT is a common approach to aligning LLMs to specific tasks and is helpful to stabilize reinforcement learning. Through the SFT, models can know the repo-level code completion task

Algorithm 1: The two-stage training pipeline in CoLA.

Input: $\mathcal{M}$: An LLM to be trained, $\mathcal{D}_{\text{SFT}}$: Training data for SFT, $\mathcal{D}_{\text{RL}}$: Training data for RL
Output: $\mathcal{M}_{\text{CoLA}}$: Trained model using CoLA

1 **Supervised Fine-Tuning (SFT):**
2 Initialize model parameters $\mathcal{M}_{\text{SFT}}$ with $\mathcal{M}$;
3 **for** *each batch B from $\mathcal{D}_{SFT}$* **do**
4 Compute the loss $\mathcal{L}_{\text{SFT}}$ on the batch B using Equation 1;
5 Update $\mathcal{M}_{\text{SFT}}$ by minimizing $\mathcal{L}_{\text{SFT}}$;
6 **Preference Data Construction:**
7 *// Create preference data for RL training*;
8 $\mathcal{D}_{\text{Pre}} \leftarrow \emptyset$; `// Initialize preference dataset`
9 **for** *each prompt* $\mathbf{I}$ *and ground truth* $\mathbf{O}_i^*$ *in $\mathcal{D}_{DPO}$* **do**
10 $\mathbf{O} \leftarrow$ Generate multiple candidate completions using $\mathcal{M}_{\text{SFT}}(\mathbf{I}_i)$;
11 Filter $\mathbf{O}^-$ by removing empty, exact matches, partial matches, and duplicates;
12 **for** *each* $\mathbf{O}_j^-$ *in* $\mathbf{O}^-$ **do**
13 $\mathcal{D}_{\text{Pre}} \leftarrow \mathcal{D}_{\text{Pre}} \cup \{(\mathbf{I}_i, \mathbf{O}_i^*, \mathbf{O}_j^-)\}$;
14 **Reinforcement Learning (RL):**
15 Initialize $\mathcal{M}_{\text{RL}}$ with $\mathcal{M}_{\text{SFT}}$;
16 $\pi_{\text{ref}} \leftarrow \mathcal{M}_{\text{SFT}}$;
17 **for** *each batch B in $\mathcal{D}_{Pre}$* **do**
18 Compute the loss $\mathcal{L}_{\text{RL}}$ on the batch B using Equation 2;
19 Update $\mathcal{M}_{\text{RL}}$ by minimizing $\mathcal{L}_{\text{RL}}$;
20 $\mathcal{M}_{\text{CoLA}} \leftarrow \mathcal{M}_{\text{RL}}$;
21 **return** $\mathcal{M}_{\text{CoLA}}$

and generate plausible completions. Specifically, we fine-tune models by minimizing the following loss function:

$$\mathcal{L}_{\text{SFT}} = -\frac{1}{N}\sum_{i=1}^{N} \log P(\mathbf{O}_i^*|\mathbf{I}_i), \tag{1}$$

where $\mathbf{I}_i$ and $\mathbf{O}_i^*$ denote the input context and ground-truth code in the i-th training sample, respectively. N is the size of the training data.

2) Reinforcement Learning (RL, Line 7-21 in Algorithm 1): This stage aims to leverage RL to further improve the model's capability in utilizing cross-file context. We employ a popular offline RL approach - *Direct Preference Optimization (DPO)* [24]. DPO can refine the model's behavior without an explicit reward model and has shown impressive results in many tasks such as mathematical reasoning [25], [26].

Specifically, DPO first collects lots of preference data, which comprises triples - {input context, chosen code, rejected code}. The chosen code is the ground-truth code, and the rejected code is the incorrect code generated by models. DPO increases the generation probability of the chosen code and decreases the generation probability of the rejected code. To obtain the rejected code, we use the SFT-tuned model to generate multiple candidate completions for each input context. We select high-quality rejected code from these candidate completions through rigorous filtration. The filtration excludes three types of completions: (i) empty completions, (ii) same completions as ground truth code, and (iii) completions containing the ground-truth code.

Based on the preference data, we optimize the models by minimizing the following loss function:

$$\mathcal{L}_{\text{RL}} = -\mathbb{E}\left[\log\sigma\left(\beta\left(\log\frac{\pi_\theta(\mathbf{O}^*|\mathbf{I})}{\pi_{\text{ref}}(\mathbf{O}^*|\mathbf{I})} - \log\frac{\pi_\theta(\mathbf{O}^-|\mathbf{I})}{\pi_{\text{ref}}(\mathbf{O}^-|\mathbf{I})}\right)\right)\right] \tag{2}$$

where $\mathbf{I}$, $\mathbf{O}^*$, and $\mathbf{O}^-$ denote the input context, chosen code, and rejected code. π_θ is the current model being optimized and π_{ref} is the frozen reference model (*i.e.,* the SFT-tuned model). β is a scaling hyper-parameter, and σ is the common sigmoid activation function.

The loss function encourages the model π_θ to assign higher probabilities to $\mathbf{O}^*$ and lower probabilities to $\mathbf{O}^-$. The term $\log\frac{\pi_\theta(\mathbf{O}|\mathbf{I}')}{\pi_{\text{ref}}(\mathbf{O}|\mathbf{I}')}$ represents the log-probability ratio between the current model and the reference model. This ratio acts as a regularization term, ensuring that the optimized model does not deviate too far from the original SFT-tuned policy.

