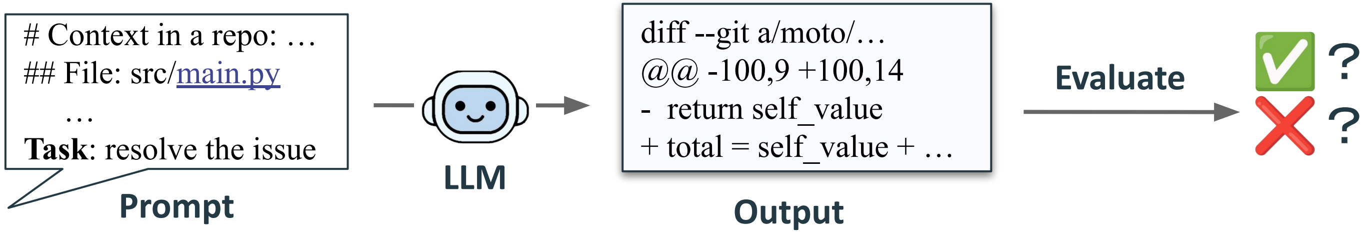
Synthetic Data Approaches for Coding Agent Training

Yiqing Xie

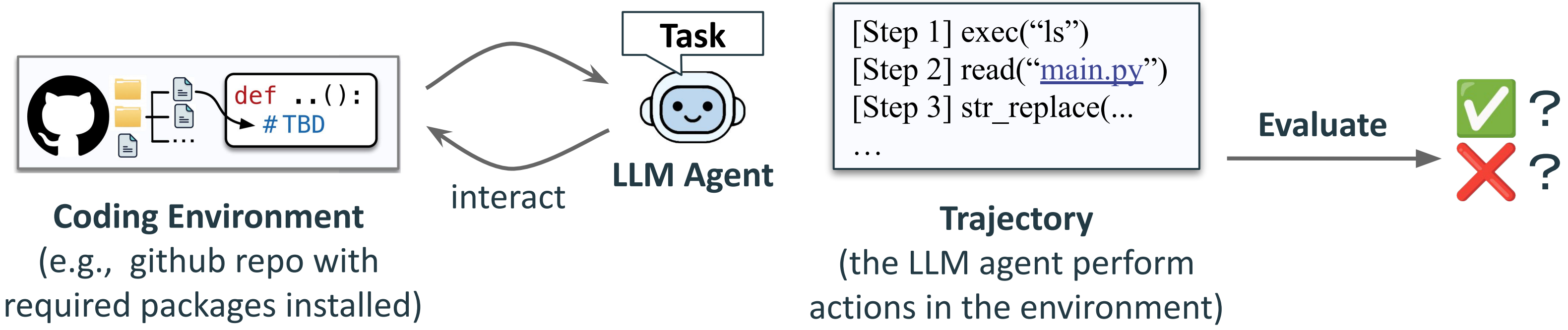
11-891: Neural Code Generation

Introduction: from Code Generation to Coding Agent

- Code-Gen training instance: <input, output, eval score>

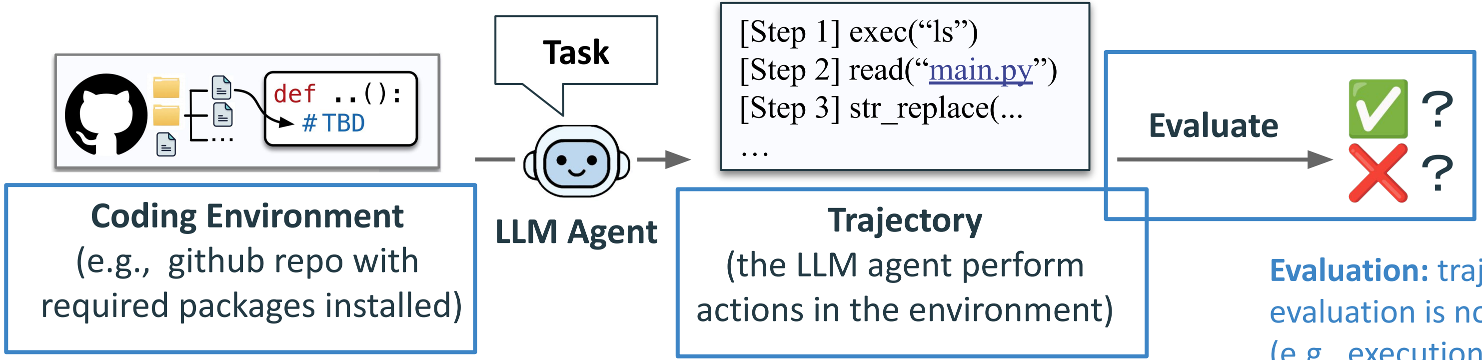


- Coding agent training instance: <task & environment, trajectory, eval score>



Challenge: Obtaining Agent Training Data

- It is non-trivial to obtain agent training data



Evaluation: trajectory evaluation is non-trivial (e.g., execution-based feedback)

Task & Agent environment: require large effort to build

Trajectory: difficult to obtain

- Synthesize model trajectories: non-trivial
- Model rollout: requires high cost

Related Work: SWE-Gym (10/21 Lecture)

- Task: issue-solving
  - SWE-bench-like instances: (1) executable repo; (2) issue with test cases
- Training instance construction
  - Input: issues that are equipped with test cases 🔴
  - Environment: developers manually install the repos for execution 👨‍💻👨‍💻
  - Outcome: <u>Real</u> issues; <u>Real</u> test cases

- Training recipe: rejection sampling finetuning
  - Rollout -> evaluate -> SFT with successful trajectories as targets
- Bottleneck: **scalability**
  - 2.4k instances, 12 repos, only 491 training trajectories

Our Solutions:
Scalable Agent Training Data Construction

- Environment & Evaluation:
  - [RepoST] Can we reduce the complexity of training environments?
- Task:
  - [Hybrid-Gym] Can we transfer from coding tasks that are easier to scale-up?
- Trajectory:
  - [CMTrans] Can we construct synthetic trajectories / training targets?

Our Solutions:
Scalable Agent Training Data Construction

- **Environment & Evaluation:**
  - **[RepoST] Can we reduce the complexity of training environments?**
- Task:
  - [Hybrid-Gym] Can we transfer from coding tasks that are easier to scale-up?
- Trajectory:
  - [CMTrans] Can we construct synthetic trajectories / training targets?

Related Work: R2E-Gym

- Task: issue-solving
  - Goal: environments with (1) executable repo; <u>(2) issue with test cases</u>
- High-level idea: synthetic test cases
  - It is possible to convert commits to issues
  - (with an executable repo) it is possible to apply LLMs to generate tests

Related Work: R2E-Gym

- Task: issue-solving
  - Goal: environments with (1) executable repo; (2) issue with test cases

- High-level idea: synthetic test cases
  - It is possible to convert **commits** to **issues**
    - Use an agent to convert commits to issues
      - Statement, failing tests, execution traces, …
  - (with an executable repo) it is possible to apply LLMs to **generate tests**
    - Use an agent to generate Fail-to-Pass tests for the commits

Related Work: R2E-Gym

- Task: issue-solving
- Training instance construction
  - Input: repos with a large number of commits 🟡
  - Environment: Agent & developer install the repo and write dockerfiles 🤖👨‍💻
  - Outcome: Real commits & Real + Synthetic test cases
    - Use agents to convert commits to issues & generate tests
- Training recipe: rejection sampling finetuning
- Improved scalability -> performance
  - 4.5k instances, 10 repos
  - 491 -> 3.3k training trajectories

Model Size	SWEBench-Verified Base-model	SWE-Gym	Ours	Δ
7B	1.8 (±1.3)	10.6 (±2.1)	**19.0** (±1.0)	+8.4
14B	4.0 (±1.6)	16.4 (±2.0)	**26.8** (±1.4)	+10.4
32B	7.0 (±1.3)	20.6 (±2.1)	**34.4** (±1.2)	+13.8

