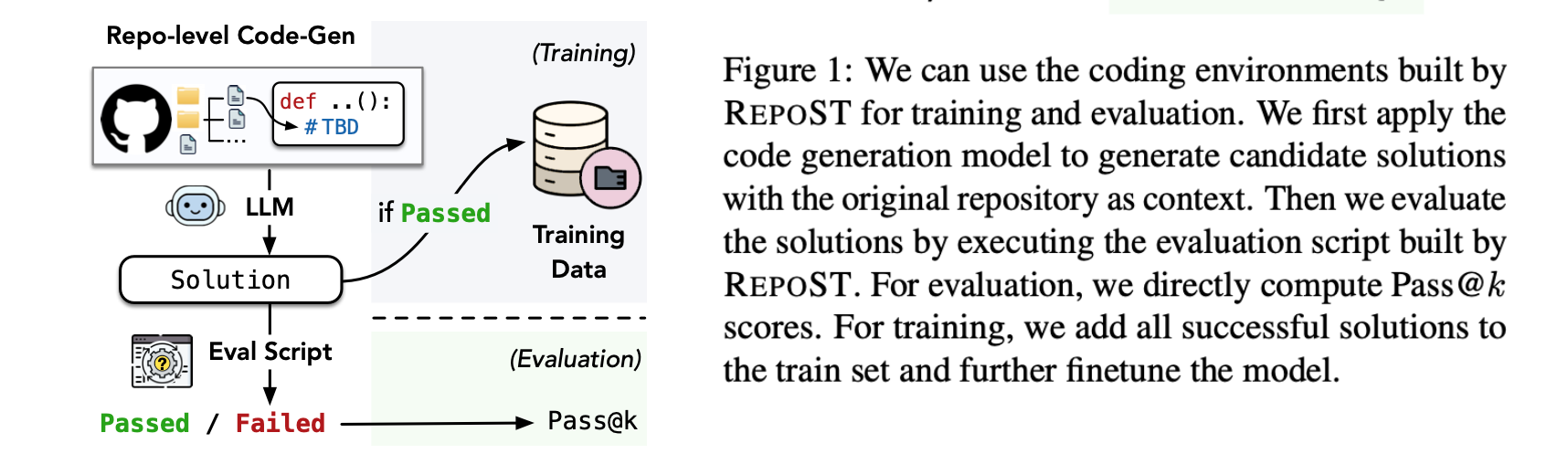
## RepoST: Scalable Repository-Level Coding Environment Construction with Sandbox Testing