IV. aiXcoder-7B-v2

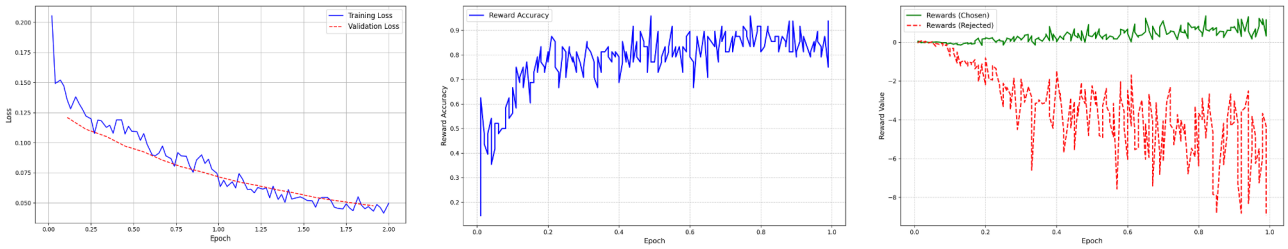
CoLA is general to existing LLMs. We take the SOTA LLM on code completion - aiXcoder-7B [5] as an example and present aiXcoder-7B-v2. This section describes the training details and the training curves.

A. Training Details

SFT (Line 1-6 in Algorithm 1). We initialize $\mathcal{M}_{\text{SFT}}$ with aiXcoder-7B. Then, we fine-tune the model for 2 epochs. To prevent overfitting, we use a small learning rate of 2×10^{-6}, a weight decay of 0.01, and a maximum gradient norm of 1.0. The batch size is 128. These hyper-parameters are inspired by previous work [27]. For computational efficiency, we employ the TRL framework [28] with DeepSpeed Zero-3 [29] and Flash Attention-2 [30]. SFT is conducted on 8 NVIDIA A100 (40GB) GPUs and costs 20 hours.

Preference Data Construction (Line 7-14 in Algorithm 1). For each sample, the SFT-tuned model $\mathcal{M}_{\text{SFT}}$ generates 10 candidate completions using Top-p sampling [31] with $p = 0.95$ and temperature $T = 1.5$ to obtain diverse completions. To ensure training efficiency, we select up to three unique and incorrect completions for each sample as rejected code. Finally, we obtain 75,169 triples as preference data.

DPO (Line 15-20 in Algorithm 1). We implement the $\mathcal{M}_{\text{DPO}}$ and reference model π_{ref} with the SFT-tuned model $\mathcal{M}_{\text{SFT}}$. To improve training efficiency, we use Low-Rank Adaptation (LoRA) [32] to train $\mathcal{M}_{\text{DPO}}$. The rank and alpha values of LoRA are set to 32 and 16. We apply LoRA to all transformer modules, including all self-attention weights and feed-forward weights. We train $\mathcal{M}_{\text{DPO}}$ for 1 epoch over the



(a) Training Losses During SFT (b) Reward Accuracy During DPO Training (c) Rewards During DPO Training

Fig. 4: Training curves in SFT and RL. (a) Training and validation losses during SFT. (b) Reward accuracy during RL training. (c) Chosen vs. rejected rewards during RL training.

preference dataset with a batch size of 128. β is set to 0.9. Existing work [33] found that LoRA's best learning rates often in $[1\times10^{-5}, 5\times10^{-4}]$. Thus, we use a learning rate of 1×10^{-4} with a linear scheduler and warm-up. DPO is conducted on 8 NVIDIA A100 (40GB) GPUs and costs 31 hours.

B. Training Curves

We show the training curves of aiXcoder-7B-v2, including the training loss, reward accuracy, and the reward of chosen and rejected code.

Training curves in SFT. During the SFT phase, we randomly selected 5% of the training data for validation. As shown in Figure 4a, both the training and validation losses decrease steadily over the course of training, indicating that the model is converging.

Training curves in RL. We monitor the RL training process using two metrics: reward accuracy and the reward of chosen and rejected code. ❶ *Reward accuracy* reflects the model's ability to distinguish between chosen and rejected code. The curve of reward accuracy is shown in Figure 4b. We can see a rapid increase in reward accuracy during the initial 0.4 epochs, where the accuracy rises from 0.2 to 0.8. Between 0.4 and 1.0 epochs, the accuracy continues to improve, reaching a plateau around 0.9. ❷ *Reward of chosen and rejected code* means the mean log probabilities of the model for the chosen and rejected code. Figure 4c shows the reward of the chosen and rejected code during RL training. We can see that the reward of the chosen code consistently increases, and the reward of rejected completions declines. The two curves above show that the model effectively distinguishes between chosen and rejected code and tends to generate chosen code.

V. Study Design

To evaluate the effectiveness of CoLA, we conduct a large-scale study. In this section, we describe the design of our study, including research questions, benchmarks, baselines, and evaluation metrics.

A. Research Questions

Our study aims to answer the following research questions (RQs):

RQ1: How much improvement does CoLA achieve in repo-level code completion? We aim to evaluate the improvement in repo-level code completion performance when using CoLA. We apply CoLA to aiXcoder-7B and present aiXcoder-7B-v2. Then we compare it to existing SOTA LLMs on multiple benchmarks.

RQ2: How effectively does CoLA generalize across different programming languages? This RQ investigates whether the context utilization capabilities learned by CoLA can generalize across languages. We train aiXcoder-7B-v2 on CoLA-132K covering four languages and test the model on the other two languages.

RQ3: Does CoLA maintain its effectiveness across different LLMs? Besides the aiXcoder-7B, we apply CoLA to other two popular LLMs (*i.e.,* DeepSeek-Coder-6.7B and Code Llama-7B) to evaluate whether CoLA can improve them in repo-level code completion.

RQ4: Does CoLA improve the capability of LLMs to utilize the cross-file context? This RQ evaluates whether CoLA enhances the LLMs' capability of utilizing APIs and similar code in the cross-file context.

RQ5: How do different types of context-aware code and training stages contribute to CoLA? This RQ conducts an ablation study, which explores the contributions of two types of context-aware code (*i.e.,* cross-file API invocations and code spans similar to cross-file context) and two training stages (*i.e.,* SFT and RL) to CoLA.

B. Benchmarks

We conduct experiments on the following benchmarks:

CoLT-132K (Test) is the test set of CoLA-132K, consisting of 12,000 samples. It covers four programming languages (*i.e.,* Python, Java, C/C++, Go) and asks LLMs to generate two types of context-aware code, including cross-file API invocations and code spans similar to the cross-file context. Each language contains 3,000 samples. Notably, the test set was meticulously collected from repositories created after March 2024 to prevent data leakage. For detailed information, please refer to Section III-A.