Related Work: R2E-Gym: Discussion Questions

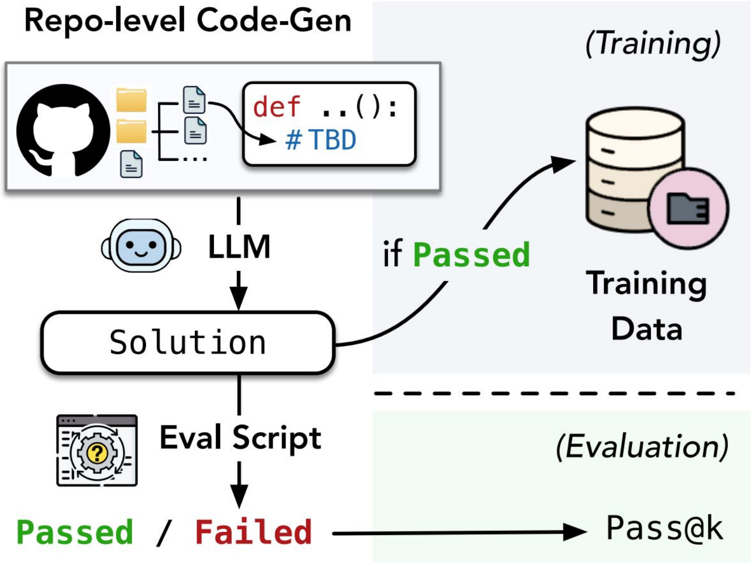
- Is data scale all you need?
  - [**Weihua**] How to balance the data quantity and quality?
  - [**Gavin**] Many work has looked at scaling dataset by collecting more data. It seems like much fewer work looks at how to improve the quality of the data or use them strategically.

RepoST: Environment Construction with Sandbox Testing

- We need executable environments for code generation tasks:
  - We need executability for environment feedback & for evaluation
- Bottlenecks for scalability:
  - Test case generation is not always trivial for LLMs 🟡
  - Building repos is challenging for both human and LLMs 🤖👨‍💻
    - One docker environment for each instance: not portable
- Intuition:
  - A repo may contain complicated functionality. If the goal is only to modify a small part of the repo, we do not need the executability of the whole repo
    - Can we reduce the complexity of training environments?

RepoST: Environment Construction with Sandbox Testing

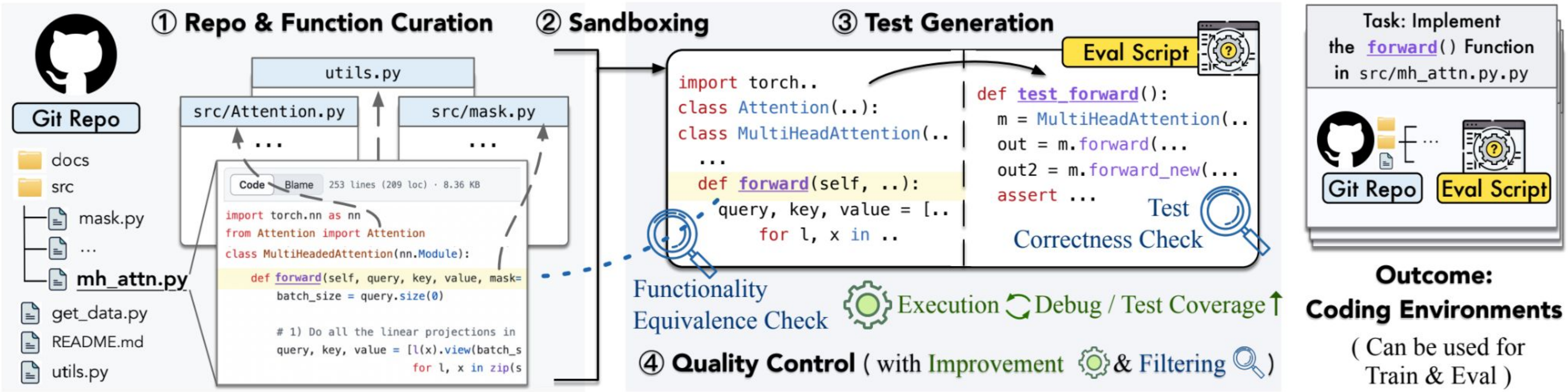
- Our method: Sandbox Testing
  - **Intuition**: we do not need the executability of the whole repo
  - **Code idea**: test the target function in a separate evaluation script
    - We use an LLM to (1) set up the dependencies and (2) write tests in the eval script
  - **Usage**: the agents can still access the orig repo, and they obtain execution feedback from the eval script. It is also used for trajectory evaluation.



Easier environment setup -> High scalability

RepoST: The Framework to construct Eval Scripts

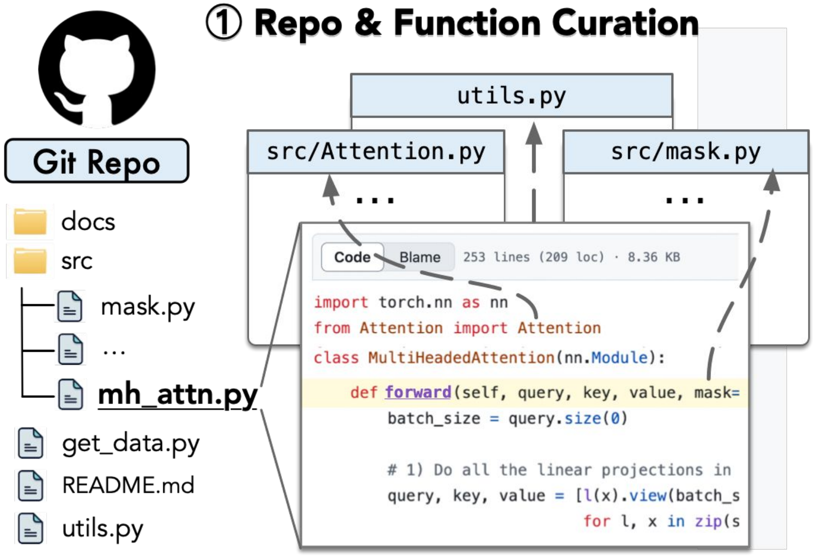
- The RepoST Framework
  - ① Repo & Function Curation
  - ② ③ Evaluation Script Creation (Sandboxing + Test Generation)
  - ④ Quality Control (Execution, Coverage, Functionality, Test Correctness)



RepoST: The Framework to construct Eval Scripts

[Step-①] Repo & Function Curation

- Randomly sample GitHub repo
  - Filters: License, date, size, …
  - Unlike previous methods, we do not require setup files or tests
- Sample functions & extract dependencies
  - Dependencies: modules directly/indirectly called by the function