#### Yiqing Xie1, Alex Xie1, Divyanshu Sheth1, Pengfei Liu2, Daniel Fried1, Carolyn Rose1

#### 1Carnegie Mellon University, 2Shanghai Jiao Tong University



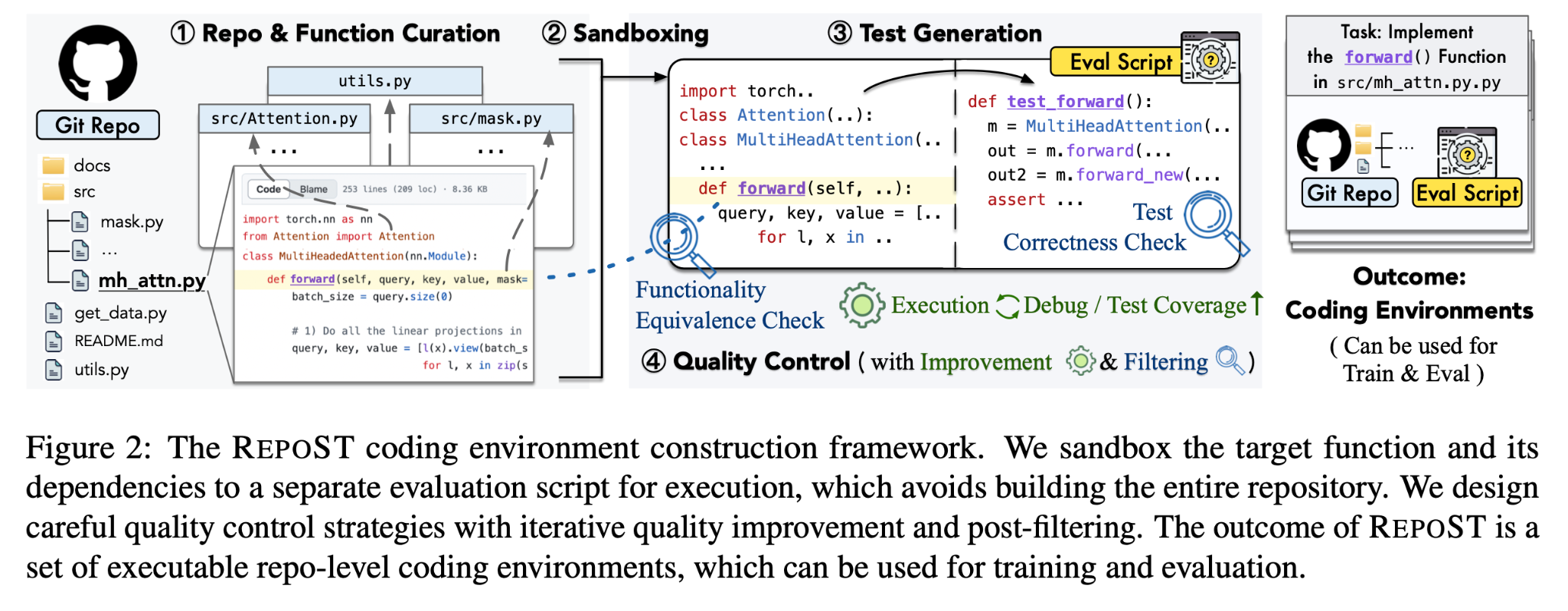
### Abstract

We present RepoST, a scalable method to construct environments that provide execution feedback for repository-level code generation for both training and evaluation. Unlike existing works that aim to build entire repositories for execution, which is challenging for both human and LLMs, we provide execution feedback with sandbox testing, which isolates a given target function and its dependencies to a separate script for testing. Sandbox testing reduces the complexity of external dependencies and enables constructing environments at a large scale. We use our method to construct RepoST-Train, a large-scale train set with 7,415 functions from 832 repositories. Training with the execution feedback provided by RepoST-Train leads to a performance gain of 5.5% Pass@1 on HumanEval and 3.5% Pass@1 on RepoEval. We also build an evaluation dataset, RepoST-Eval, and benchmark 12 code generation models.

### The RepoST Framework

**RepoST** is an automated framework that constructs **repo**-level coding environments using **S**andbox **T**esting. Specifically, given a function in a GitHub repository, we sandbox the function and its local dependencies to a separate script and generate tests with an LLM.

* **Original GitHub Repo as Context**: As shown in Figure 1, the models generate the target function with the entire GitHub repository as context. We then use the evaluation script to obtain execution feedback.
* **High Scalability**: Compared to integration testing used by previous datasets, we highlight the benefits of sandbox testing in constructing scalable coding environments: (1) In general, the external dependencies of a function are typically much simpler than a repository. By isolating the function and its local dependencies, we can execute the function by only installing the necessary packages. (2) If any execution error occurs, we can debug the separate scripts without modifying the original repository.
* **Carefully-Designed Quality Check Strategies**: we iteratively resolve environment or runtime errors and improve test coverage. We also conduct execution-based, AST-based, and LLM-based quality checks and only keep examples where the functionality of the sandboxed function does not alter and the tests are valid, reasonable, and have high coverage.