CrossCodeEval [14] is a multilingual repo-level code completion benchmark, including 2,665 Python samples, 2,139 Java samples, 3,356 TypeScript samples, and 1,768 C# samples. We reuse the cross-file context retrieved by BM25 provided in CrossCodeEval.

C. Baseline Models

We selected 8 popular open-source LLMs for comparison, which are widely used in code completion tasks. These models are categorized into two groups based on their scales.

❶ **LLMs with 7 billion parameters.** These LLMs have a comparable parameter scale to aiXcoder-7B-v2.

- **aiXcoder-7B** [5] is developed by aiXcoder company. With 7 billion parameters, aiXcoder-7B achieves SOTA performance in coding tasks, particularly in code completion.
- **aiXcoder-7B-cont** is a continually pre-trained model of aiXcoder-7B. We randomly sample code files from repositories in CoLA-132K and train aiXcoder-7B with its original pre-training objective. The amount of training tokens is equal to that of aiXcoder-7B-v2. We contacted the developers of aiXcoder-7B and used the official code implementation they provided for pre-training.
- **Qwen2.5-Coder-7B** [16], developed by Alibaba, is an enhanced version of Qwen2.5 [27]. It is trained on an expansive dataset comprising source code, text-code alignment data, and synthetic data, amounting to 5.5 trillion tokens.
- **DeepSeek-Coder-6.7B** [6], from DeepSeek AI, is trained from scratch on 2 trillion tokens across 87 programming languages. It is designed to handle complex coding tasks.
- **Code Llama-7B** [7], developed by Meta AI, is initialized with Llama 2 [34] and trained on 500B tokens from a large-scale code corpus. It supports common programming languages and has a default context length of 100K tokens.
- **StarCoder2-7B** [35], introduced by BigCode, is trained on The Stack v2 dataset following additional code fine-tuning stage. It supports multilingual code generation.

❷ **LLMs with larger sizes.** We also add two LLMs with larger scales as baselines.

- **Code Llama-13B** [7], the upgraded version of the Code Llama model, expands the parameter size to 13 billion.
- **DeepSeek-Coder-33B** [6] is the largest variant of the dense model of the DeepSeek-Coder series.

We employ the official prompting formats of LLMs for fill-in-the-middle (FIM) code completion. Specifically,

- DeepSeek-Coder: `[cross-file context]<|fim_begin|>[prefix]<|fim_hole|>[suffix]<|fim_end|>`
- aiXcoder: `[cross-file context]<AIX-PRE><AIX-POST>[suffix]<AIX-MIDDLE>[prefix]`
- Code Llama: `[cross-file context]<PRE>[prefix]<SUF>[suffix]<MID>`
- Qwen2.5-Coder: `<|repo_name|>[name]<|file_sep|>[path]<|fim_prefix|>[prefix]<|fim_suffix|>[suffix]<|fim_middle|>`

When producing prompts, we first include the in-file prefix and suffix. Then, we add cross-file context from dependency-based and retrieval-based methods in an alternating order, until reaching the maximum context window of LLMs. We employ the original maximum context window of each model.

We omit some related repo-level code completion approaches for comparison [2]–[4], [10], as we focus on different research problems. **Our approach focuses on improving the capability of LLMs in utilizing context.** In contrast, existing repo-level code completion approaches [2]–[4], [10] focus on how to extract relevant context and treat LLMs as black-box code generators. In practice, our work and these recent works are complementary. The experimental results in Section VII-B demonstrate the complementarity of our work and existing repo-level code completion approaches.

D. Evaluation Metrics

Following previous works [2], [4], [5], [14], we use the following two evaluation metrics:

- **Exact Match (EM)** quantifies the proportion of predictions that identically match ground truths.
- **BLEU [36]** computes n-gram overlap between predictions and ground truths. n is empirically set to 4.

Following recent works [6], [14], we use greedy decoding during inference. When predicting API invocations, we extract the first non-empty line of LLMs' completions for evaluation. When predicting code spans, we truncate the model outputs at the special token and extract the rest code for evaluation.

VI. Results and Analyses

A. RQ1: Improvements in Repo-level Code Completion

Setting. As stated in Section IV, we apply CoLA to aiXcoder-7B and present aiXcoder-7B-v2. To validate the effectiveness of CoLA, we compare aiXcoder-7B-v2 to eight LLMs on repo-level code completion.

Results and Analyses. The results are shown in Table I and Table II. We show the relative improvements of aiXcoder-7B-v2 compared to aiXcoder-7B.

CoLA **substantially improves the performance of aiXcoder-7B on repo-level code completion.** aiXcoder-7B-v2 substantially outperforms aiXcoder-7B on CoLA-132K and CrossCodeEval. Especially when completing Python's code spans, aiXcoder-7B-v2 outperforms aiXcoder-7B by 19.7% in EM and 14.3% in BLEU. aiXcoder-7B-v2 even outperforms larger models, including Code Llama-13B and DeepSeek-Coder-33B. The results show that CoLA can effectively improve the performance of LLMs in repo-level code completion.

The effectiveness of CoLA **is not attributed to additional training data.** aiXcoder-7B-cont is continually trained from aiXcoder-7B with its original pre-training objectives. The amount of training tokens is equal to that of aiXcoder-7B-v2. We can see that aiXcoder-7B-v2 substantially outperforms aiXcoder-7B-cont. It demonstrates that the superiority of CoLA is not caused by additional training data. We also notice that aiXcoder-7B-cont is slightly weaker than aiXcoder-7B. We speculate that this is because aiXcoder-7B has been fully trained on 1.2 trillion unique tokens. Further pre-training may cause slight overfitting. We also contacted the development team of aiXcoder-7B and confirmed the reliability of the experimental results.

TABLE I: The performance of LLMs in CoLA-132K.