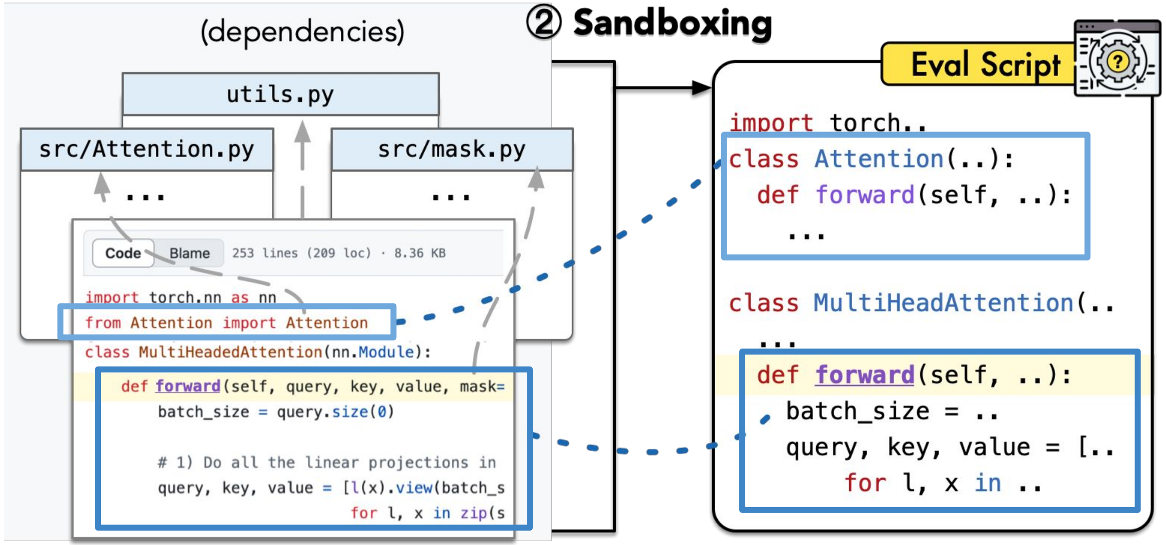


Target Function: MultiHeadedAttention.forward()

RepoST: The Framework to construct Eval Scripts

[Step-②] Sandboxing

- We prompt an LLM to aggregate the dependencies into one script
  - We only want to keep one evaluation script to avoid relative import issues



RepoST: The Framework to construct Eval Scripts

[Step-②] Sandboxing

- One challenging case:
  - External APIs and file reading
- We explicitly prompt the LLM to
  - Create **mock connections** for any external API, and
  - Create **mock strings** or write **example files** to a specific directory for file reading

Orig Function

```python
def API_call(prompt: str) -> str:
    return API_client.generate(
        model=API_MODEL, messages=[...]
    )
```

Eval Script

```python
class Mock_API(object):
  ...
  def generate(model, messages):
    return "mock_" + messages[0]['content']

API_MODEL = "model_name"
API_client = Mock_API("api_key")
def API_call(prompt: str) -> str:
    return API_client.generate(
        model=API_MODEL, messages=[...]
    )
...
```

RepoST: The Framework to construct Eval Scripts

[Step-③] Test Generation

- After aggregating the function and its dependencies to one script
- We prompt the LLM to generate tests using **equivalence testing**
  - Goal: compare the behavior of the model-generated & GT implementation
  - Do not need to predict the actual function output; a relative easy task

```
def test_indexer_from_folders_sigmf(self):
  result = indexer(self.test_dir.name)
  expected = ref_indexer(self.test_dir.name)

  self.assertEqual(len(result), len(expected))

  for res, exp in zip(result, expected):
     self.assertEqual(res[1].num_bytes, exp[1].num_bytes)
     self.assertEqual(res[1].byte_offset, exp[1].byte_offset)
```

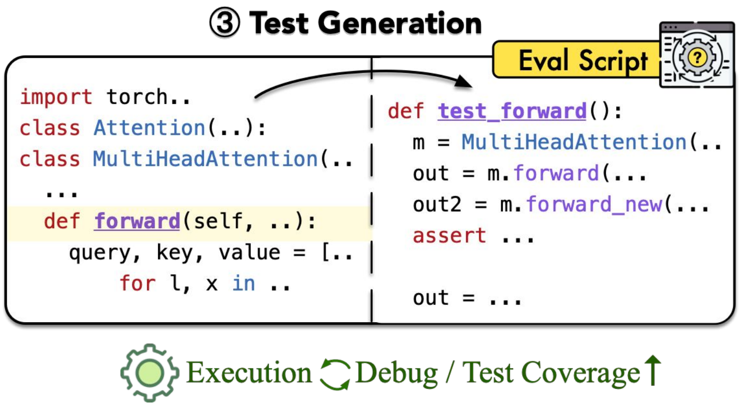
model-generated implementation

GT implementation

RepoST: The Framework to construct Eval Scripts

[Step-③] Test Generation

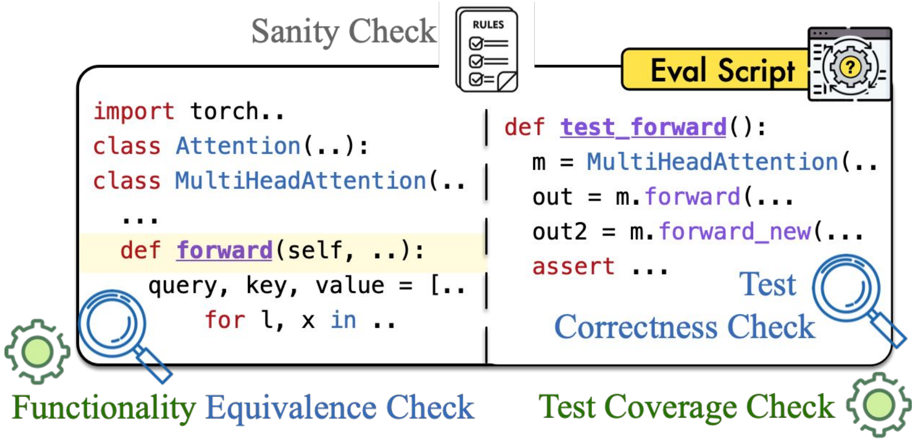
- In principle, the original target function should pass all test cases
- We iteratively execute the evaluation script and prompt the LLM to **debug** it (if needed)
- We **automatically install required packages** by reading `ModuleNotFound` errors
  - On average, one evaluation script only requires 2.1 libraries (26.5 for a repo)



RepoST: The Framework to construct Eval Scripts

[Step-④] Quality Control

- (**Sanity Checks**) the function should be called, tests should be called, … (rule-based check)
- (**Functionality Equivalence Check**) The sandboxed and the orig function should have same functionalities (AST-based & LLM-based check)
- (**Test Correctness & Coverage Check**) The test should be correct and have high coverage (Execution-based & LLM-based check)

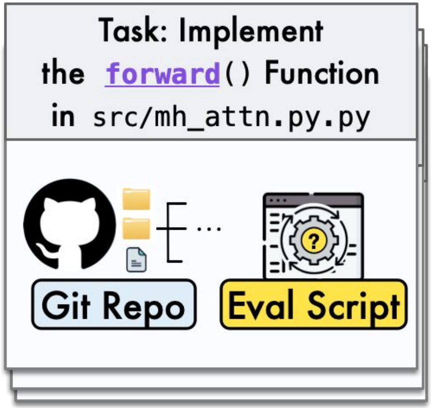


④ **Quality Control** (with Filtering)

(human study) Agreement between human and LLM-based check

RepoST: The Framework to construct Eval Scripts

- Outcome coding environments:
  - **Coding task**:
    - Function generation
      - Future work: adapt to other tasks
  - **Orig repo**:
    - Naturally occurring GitHub repos
      - We do not modify the orig repos
  - **Executable eval script**:
    - Provided feedback for the agent
    - Provide evaluation



Outcome:
Coding Environments
(Can be used for Train & Eval)

RepoST: Discussion Questions

- Environment construction
  - [**Sathwik**] combine LLM-based environment construction and static or dynamic program analysis (e.g., dependency graph)?
- Bias of the dataset
  - [**Yuxin**] While it successfully creates an executable environment, does it risk oversimplifying the problem?
  - [**Yi Wu**] How might this affect the generalizability of trained models to real-world repositories with unmockable dependencies, like hardware-specific APIs, and what alternative validation strategies could mitigate such risks?
  - [**Vincent**] How might this sandboxing approach systematically bias the dataset toward certain types of functions while excluding others?