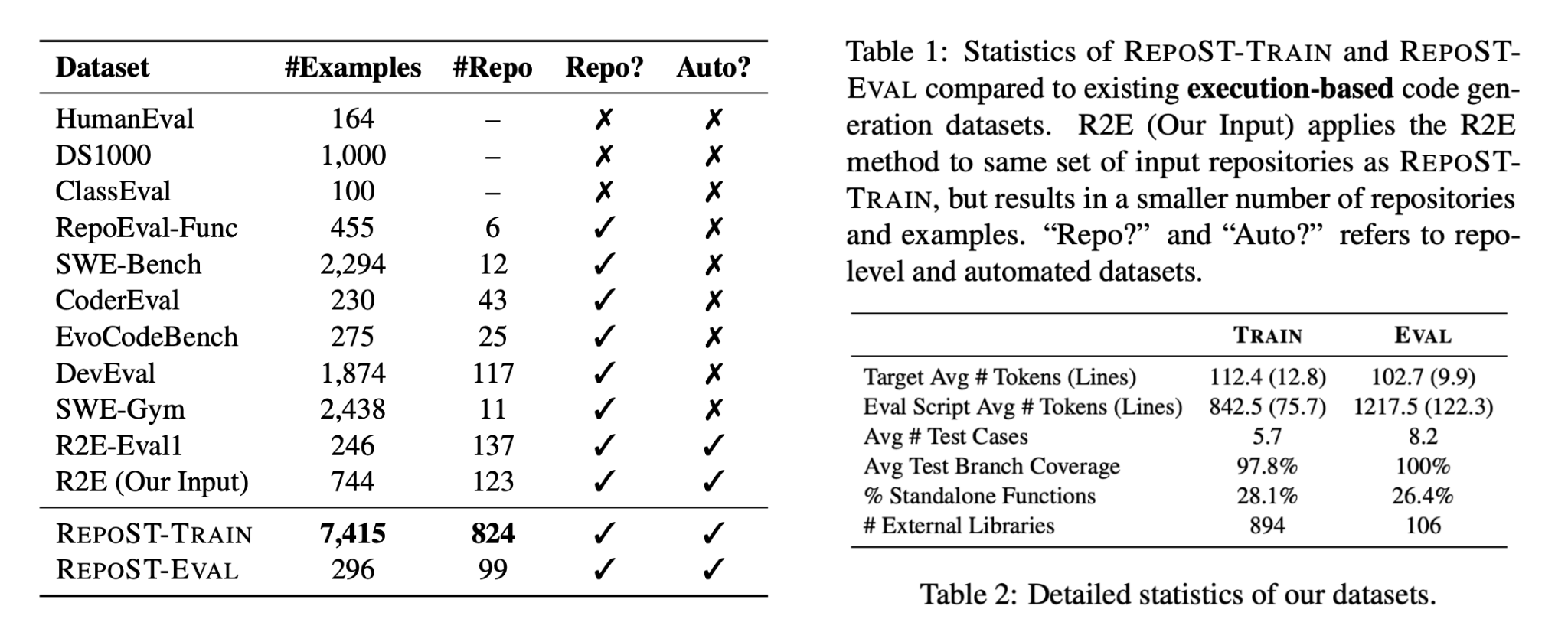
### Constructing Training / Evaluation Environments with RepoST

With our framework, we build a train set and an evaluation set: RepoST-Train and RepoST-Eval.

* To our knowledge, **RepoST-Train** is currently the largest repo-level code generation dataset with execution support, with 7,415 functions sampled from 824 repositories. The large scale enables training on RepoST-Train and evaluating on other benchmarks such as RepoEval or HumanEval.
* **RepoST-Eval** contains 296 functions sampled from 99 repositories. Because RepoST is fully automated, it can be potentially used to construct live benchmarks to avoid contamination issues.

We conduct careful quality verification with two human studies.

* The first human study demonstrates the **agreement** between human and LLM-based quality checkers, which indicates that after applying our filtering strategies for quality control, the remaining examples have high quality.
* The second human study checks whether the examples are **solvable** by human. Results show that 81.5% of the examples were solved by human, indicating that most examples are reasonable and not too complicated.