Model	Cross-file API Invocation								Code Span							
	Python		Java		C++		Go		Python		Java		C++		Go	
	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU
7B Models																
Code Llama-7B	55.8	68.4	66.7	78.5	50.7	65.6	50.1	63.4	39.9	56.8	53.9	68.5	37.3	53.1	46.5	61.2
StarCoder2-7B	46.9	61.3	64.8	78.4	37.8	52.9	51.2	69.6	25.5	37.5	40.3	50.2	24.8	36.8	34.8	47.2
DeepSeek-Coder-6.7B	58.0	69.4	67.4	78.6	46.8	59.7	57.5	71.7	41.4	59.1	58.9	73.2	38.5	54.8	57.2	72.2
Qwen2.5-Coder-7B	55.3	75.6	66.1	84.4	50.7	74.0	45.8	69.5	48.1	67.9	59.1	81.6	46.5	68.7	57.3	75.9
aiXcoder-7B	62.5	78.5	73.3	88.7	56.0	79.2	62.3	83.2	48.6	67.1	68.8	83.4	52.9	70.1	65.2	79.0
aiXcoder-7B-cont	61.6	73.7	73.5	84.6	56.0	71.3	61.7	76.8	42.9	59.2	67.1	80.6	47.2	62.8	47.1	59.7
aiXcoder-7B-v2	**69.7**	**83.8**	**78.5**	**89.6**	**64.1**	**80.6**	**68.2**	**84.1**	**58.2**	**76.7**	**74.8**	**89.2**	**61.1**	**79.9**	**73.8**	**87.4**
Relative Improve.	+11.5%	+6.8%	+7.1%	+1.0%	+14.5%	+1.8%	+9.5%	+1.1%	+19.7%	+14.3%	+8.8%	+7.0%	+15.5%	+13.9%	+13.2%	+10.6%
>7B Models																
Code Llama-13B	56.4	69.8	67.2	79.2	52.1	67.5	45.4	55.9	46.1	63.1	61.6	76.6	45.8	62.7	49.6	66.5
DeepSeek-Coder-33B	61.3	71.6	70.1	81.2	51.5	65.6	60.1	73.9	44.6	62.5	62.4	77.2	43.8	61.0	59.4	74.8

TABLE II: The performance of LLMs on CrossCodeEval.

Model	Python		Java	
	EM	BLEU	EM	BLEU
7B Models				
Code Llama-7B	36.06	54.27	40.58	65.04
StarCoder2-7B	11.97	37.04	34.32	56.71
DeepSeek-Coder-6.7B	13.32	35.00	36.65	59.38
Qwen2.5-Coder-7B	38.27	57.53	42.59	65.98
aiXcoder-7B	31.22	52.69	42.50	66.59
aiXcoder-7B-cont	32.26	52.86	43.19	66.72
aiXcoder-7B-v2	**40.52**	**63.40**	**51.43**	**74.95**
Relative Improve.	+29.8%	+20.3%	+21.0%	+12.6%
>7B Models				
Code Llama-13B	38.76	57.19	42.87	66.01
DeepSeek-Coder-33B	18.46	39.39	37.21	59.57

TABLE III: The generalization ability of aiXcoder-7B-v2.

Model	TypeScript		C#	
	EM	BLEU	EM	BLEU
7B Models				
Code Llama-7B	45.29	61.53	51.07	68.22
StarCoder2-7B	33.55	55.65	42.48	61.47
DeepSeek-Coder-6.7B	34.80	53.02	37.05	52.44
Qwen2.5-Coder-7B	47.34	66.28	**57.86**	74.28
aiXcoder-7B	46.96	66.65	46.32	68.97
aiXcoder-7B-cont	47.19	67.24	47.16	69.62
aiXcoder-7B-v2	**55.96**	**75.74**	55.88	**76.17**
Relative Improve.	+19.2%	+13.6%	+20.6%	+10.4%
>7B Models				
Code Llama-13B	49.20	64.55	57.07	71.22
DeepSeek-Coder-33B	38.17	56.32	38.12	53.95

Answer to RQ1: aiXcoder-7B-v2 substantially improves aiXcoder-7B on repo-level code completion, showing CoLA's effectiveness. The improvements over aiXcoder-7B-cont exclude the impact of additional training data.

B. RQ2: Generalization Across Different Programming Languages

Setting. We train aiXcoder-7B-v2 on CoLA-132K including four programming languages (*i.e.,* Python, Java, C++, and Go). To evaluate generalization ability, we test it on the other two languages (*i.e.,* C# and TypeScript) in CrossCodeEval.

Results and Analyses. The results are shown in Table III.

The context utilization ability learned by CoLA can generalize to new languages. In languages not included in CoLA-132K, such as C# and TypeScript, aiXcoder-7B-v2 demonstrates strong performance. In TypeScript, EM increases from 46.96 to 55.96, a 19.17% improvement. In C#, EM increases from 46.32 to 55.88, a boost of 20.64%. These results show that the context utilization capability learned by CoLA has a good generalization ability.

Answer to RQ2: The capability learned by CoLA generalizes to new languages. In languages like C# and TypeScript, aiXcoder-7B-v2 achieves significant improvements, with EM increasing by up to 19.17% and 20.64%.

C. RQ3: Effectiveness Across Different LLMs

This RQ validates that CoLA is general to different LLMs and can bring stable improvements.

Setting. We apply CoLA to DeepSeek-Coder-6.7B and Code Llama-7B. The training details are consistent with aiXcoder-7B-v2 (Section IV). Then, we evaluate the improvements of CoLA upon both LLMs.

Results and Analyses. Table IV shows the results on CoLA-132K.

CoLA **brings substantial improvements on both LLMs.** The application of CoLA to both LLMs brings significant improvements in both EM and BLEU. For DeepSeek-Coder-6.7B, the EM is improved by up to 53.8% and BLEU is improved by up to 41.9%. For Code Llama-7B, the EM is improved by up to 54% and BLEU is improved by up to 43.8%. These results show the effectiveness of CoLA across different LLMs.