RepoST: Resulting Datasets

Dataset Statistics

- By the release date, RepoST-Train is the **largest** repo-level code-gen dataset with test cases (to our knowledge)
    - R2E-Gym: 12 repos
    - SWE-Smith: 128 repos
- Compared to R2E, with the same repos as input, RepoST can create a much larger dataset

Dataset	#Examples	#Repo	Repo?	Auto?
HumanEval	164	–	✗	✗
DS1000	1,000	–	✗	✗
ClassEval	100	–	✗	✗
RepoEval-Func	455	6	✓	✗
SWE-Bench	2,294	12	✓	✗
CoderEval	230	43	✓	✗
EvoCodeBench	275	25	✓	✗
DevEval	1,874	117	✓	✗
SWE-Gym	2,438	11	✓	✗
R2E-Eval1	246	137	✓	✓
R2E (Our Input)	744	123	✓	✓
REPOST-TRAIN	**7,415**	**824**	✓	✓
REPOST-EVAL	296	99	✓	✓

RepoST: Resulting Datasets

- We are able to construct relatively **complex** examples
- We are able to construct datasets with **diverse libraries** (-> examples with diverse topics)
- The **% of standalone functions** is similar to the real-world distribution (27% @DevEval)
  - This means RepoST is not biased towards standalone functions
    - Although they are easy for sandboxing and test generation

	TRAIN	EVAL
Target Avg # Tokens (Lines)	112.4 (12.8)	102.7 (9.9)
Eval Script Avg # Tokens (Lines)	842.5 (75.7)	1217.5 (122.3)
Avg # Test Cases	5.7	8.2
Avg Test Branch Coverage	97.8%	100%
% Standalone Functions	28.1%	26.4%
# External Libraries	894	106

Table 2: Detailed statistics of our datasets.

RepoST: Training with our Dataset

- Training Setup
  - Base models: StarCoder2-7B, Qwen2.5-Coder-7B
  - Datasets: HumanEval, RepoEval-func, RepoST-Eval
  - Baselines:
    - **Zero-shot**
    - Vanilla Supervised Finetuning (**SFT**)
      - (code context, GT target function)
    - Rejection Sampling Finetuning **(RFT)** w/ distillation
      - (code context, GPT-4o/Claude-3.5 generated correct function)
    - Rejection Sampling Finetuning **(RFT)** w/ self-training
      - (code context, self-generated correct function)

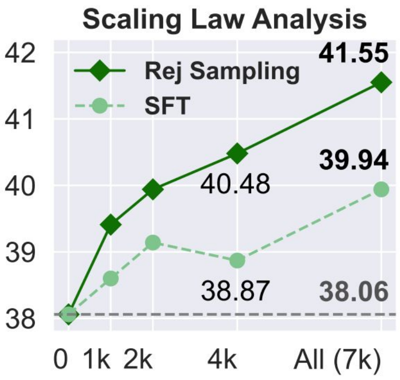
RepoST: Training with our Dataset

- **RFT (Distill) > RFT (Self) > SFT > Zero-Shot**
- The training can improve both repo-level code-gen and algorithm problems (HumanEval)
- The performance gain on RepoEval-Func and RepoST-Eval are similar

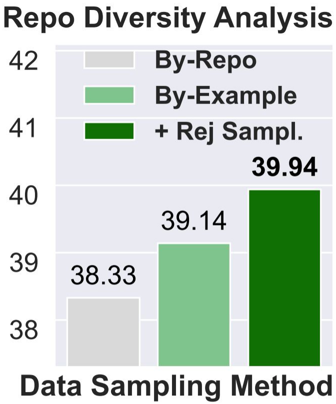
Model	HumanEval		RepoEval-Func		RepoST-Eval	
	Pass@1	Δ	Pass@1	Δ	Pass@1	Δ
StarCoder2-7B (Lozhkov et al., 2024)	34.76	–	32.98	–	26.35	–
+ SFT	37.20	↑2.44	33.78	↑0.80	27.70	↑1.35
+ RFT (Self)	39.63	↑4.87	34.58	↑1.61	28.38	↑2.03
+ RFT (Distill)	**40.24**	**↑5.49**	**35.12**	**↑2.14**	**29.05**	**↑2.70**
Qwen2.5-Coder-7B (Hui et al., 2024)	79.27	–	38.06	–	29.39	–
+ SFT	80.48	↑1.21	39.94	↑1.88	30.74	↑1.35
+ RFT (Self)	**84.76**	**↑5.49**	40.75	↑2.69	31.76	↑2.36
+ RFT (Distill)	**84.76**	**↑5.49**	**41.55**	**↑3.49**	**32.43**	**↑3.04**

RepoST: Training with our Dataset

- Scaling Law Analysis
  - Data scale matters
  - We still have RFT > SFT under different scales
- Repo Diversity Analysis
  - Method 1: randomly sample 2k examples
  - Method 2: sample repos, include all functions in the repo, until # Examples reach 2k
  - Method 2 (RFT) > Method 2 (SFT) > Method 1 (SFT)
    - Training on diverse repos matters



(a) Scaling law analysis.



(b) Repository diversity.

RepoST: Conclusions

- Goal: constructing executable environments in a more scalable way
- Intuition:
  - A repo may contain complicated functionality. If the goal is only to modify a small part of the repo, we do not need the executability of the whole repo
- Solution: Sandbox Testing
  - We only test the target part of the code in an executable script
- Results
  - Larger training datasets
  - Executability in training set matters; Scale matters; Diversity of training data matters

Our Solutions:
Scalable Agent Training Data Construction

- Environment & Evaluation:
    - [RepoST] Can we reduce the complexity of training environments?
- **Task:**
    - **[Hybrid-Gym] Can we transfer from coding tasks that are easier to scale-up?**
- Trajectory:
    - [CMTrans] Can we construct synthetic trajectories / training targets?

Related Work: SWE-Smith

- Task: issue-solving
  - Goal: environments with (1) executable repo; (2) issue with test cases

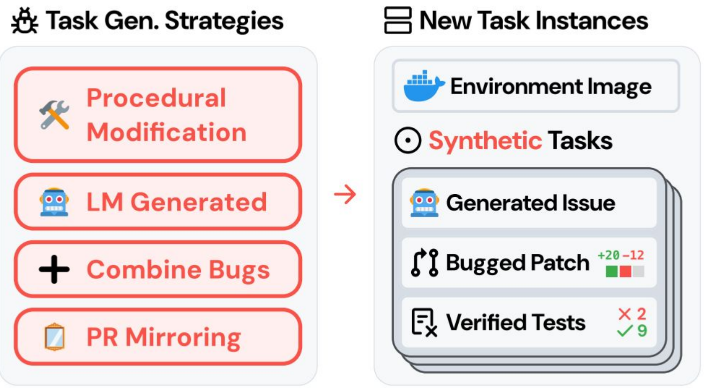
- High-level idea: given repos with test, construct synthetic issues
  - Input: repos with test cases
  - Use an agent to construct synthetic issues that break one or more tests

Related Work: SWE-Smith

- Task: issue-solving
- High-level idea: given repos with test, construct synthetic issues

- Construct synthetic bugs & issues
  - LLM-based bug construction
  - AST-based bug construction
  - Combine Bugs from the same file/module
  - Construct PRs based on the bugs

- Leveraging existing tests
  - Only keep patches that break a test



Related Work: SWE-Smith

- Task: issue-solving
- Training instance construction
  - Input: repos with test cases 🟡
  - Environment: Agent & developer install the repo and write dockerfiles 🤖👨‍💻
  - Outcome: Synthetic issues & Real test cases
- Training recipe: rejection sampling finetuning
- Improved scalability: 50k instances, 128 repos, 5k training trajectories

Model	System	Train Size	Lite	Verified
Lingma-SWE-GPT-72B (Ma et al., 2024)	SWE-SynInfer	-	-	28.8
Qwen3-235B-A22B (Qwen et al., 2025)	OpenHands	-	-	34.4
R2E-Gym-32B (Jain et al., 2025)	OpenHands	3.3k	-	34.4
SWE-fixer-72B (Xie et al., 2025a)	SWE-Fixer	110k	24.7	32.8
SWE-gym-32B (Pan et al., 2024)	OpenHands	491	15.3	20.6
SWE-agent-LM-7B	SWE-agent	2k	11.7	15.2
SWE-agent-LM-32B	SWE-agent	5k	**30.7**	**40.2**

Related Work: SWE-Smith: Discussion Questions

- **Generalizability from synthetic issues to real ones**
  - [**Nicole**] what are some limitation for SWE-smith, since real world senario might be more complex than the bug-generation strategies (LM rewrite, AST edit, PR inversion, and combination)
  - [**Jack**] Since SWE-smith relies on a synthetically generated bug dataset, how well can SE agents trained on it generalize to real-world software development scenarios?
  - [**Gaokai**] Are the bug generated reasonable enough that they might actually exist in reality?
  - [**Karen**] How representative are the task they generate compared to real world tasks?