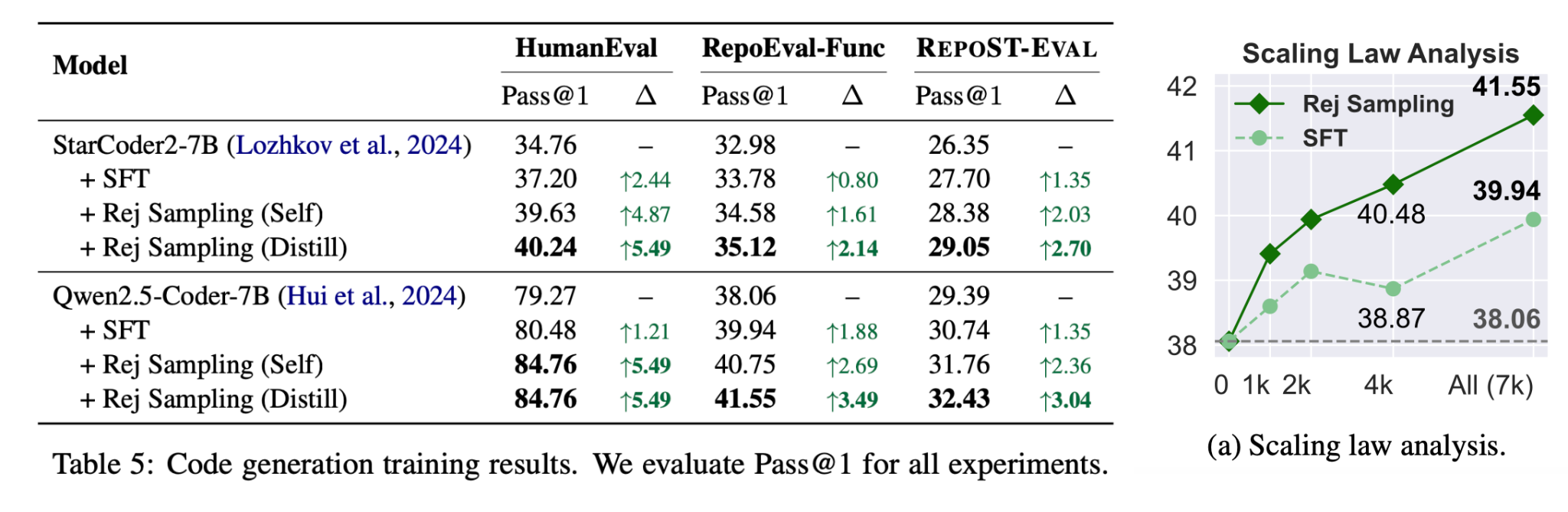


### Experiments: Training with RepoST-Train

With RepoST-Train, we can first train the model with supervised finetuning (**SFT**), with the code context as the input and the ground truth target function as the output. The execution feedback provided by our RepoST evaluation scripts allows us to employ the **rejection sampling** algorithm and further finetune the model on correct model-generated solutions.

We train our model with RepoST-Train and evaluate on HumanEval, RepoEval-Func, and RepoST-Eval. Results show that:

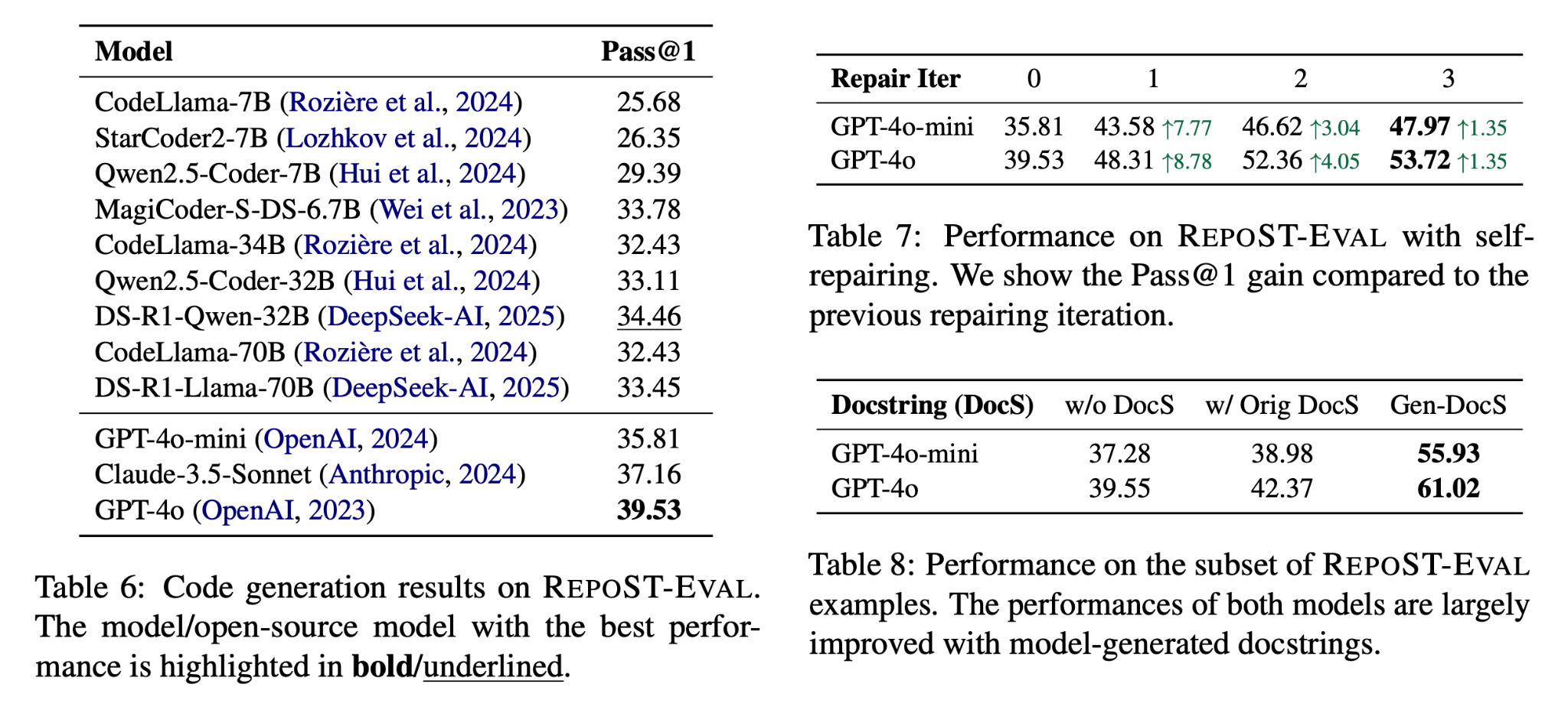
* Models trained with RepoST-Train can generalize well to other public benchmarks, including algorithm problems and repo-level code generation datasets. This demonstrates the effectiveness of constructing SFT training data.
* Training with rejection sampling, even with self-training only, achieves better performance than vanilla SFT. This shows the benefit of training environments that can provide execution feedback.
* The performance of Rej Sampling (Distill) increases as we scale up the training data. The results suggest the advantage of training datasets with high scalability.



### Experiments: Evaluation with RepoST-Eval

We benchmark 12 Code LLMs on RepoST-Eval to evaluate their abilities to generate code in real GitHub repositories. Results show that:

* Our benchmark is able to distinguish different models. The best model (GPT-4o) only achieves 39.53 Pass@1, which shows that our benchmark is challenging and has a large room for improvement.
* Models achieve much higher performance when provided with generated docstrings. One possible explanation is that the quality of human-written docstrings varies widely (on average 36.82/3.95 tokens/lines). Some docstrings may only contain limited information, while the generated docstrings generally contain more details (on average 130.84/9.92 tokens/lines).



### Code and Data

Both the code of RepoST and the RepoST-Train/Eval datasets (including the docker environment) are available on GitHub.