TABLE IV: The performance of CoLA on DS-Coder-6.7B and Code Llama-7B upon CoLA-132K. "DS": DeepSeek.

Model	Cross-file API Invocation								Code Span							
	Python		Java		C++		Go		Python		Java		C++		Go	
	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU	EM	BLEU
DS-Coder-6.7B	58.0	69.4	67.4	78.6	46.8	59.7	57.5	71.7	41.4	59.1	58.9	73.2	38.5	54.8	57.2	72.2
+CoLA	**70.7**	**84.3**	**77.0**	**88.7**	**63.3**	**78.9**	**68.2**	**84.1**	**57.6**	**75.6**	**73.6**	**87.6**	**59.2**	**77.8**	**73.4**	**87.0**
Relative Improve.	+21.9%	+21.5%	+14.2%	+12.9%	+35.3%	+32.1%	+18.7%	+17.3%	+39.1%	+27.9%	+25.0%	+19.7%	+53.8%	+41.9%	+28.4%	+20.4%
Code Llama-7B	55.8	68.4	66.7	78.5	50.7	65.6	50.1	63.4	39.9	56.8	53.8	68.5	37.3	53.0	46.5	61.2
+CoLA	**67.8**	**84.5**	**75.7**	**87.1**	**60.3**	**77.2**	**66.1**	**82.7**	**53.2**	**72.4**	**72.2**	**87.3**	**57.1**	**76.2**	**71.6**	**85.8**
Relative Improve.	+21.5%	+23.5%	+13.5%	+10.9%	+18.9%	+17.7%	+31.9%	+30.4%	+33.3%	+27.5%	+34.2%	+27.4%	+53.2%	+43.8%	+54.0%	+40.2%

TABLE V: The API accuracy of LLMs on CoLA-132K.

Model	API Accuracy (%)			
	Python	Java	C++	Go
Code Llama-7B	76.6	79.6	70.0	62.9
StarCoder2-7B	71.0	81.3	57.6	69.1
DeepSeek-Coder-6.7B	77.6	80.4	64.3	70.3
Qwen2.5-Coder-7B	82.8	80.7	70.2	60.6
Code Llama-13B	78.6	81.6	71.8	56.5
DeepSeek-Coder-33B	79.7	82.8	69.8	74.2
aiXcoder-7B	83.4	86.5	76.6	76.4
aiXcoder-7B-v2	**94.4**	**92.4**	**87.7**	**81.3**

Answer to RQ3: Besides aiXcoder-7B, CoLA effectively improves different LLMs (*e.g.,* DeepSeek-Coder-6.7B and Code Llama-7B). The EM is improved by up to 54% and BLEU is improved by up to 43.8%.

D. RQ4: Improvements on Context Utilization Capability

In this RQ, we design two probing experiments to evaluate whether CoLA enhances the model's ability to utilize APIs and similar code in the cross-file context.

❶ The performance in utilizing APIs in the cross-file context.

Setting. We expect LLMs can actively locate and invoke relevant APIs presented in the cross-file context. To quantify the model's ability to invoke these APIs, we introduce a new metric: **API Accuracy (API Acc)**. It only measures the accuracy of API names in the completion results. If the API name in LLMs' completions matches the ground truth, it is counted as 1; otherwise, it is 0. Different from traditional evaluation metrics (*e.g.,* EM), API Acc only cares whether LLMs invoke correct APIs and excludes the impact of other factors (*e.g.,* arguments).

Results and Analyses. As shown in Table V, across all four programming languages, CoLA substantially improves the API acc of aiXcoder-7B. Specifically, aiXcoder-7B-v2 surpasses aiXcoder-7B by 13.2% on Python, 6.8% on Java, 14.5% on C++, and 6.4% on Go. Figure 1 shows a real case, where only aiXcoder-7B-v2 correctly invokes APIs among the three LLMs. These results demonstrate that CoLA improves the capability of LLMs in utilizing relevant APIs.

❷ The performance in utilizing similar code in the cross-file context.

Setting. In this experiment, we explore whether CoLA enhances the model's ability to utilize similar code. We categorize test samples in CoLA-132K (Code Spans) into different groups based on the maximum similarity between the **retrieved code snippets** and the **ground-truth code**. For each group, we report the number of correct completions by aiXcoder-7B and aiXcoder-7B-v2.

Results and Analyses. The results are presented in Figure 5. aiXcoder-7B-v2 substantially outperforms aiXcoder-7B across all groups. Figure 2 shows a real case, where aiXcoder-7B-v2 successfully utilizes the information in similar code and generates correct code. The results show that CoLA enhances the ability of LLMs to reuse knowledge (*e.g.,* user-defined paths) in similar code.

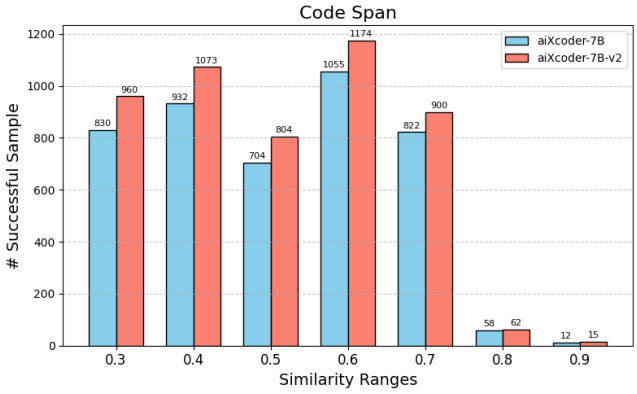


Fig. 5: The performance of aiXcoder-7B and aiXcoder-7B-v2 on different groups of test samples. The x-axis represents the maximum similarity between the ground truth code and the code snippets.

Answer to RQ4: CoLA significantly improves the capability of aiXcoder-7B in utilizing relevant APIs and similar code in the cross-file context.

E. RQ5: Ablation study

In this RQ, we conduct an ablation study, which evaluates the contributions of two types of training data (*i.e.,* cross-file API invocations and code spans similar to cross-file context) and two training stages (*i.e.,* SFT and RL) to CoLA.