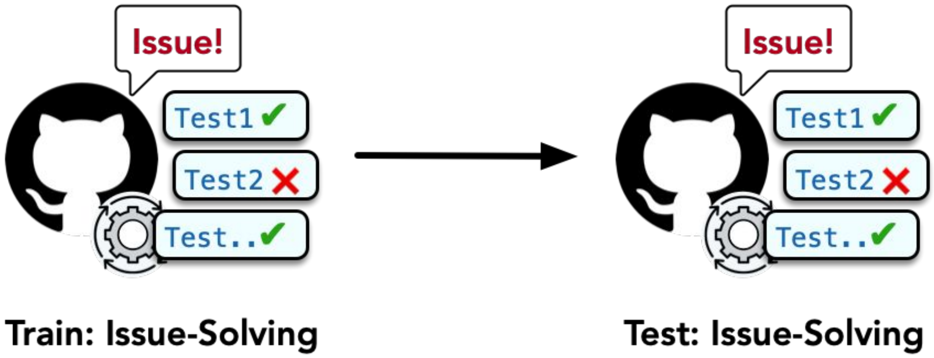
Related Work: SWE-Smith: Discussion Questions

- **Generalizability from synthetic issues to real ones**
  - [**Nicole**] what are some limitation for SWE-smith, since real world senario might be more complex than the bug-generation strategies (LM rewrite, AST edit, PR inversion, and combination)
  - [**Jack**] Since SWE-smith relies on a synthetically generated bug dataset, how well can SE agents trained on it generalize to real-world software development scenarios?
  - [**Gaokai**] Are the bug generated reasonable enough that they might actually exist in reality?
  - [**Karen**] How representative are the task they generate compared to real world tasks?

  - [**Weiwei**] Any other synthetic SWE tasks can be created from repo to increase diversity?

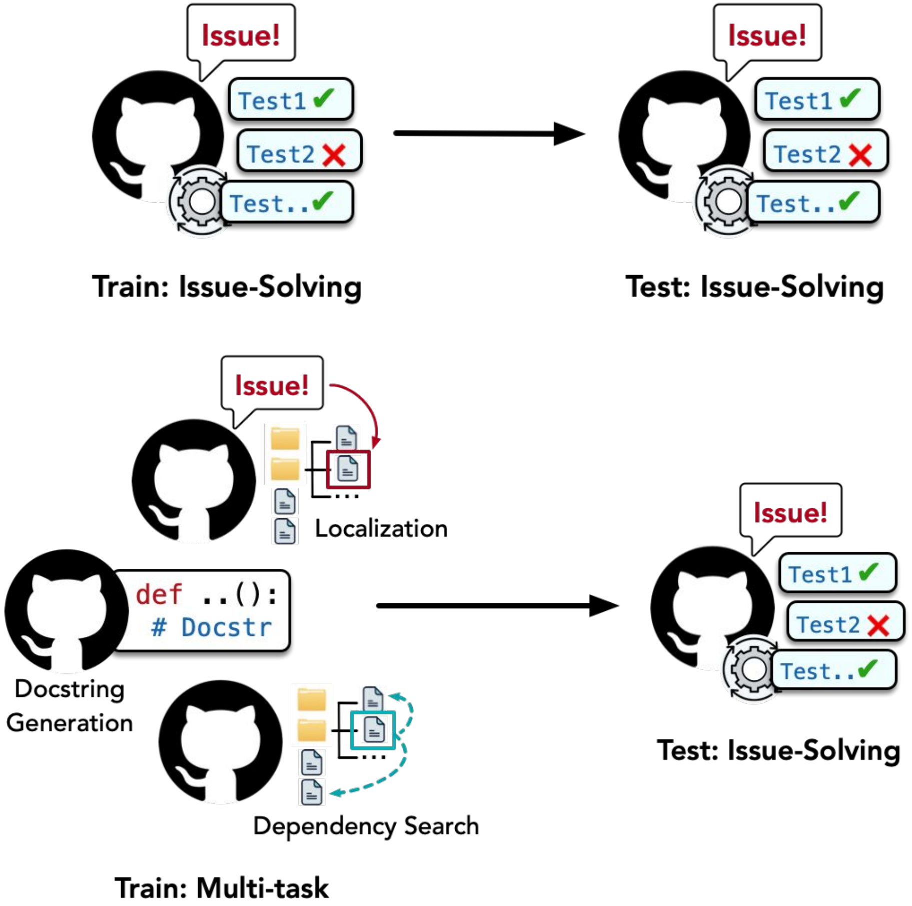
Hybrid-Gym (ongoing work): Motivation

- Background: Previous work mainly focuses on in-domain SFT
  - It's complicated to create an issue-solving environment
    - Each instance typically requires (1) a repo with all packages installed; (2) executable test cases for the target issue
  - Researchers are trying to improve the scalability of training data



Hybrid-Gym (ongoing work): Motivation

- Previous work mainly focuses on in-domain SFT
  - It's complicated to create an issue-solving environment
- **Motivation: To what extent can we transfer from tasks that only require lightweight environment?**



Hybrid-Gym (ongoing work): Task Design

- What tasks to transfer from?
  - Requirement: only need lightweight environment
  - First attempt: the basic abilities required for the downstream task
    - Tool usage; Repo exploration; Code generation w/ Execution

- Initial tasks
  - String replacement (tool usage)
  - Docstring generation (repo exploration & tool usage)
  - LiveCodeBench-Repo (code generation w/ execution & tool usage)
  - Localization (repo exploration & tool usage)

Hybrid-Gym (ongoing work): Initial Tasks

- Results: training on easy tasks can transfer to the complicated issue-solving task
  - Qwen2.5Coder zero-shot: 1 resolved / 2 localized (file) / 4 non-empty (out of 50)
  - + LiveCodeBench-repo SFT: 1 resolved / 3 localized (file) / 19 non-empty
  - + Str-Replace SFT: 2 resolved / 4 localized (file) / 21 non-empty
  - + Doc-Gen SFT: 5 resolved / 15 localized (file) / 22 non-empty
  - + Localization SFT: 11 resolved / 22 localized (file) / 23 non-empty
  - (baseline) + SWE-Gym SFT: **12 resolved / 37 localized (file) / 41 non-empty**