Settings. To quantify the contributions of each component, we remove different components from aiXcoder-7B-v2 and evaluate the model on CoLA-132K. We report the average

TABLE VI: The results of the ablation study.

SFT	DPO	EM	BLEU
✗	✗	60.4	77.4
✓	✗	67.3 (↑ 11.39%)	83.3 (↑ 7.66%)
✓	✓	**68.0** (↑ 12.55%)	**83.7** (↑ 8.14%)
API	**Code Span**	**EM**	**BLEU**
✗	✗	60.4	77.4
✓	✗	52.4 (↓ 13.2%)	72.1 (↓ 6.8%)
✗	✓	60.2 (↓ 0.3%)	76.7 (↓ 0.9%)
✓	✓	**68.0** (↑ 12.55%)	**83.7** (↑ 8.14%)

EM and BLEU scores on the whole test set. Since DPO relies on SFT for cold start, otherwise it cannot stabilize training, we do not show the experimental results of utilizing only DPO.

Results and Analyses. The results of the ablation study are shown in Table VI. From the perspective of training stages, both SFT and DPO contribute to CoLA's performance. From the view of the training data, training only on cross-file API invocations or code spans causes slight overfitting, resulting in decreased accuracy. Thus, two types of context-aware code and two training stages all contribute to CoLA's performance.

Answer to RQ5: Two types of context-aware code and two training stages all contribute to the performance of CoLA.

VII. Discussion

A. Performance on Single-file Code Completion and Code Generation

TABLE VII: The performance of aiXcoder-7B and aiXcoder-7B-v2 upon single-file completion (EM and BLEU) and code generation (Pass@1).

Model	Single-file		HumanEval
	EM	BLEU	Pass@1
aiXcoder-7B	46.1	65.8	43.2
aiXcoder-7B-v2	53.1	74.6	**51.8**
Relative Improve.	+15.2%	+13.8%	+20.0%

A natural concern is whether CoLA improves repo-level code completion but hurts the performance on other tasks. To address this concern, we evaluate the performance of aiXcoder-7B and aiXcoder-7B-v2 on single-file completion and general code generation. For single-file completion, we remove all cross-file contexts in CoLA-132K and evaluate models on the new data. For code generation, we evaluate models on a popular code generation benchmark - HumanEval [37]. We report the average of EM and BLEU for single-file completion, and Pass@1 for code generation.

Results and Analyses. The results are shown in Table VII. On single-file completion, the EM of aiXcoder-7B-v2 reaches 53.1, compared to 46.1 of aiXcoder-7B, with an improvement of 15.2%. On HumanEval, the Pass@1 of aiXcoder-7B-v2 reaches 51.8, compared to 43.2 of aiXcoder-7B, with an improvement of 20.0%. These results show that CoLA can improve the code generation and completion capabilities of LLMs beyond repo-level code completion.

TABLE VIII: The performance of CoLA when integrated with RepoCoder and GraphCoder.

CoLA + **RepoCoder**				
Model	**Oracle**		**Iteration-2**	
	EM	BLEU	EM	BLEU
aiXcoder-7B	48.11	58.41	38.56	49.28
aiXcoder-7B-v2	**53.94**	**64.77**	**45.06**	**56.76**
Relative Improve.	+12.1%	+10.9%	+16.9%	+15.2%
CoLA + **GraphCoder**				
Model	**Line-level**		**API-level**	
	EM	BLEU	EM	BLEU
aiXcoder-7B	10.50	16.57	6.72	15.06
aiXcoder-7B-v2	**13.63**	**20.61**	**9.11**	**20.09**
Relative Improve.	+29.7%	+24.4%	+35.6%	+33.4%

B. Complementary to Existing Code Completion Methods

To validate the complementarity between our work and existing code completion approaches, we apply aiXcoder-7B and aiXcoder-7B-v2 (trained by our work) into two representative repo-level code completion approaches (*i.e.,* RepoCoder [2] and GraphCoder [3]). We conduct experiments on the original benchmarks in RepoCoder and GraphCoder. The results are shown in Table VIII. RepoCoder / GraphCoder with aiXcoder-7B-v2 outperforms that with aiXcoder-7B. For example, RepoCoder achieves an EM improvement from 38.56 to 45.06 after Iteration-2, while GraphCoder improves from 10.50 to 13.63 at the line level. These results indicate that our work complements existing repo-level code completion approaches.

VIII. Threats to Validity

There are four main threats to the validity of our work:

Data leakage. It means that the test data is leaked to the training data, resulting in unfair evaluations. Our experiments are conducted on two benchmarks, *i.e.,*CoLA-132K (Test set) and CrossCodeEval. We take three measures to address the data leakage threat. First, the training data of our studied LLMs is cut off as of March 2024. Thus, we only collect repositories created after March 2024 to construct CoLA-132K (Test set). Thus, the test data of CoLA-132K does not appear in the training data of LLMs. Second, when collecting CoLA-132K, we excluded repositories and the ground-truth code in CrossCodeEval. Thus, CrossCodeEval is not leaked to the training data of CoLA-132K. Third, as stated in the original paper of aiXcoder-7B [5], the developers of aiXcoder-7B have excluded CrossCodeEval from the pre-training data. Thus, the improvements of aiXcoder-7B-v2 on CrossCodeEval are not impacted by data leakage.

Limited types of context-aware code. Our CoLA-132K only considers two types of context-aware code (*i.e.,* cross-file API invocations and code spans similar to cross-file context). Our motivation is that these two types of code have explicit and common relationships with cross-file context (*e.g.,* calling, high code similarity). Thus, we can leverage parsers and code retrievers to collect these types of code for training LLMs. We also notice that there may be *implicit* relationships between

code and the cross-file context, for example, providing background of the current repository. However, it is hard to define implicit relationships and collect such type of code. Thus, we leave other types of context-aware to future work.

The performance on generating code without similar code. As stated in Section III-A2, CoLA-132K only retains code spans whose maximum similarity with the cross-file context exceeds the threshold (*i.e.,* 0.3). Thus, one threat is: how well does the model perform when the code to be completed has low similarity to the cross-file context? To address this threat, we collect code spans whose maximum similarity with the cross-file context is lower than the threshold. aiXcoder-7B-v2 correctly completes 320 code spans, and aiXcoder-7B correctly completes 217 code spans, demonstrating improved performance on low-similarity cases.

The impact of repository versions. Following standard practices [2]–[4], [14], we extract the in-file and cross-file context from the current version of the repositories. However, when developers write the ground-truth code, the repository in a previous version, and the context may be slightly different. This problem is unexplored and out of the scope of our work. Besides, it is hard to precisely extract context due to the tangled commits. To address this threat, we select 10 repositories from CoLA-132K, manually add an incomplete function, and ask LLMs to complete the function. aiXcoder-7B-v2 correctly completes 6 functions, while aiXcoder-7B-v2 only completes 3 functions. The results avoid the impact of repository versions and show the effectiveness of CoLA.

IX. Related Work

Repo-level Code Completion. Repo-level code completion has emerged as a critical challenge in software engineering, aiming to enhance code completion accuracy by leveraging the broader repository context. Prior studies have predominantly focused on the retrieval of relevant context from the repository, with various strategies proposed to extract pertinent code snippets [10], [38]–[40]. For instance, RepoCoder [2] introduces an iterative retrieval and generation approach. Similarly, CrossCodeEval [14] proposes a straightforward retrieval method, using the lines preceding the cursor as a query to extract and append similar code snippets. GraphCoder [3] advances this by constructing a Code Context Graph (CCG) to represent the repository, enabling more sophisticated retrieval based on control flow, data flow, and dependencies. While these approaches have made significant strides in context retrieval, how to effectively utilize the retrieved context remains underexplored. Most existing studies view LLMs as a "perfect" generator and simply input long contexts into LLMs. However, this simplistic approach may not fully exploit the potential of the retrieved information. To address this limitation, FT2Ra [41] integrates retrieval results into the generation process by modifying the model's logits based on the retrieved content. Similarly, COCOMIC [4] fuses retrieved cross-file contexts with in-file contexts within the model's self-attention mechanism. However, both FT2Ra and COCOMIC require modifications to the model architecture, making them unsuitable for already pre-trained LLMs.

Our work diverges from this retrieval-centric paradigm by focusing on how to effectively assimilate and utilize the long cross-file context through a novel approach. In practice, our approach is complementary to the above related work.

Repo-level Pre-training. Repo-level pre-training has become a popular approach to enhancing LLMs' long-context code completion capability [6], [16], [42], [43], creating long-context samples by co-locating relevant files within repositories and continually pre-training LLMs on these samples.

Compared with existing repo-level pre-training studies, our work has the following advantages: (1) Data construction. Existing repo-level pre-training focuses on how to pack cross-file context, such as dependency graph-based packing and unit test-based lexical packing in CodeGemma [43]. However, the ground truths may be irrelevant to the cross-file context, making it difficult to effectively enhance long-context code completion. In contrast, our work emphasizes context-aware code completion. The ground truths (*e.g.,* cross-file API invocations) are highly relevant to the cross-file context, thus benefiting long-context code completion. (2) Training techniques. Existing studies mainly employ the next-token prediction training, whereas we introduce reinforcement learning to explicitly penalize LLMs when they fail to utilize the context effectively. (3) Further improvements. Models such as aiXcoder-7B and DeepSeek-Coder already perform repo-level pre-training. Our proposed approach can still further improve their performance, demonstrating the effectiveness and complementarity of our work.

X. Conclusion and Future Work

In this paper, we reveal a limitation of LLMs in repo-level code completion, *i.e.,* LLMs struggle to fully utilize information (*e.g.,* relevant APIs or similar code) in the cross-file context. To address this limitation, we propose CoLA, a purely data-driven approach to explicitly teach LLMs to focus on the cross-file context. CoLA constructs a large-scale repo-level code completion dataset - CoLA-132K and trains LLMs to generate context-aware code (*i.e.,* cross-file API invocations and code spans similar to cross-file context). We apply CoLA to multiple LLMs (*e.g.,* aiXcoder-7B) and conduct extensive experiments on CoLA-132K and CrossCodeEval. Results show that CoLA substantially improves the performance of multiple LLMs in repo-level code completion.

In the future, we will further improve the ability of LLMs to process lengthy contexts. On the one hand, we plan to explore effective approaches to expanding the context window of LLMs. On the other hand, we will study new training techniques to help LLMs learn long-range dependencies.

Acknowledgment

This research is supported by the National Key R&D Program under Grant No. 2023YFB4503801, the National Natural Science Foundation of China under Grant No. 62192733, 62192730, 62192731, the Major Program (JD) of Hubei Province (No.2023BAA024).

REFERENCES

[1] "Replication package." [Online]. Available: https://anonymous.4open.science/r/aiXcoder-7B-v2/README.md

[2] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J. Lou, and W. Chen, "Repocoder: Repository-level code completion through iterative retrieval and generation," in *EMNLP*. Association for Computational Linguistics, 2023, pp. 2471–2484.

[3] W. Liu, A. Yu, D. Zan, B. Shen, W. Zhang, H. Zhao, Z. Jin, and Q. Wang, "Graphcoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph," in *ASE*. ACM, 2024, pp. 570–581.

[4] Y. Ding, Z. Wang, W. U. Ahmad, M. K. Ramanathan, R. Nallapati, P. Bhatia, D. Roth, and B. Xiang, "Cocomic: Code completion by jointly modeling in-file and cross-file context," in *LREC/COLING*. ELRA and ICCL, 2024, pp. 3433–3445.

[5] S. Jiang, J. Li, H. Zong, H. Liu, H. Zhu, S. Hu, E. Li, J. Ding, Y. Han, W. Ning, G. Wang, Y. Dong, K. Zhang, and G. Li, "aixcoder-7b: A lightweight and effective large language model for code completion," *CoRR*, vol. abs/2410.13187, 2024.