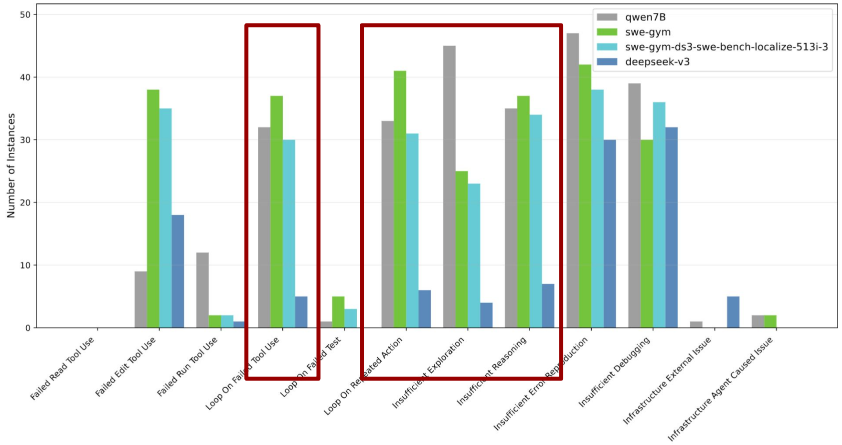
Hybrid-Gym (ongoing work): Initial Tasks

- Results:
    - Qwen2.5Coder zero-shot: 1 resolved / 2 localized (file) / 4 non-empty (out of 50)
    - + LiveCodeBench-repo SFT: 1 resolved / 3 localized (file) / 19 non-empty
    - + Str-Replace SFT: 2 resolved / 4 localized (file) / 21 non-empty
    - + Doc-Gen SFT: 5 resolved / 15 localized (file) / 22 non-empty
    - + Localization SFT: 11 resolved / 22 localized (file) / 23 non-empty

- Guidelines for task design
    - Str-Replace-SFT < Doc-Gen-SFT < Localization-SFT: **task difficulty matters**?
    - LiveCodeBench-repo SFT doesn't work: **repo exploration is a must**?

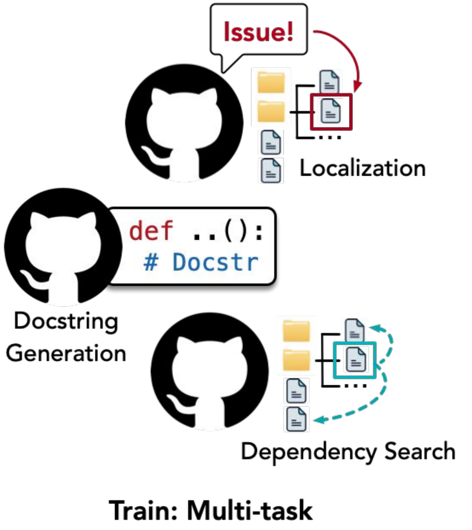
Hybrid-Gym (ongoing work): Initial Tasks

- Error analysis
  - We use an LLM to identify the types of errors for zero-shot/SFT/GT trajectories



- Guidelines for task design
  - Str-Replace-SFT < Doc-Gen-SFT < Localization-SFT: **task difficulty matters**?
  - LiveCodeBench-repo SFT doesn't work: **repo exploration is a must**?
  - Categories where the SFT models are still much weaker than GT: **Failed edit tool use & Loop on repeated action** (esp. failed tool use) -> room for improvement?

Hybrid-Gym (ongoing work): More Tasks!



- Guidelines for task design
  - Repo exploration is a must
  - Task complexity matters
  - Tool usage could be a direction for improvement
  - (our motivation) only require lightweight environments
    - [**Setup-1**] Executable Repo & Issues with test cases (issue-solving (most baselines))
      - SWE-Smith and R2E-Gym use agents to construct synthetic issues
    - [**Setup-2**] Executable Repo (func-generation, test-generation, result-replication, …)
    - [**Setup-3**] Any Repo & Execution-based evaluation (RepoST func-generation)
    - [**Setup-4**] Any Repo (doc-gen, localization-no-exec, dependency search, …)

Better Scalability

Hybrid-Gym (ongoing work): Recruiting!

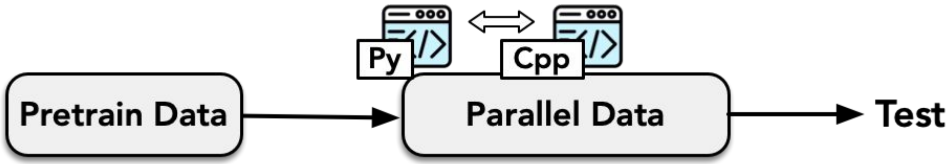
- Recruiting: Authorship for task implementation
    - We will propose some tasks but other (reasonable) proposals are welcomed
        - Initial tasks: dependency search, function generation
        - We will provide the detailed description of tasks an a step-by-step guidance for implementation
    - Task implementation
        - [Step 1] install & use OpenHands
        - [Step 2] Implement the prompt, environment set up, evaluation
        - We'll give authorship if you make substantial contributions
    - Tutorial & co-working sessions
        - We will hold co-working sessions once or twice each week
    - If you're interested, send us an email (Yiqing & Daniel)

Our Solutions:
Scalable Agent Training Data Construction

- Environment & Evaluation:
  - [RepoST] Can we reduce the complexity of training environments?
- Task:
  - [Hybrid-Gym] Can we transfer from coding tasks that are easier to scale-up?
- **Trajectory:**
  - **[CMTrans] Can we construct synthetic trajectories / training targets?**

CMTrans: Data Augmentation for Code Translation

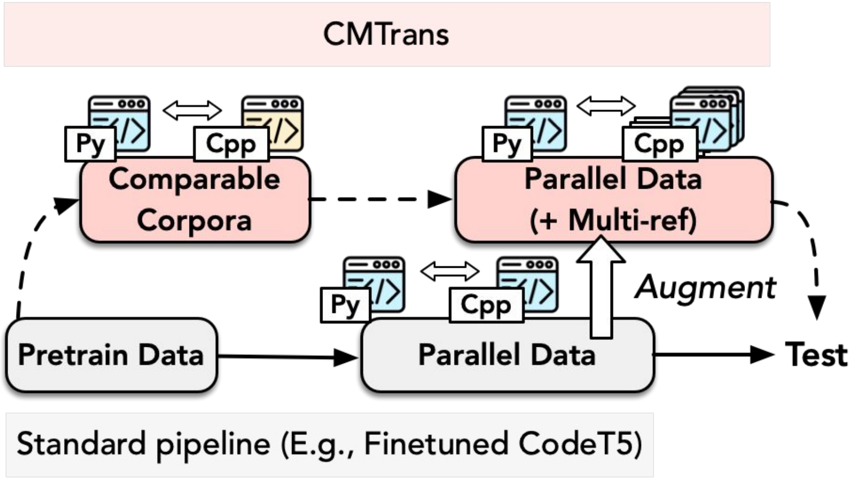
- Setting: Code generation
  - The high-level ideas could be extended to coding agent training
- Challenge: parallel data is limited
  - Standard pipeline: Pretraining -> SFT on parallel data
  - It's non-trivial to obtain pairs of parallel data with the exactly same functionality
    - In analogy to coding agent: it's non-trivial to obtain successful trajectories



CMTrans: Data Augmentation for Code Translation

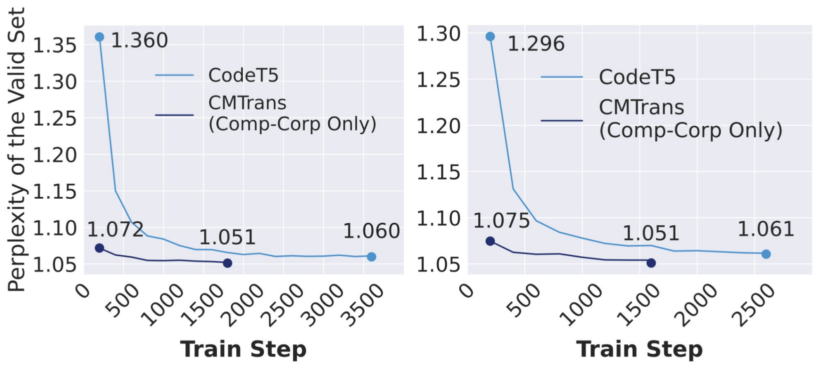
- Motivation: parallel data is limited
  - Standard pipeline: Pretraining -> SFT on parallel data

- Our solution: synthetic data pairs
  - **Comparable code pairs**: SFT with <source code, target code> pairs that do not strictly match
  - **Additional references**: generate additional target translations by test generation



CMTrans: [Method 1] SFT with Comparable Pairs

- Motivation: The model still has trouble generating fluent target-language code
  - The perplexity of the target translation is still high for zero-shot translation
  - In analogy to coding agent: the LLM still has trouble with the agentic format (e.g., making actions based on the environment feedback)

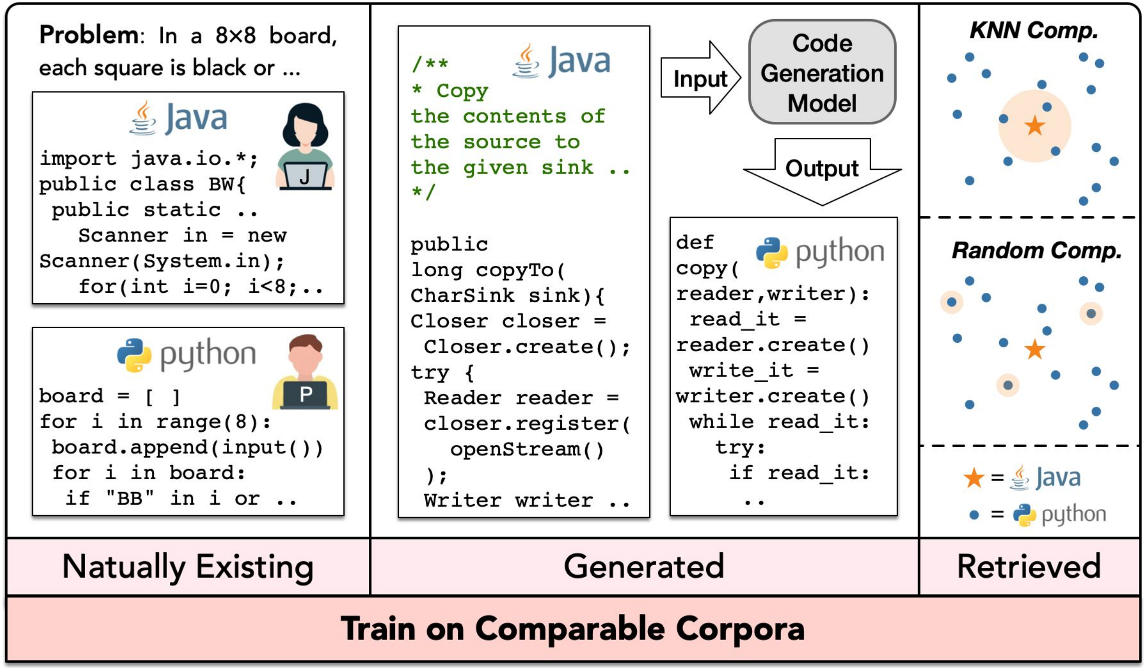


(a) Java-to-Python (b) Python-to-Java

Figure 5: Perplexity of validation set during finetuning.

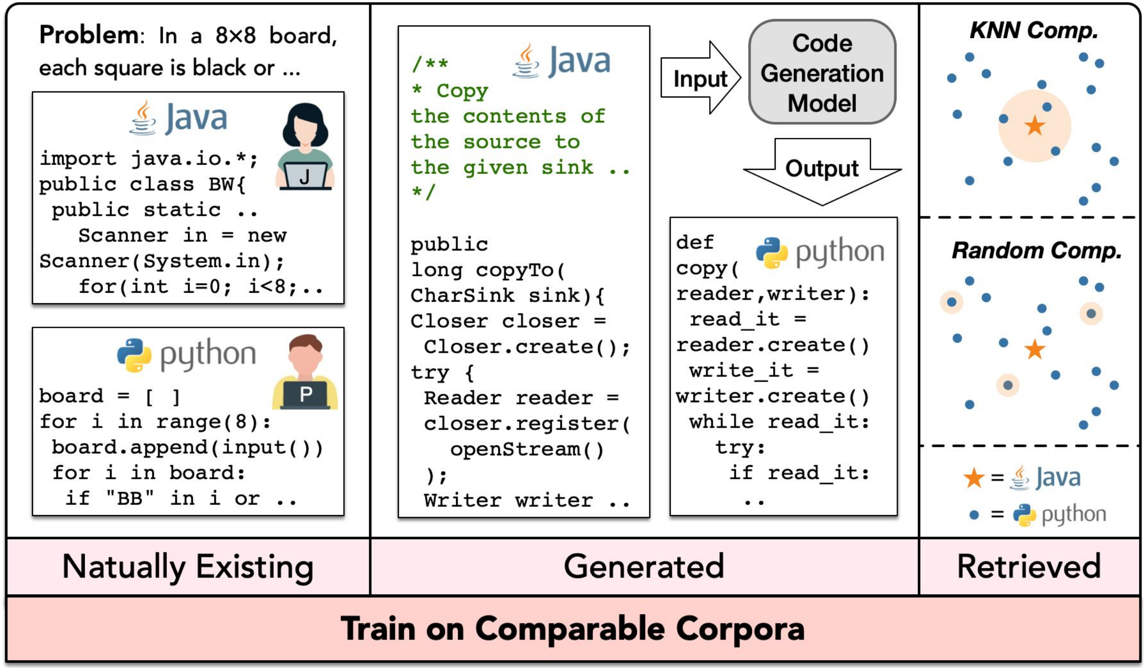
CMTrans: [Method 1] SFT with Comparable Pairs

- Intuition: we need to train the model to generate target-language-code
- We construct <source, target> pairs where the functionalities do not exactly match



CMTrans: [Method 1] SFT with Comparable Pairs

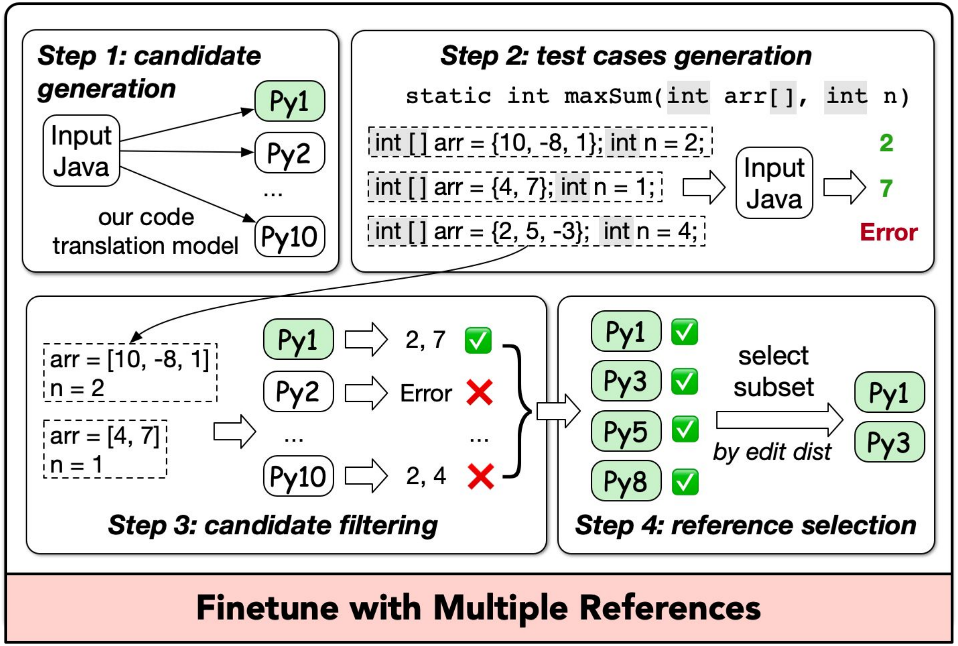
- Intuition: we need to train the model to generate target-language-code
  - In analogy to coding agent: SFT on synthetic trajectories?
  - E.g., where the feedback for the actions just "seems correct"