[6] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming - the rise of code intelligence," *CoRR*, vol. abs/2401.14196, 2024.

[7] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," *CoRR*, vol. abs/2308.12950, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.12950

[8] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024. [Online]. Available: https://doi.org/10.1162/tacl_a_00638

[9] P. Xu, W. Ping, X. Wu, L. McAfee, C. Zhu, Z. Liu, S. Subramanian, E. Bakhturina, M. Shoeybi, and B. Catanzaro, "Retrieval meets long context large language models," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=xw5nxFWMlo

[10] Y. Wang, D. Guo, J. Chen, R. Zhang, Y. Ma, and Z. Zheng, "Rlcoder: Reinforcement learning for repository-level code completion," *CoRR*, vol. abs/2407.19487, 2024.

[11] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. M. V, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Moustafa-Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder: may the source be with you!" *CoRR*, vol. abs/2305.06161, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.06161

[12] V. Sharan, S. M. Kakade, P. Liang, and G. Valiant, "Prediction with a short memory," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, I. Diakonikolas, D. Kempe, and M. Henzinger, Eds. ACM, 2018, pp. 1074–1087. [Online]. Available: https://doi.org/10.1145/3188745.3188954

[13] S. Sun, K. Krishna, A. Mattarella-Micke, and M. Iyyer, "Do long-range language models actually use long-range context?" in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 807–822. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.62

[14] Y. Ding, Z. Wang, W. U. Ahmad, H. Ding, M. Tan, N. Jain, M. K. Ramanathan, R. Nallapati, P. Bhatia, D. Roth, and B. Xiang, "Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion," in *NeurIPS*, 2023.

[15] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *CoRR*, vol. abs/2009.10297, 2020.

[16] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, A. Yang, R. Men, F. Huang, X. Ren, X. Ren, J. Zhou, and J. Lin, "Qwen2.5-coder technical report," *CoRR*, vol. abs/2409.12186, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.12186

[17] GitHub Staff, "Octoverse: AI leads Python to top language as the number of global developers surges - The GitHub Blog," October 2024. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-2024/

[18] "Gharchive." [Online]. Available: https://www.gharchive.org/

[19] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, H. Alobeidli, A. Cappelli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon LLM: outperforming curated corpora with web data only," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/fa3ed726cc5073b9c31e3e49a807789c-Abstract-Datasets_and_Benchmarks.html

[20] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *CPM*, ser. Lecture Notes in Computer Science, vol. 1848. Springer, 2000, pp. 1–10.

[21] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*. ACM, 1998, pp. 604–613.

[22] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[23] "Scip." [Online]. Available: https://github.com/sourcegraph/scip

[24] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html

[25] X. Lai, Z. Tian, Y. Chen, S. Yang, X. Peng, and J. Jia, "Step-dpo: Step-wise preference optimization for long-chain reasoning of llms," *CoRR*, vol. abs/2406.18629, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.18629

[26] Y. Wu, Z. Sun, H. Yuan, K. Ji, Y. Yang, and Q. Gu, "Self-play preference optimization for language model alignment," *CoRR*, vol. abs/2405.00675, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.00675

[27] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan, "Qwen2 technical report," *CoRR*, vol. abs/2407.10671, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.10671

[28] "Trl." [Online]. Available: https://github.com/huggingface/trl

[29] "Deepspeed ai." [Online]. Available: https://www.deepspeed.ai/2021/03/07/zero3-offload.html

[30] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=mZn2Xyh9Ec

[31] M. Finlayson, J. Hewitt, A. Koller, S. Swayamdipta, and A. Sabharwal, "Closing the curious case of neural text degeneration," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=dONpC9GL1o

[32] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[33] D. Biderman, J. Portes, J. J. G. Ortiz, M. Paul, P. Greengard, C. Jennings, D. King, S. Havens, V. Chiley, J. Frankle *et al.*, "Lora learns less and forgets less," *Transactions on Machine Learning Research*, 2024.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[35] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, and et al., "Starcoder 2 and the stack v2: The next generation," *CoRR*, vol. abs/2402.19173, 2024.

[36] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[37] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021.

[38] D. Liao, S. Pan, X. Sun, X. Ren, Q. Huang, Z. Xing, H. Jin, and Q. Li, "$\mathbf{A^{3}}$a3-codgen: A repository-level code generation framework for code reuse with local-aware, global-aware, and third-party-library-aware," *IEEE Trans. Software Eng.*, vol. 50, no. 12, pp. 3369–3384, 2024.

[39] W. Cheng, Y. Wu, and W. Hu, "Dataflow-guided retrieval augmentation for repository-level code completion," in *ACL (1)*. Association for Computational Linguistics, 2024, pp. 7957–7977.

[40] L. A. Agrawal, A. Kanade, N. Goyal, S. K. Lahiri, and S. K. Rajamani, "Monitor-guided decoding of code lms with static analysis of repository context," in *NeurIPS*, 2023.

[41] Q. Guo, X. Li, X. Xie, S. Liu, Z. Tang, R. Feng, J. Wang, J. Ge, and L. Bu, "Ft2ra: A fine-tuning-inspired approach to retrieval-augmented code completion," in *ISSTA*. ACM, 2024, pp. 313–324.

[42] M. Sapronov and E. Glukhov, "On pretraining for project-level code completion," in *ICLR 2025 Third Workshop on Deep Learning for Code*, 2025. [Online]. Available: https://openreview.net/forum?id=t9RN9WX4Ic

[43] H. Zhao, J. Hui, J. Howland, N. Nguyen, S. Zuo, A. Hu, C. A. Choquette-Choo, J. Shen, J. Kelley, K. Bansal, L. Vilnis, M. Wirth, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, Z. Gong, J. Fine, T. Warkentin, A. J. Hartman, B. Ni, K. Korevec, K. Schaefer, and S. Huffman, "Codegemma: Open code models based on gemma," *CoRR*, vol. abs/2406.11409, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.11409