CMTrans: [Method 2] SFT with Additional References

- Motivation: SFT on diverse targets teaches the model functionality equivalence
  - We indirectly teach the model that the targets have the same functionality

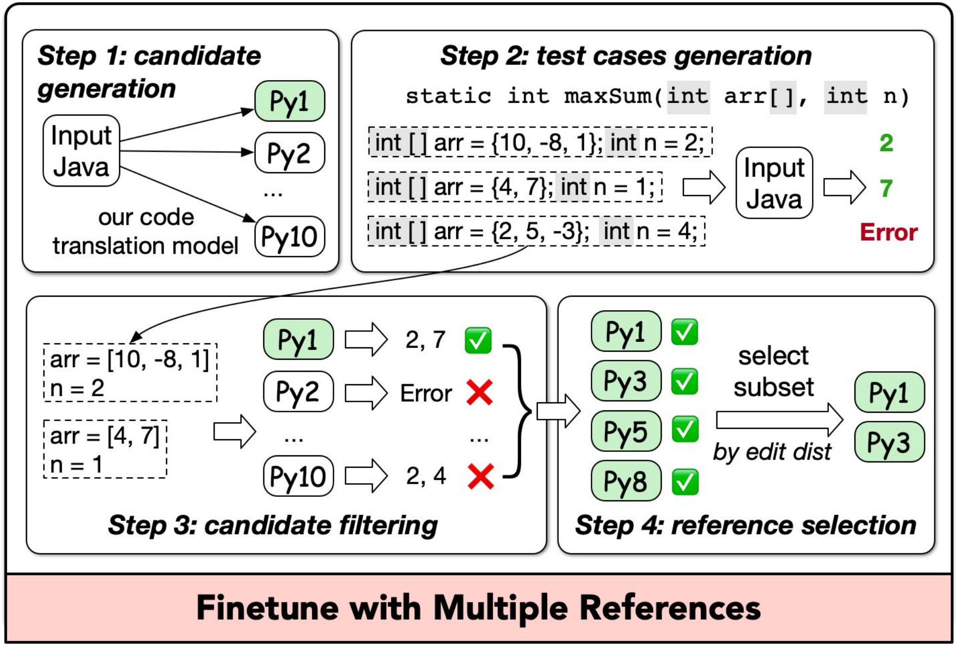
- Obtain additional targets
  - Test case generation
  - Rollout + filtering



CMTrans: [Method 2] SFT with Additional References

- Obtain additional targets
  - Test case generation
  - Rollout + filtering

  - In analogy to coding agent: can we construct more successful trajectories based on existing ones?
    - Synthetic or by rollout



CMTrans: Experiments on TransCoder-Test

- Results on TransCoder-Test: Java, Python, C++ pairwise translation
  - CMTrans outperforms the best baseline: TransCoder-ST-ft
  - CMTrans outperforms CodeT5 by >10%, which shares the same pretraining stage

Model ↓	**Java-to-Python**			**Python-to-Java**			**Avg of 6 Pairs**		
	BLEU	**CB**	**CA@1**	**BLEU**	**CB**	**CA@1**	**BLEU**	**CB**	**CA@1**
TransCoder (Roziere et al., 2020)	72.4	67.9	49.1	65.4	70.7	35.7	72.0	75.0	51.7
DOBF (Lachaux et al., 2021)	72.2	67.5	52.2	67.7	71.2	44.4	—	—	—
TransCoder-ST (Roziere et al., 2022)	73.1	68.7	68.5	70.0	71.9	58.1	71.3	74.9	66.3
CodeBERT (Feng et al., 2020)	52.0	48.9	10.4	45.4	45.0	4.2	—	—	—
CodeT5 (Wang et al., 2021)	79.4	72.5	61.0	79.0	75.9	52.7	<u>83.6</u>	80.0	62.6
PLBART (Ahmad et al., 2021a)	<u>79.9</u>	<u>73.2</u>	68.9	80.5	76.8	57.5	—	—	—
TransCoder-ST-ft (Roziere et al., 2022)	79.3	72.9	<u>69.4</u>	<u>81.4</u>	<u>78.4</u>	<u>62.0</u>	81.8	<u>80.2</u>	<u>67.6</u>
CMTrans	**80.1**	**74.2**	**73.5**	**84.3**	**82.1**	**66.0**	**84.9**	**82.0**	**70.1**

CMTrans: Experiments on TransCoder-Test

- Analysis 1: Syntax Accuracy
  - Zero-shot: 1.7
  - + Comparable pairs: **56.4**
  - + Parallel pairs: 69.7
  - + Comparable & Parallel pairs: **76.4**
  - Training on comparable pairs improves syntax accuracy, even combined with parallel data

Model ↓	J-to-P SA	P-to-J SA
No finetuning on parallel pairs		
CodeT5 (Wang et al., 2021)	1.1	1.7
+ Random Comp-Corp	20.0	41.3
+ KNN Comp-Corp	22.0	**56.4**
+ Generated Comp-Corp	**41.4**	43.2
+ Natural Comp-Corp	34.1	54.6
With finetuning on parallel pairs		
CodeT5 (Wang et al., 2021)	95.3	69.7
+ Random Comp-Corp	96.3	69.7
+ KNN Comp-Corp	96.3	71.0
+ Generated Comp-Corp	**97.6**	71.2
+ Natural Comp-Corp	97.4	**76.4**

Table 6: Syntax Accuracy (SA) on TransCoder-test before finetuning, which evaluates whether the program can be compiled without syntax errors.

CMTrans: Experiments on TransCoder-Test

- Analysis 2: space of correct solutions
  - After SFT with multi-references, the model learns to generate more unique correct solutions
  - In analogy to coding agent: SFT with augmented trajectories -> learn to explore a more diverse and reasonable action space?

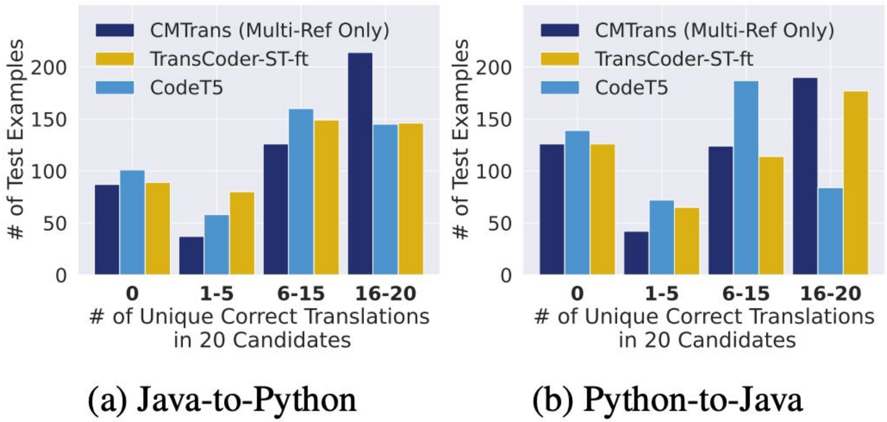


Figure 6: Number of unique correct translations in 20 candidates for each test example. We use beam search for each method, so the generated candidates are guaranteed to be distinct.

CMTrans: Conclusions

- Motivation: Lack of parallel code pairs as training data
- CMTrans: two data augmentation techniques
  - SFT with comparable code pairs (i.e., code pairs that do not strictly match)
  - SFT with additional references
- Analogy to coding agent training
  - Does the target trajectories need to be perfect? (E.g., can we construct trajectories where the feedback for the actions just “seems correct”?)
  - Can we construct more successful trajectories based on existing ones?