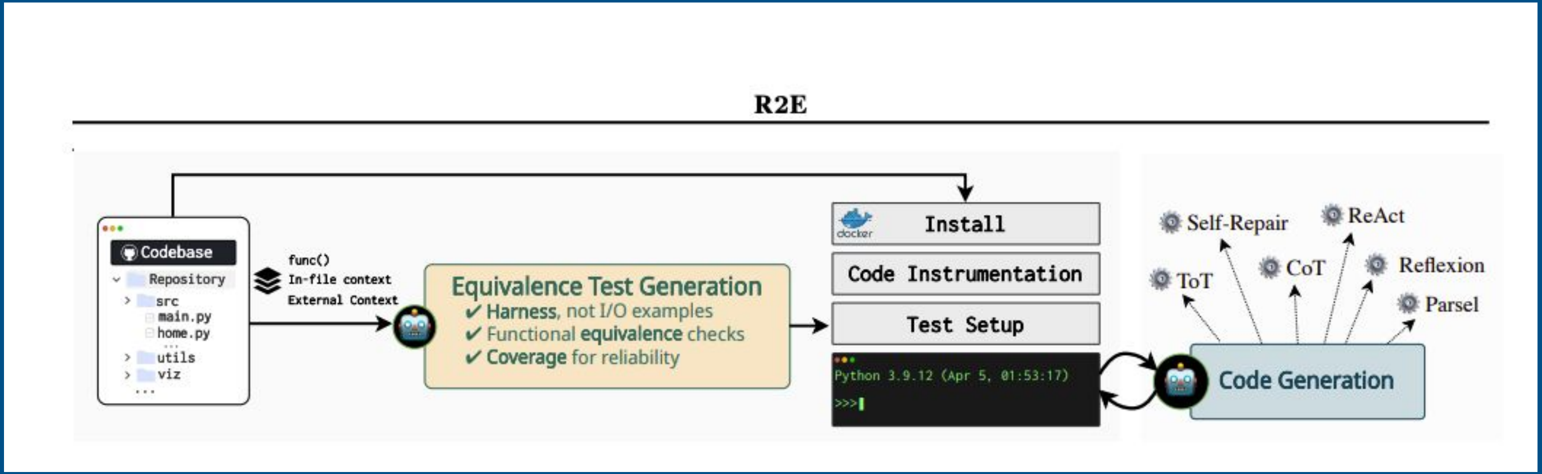
Repository To Environment

Presenter:
Jinjian Liu



Road Map

1. **R2E Overview**
2. **Scaling**
3. **Multilingual**
4. **Pyperf**
5. **Future Directions**

R2E Overview

Foundation: "R2E: Turning Any GitHub Repository into a Programming Agent Environment." *ICML 2024 Poster*, May 2024, https://r2e.dev/. https://sky.cs.berkeley.edu
Motivation: Build a scalable and reliable testbed for AI-assisted code generation. R2E is created as a framework to evaluate code-generating systems and produce benchmarks.

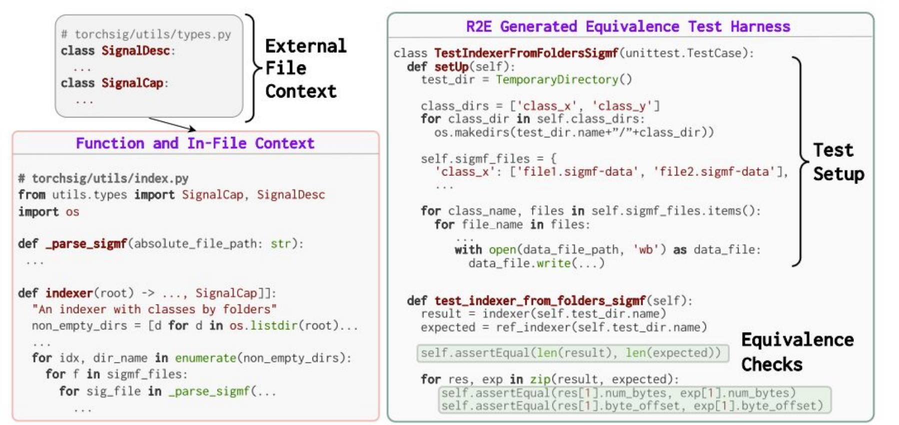
Steps:
1. Problem curation: Based on collected qualifying repositories (40 + stars, non-fork, etc.), curate qualifying functions for standard correctness evaluation testing (has docstring, connected with other repo components, no GPU requirements, etc.) -> 9825 problems from 429 repositories.

2. Test Harness Generation: Generate high-quality equivalence test harnesses using LLM like GPT-4 with “dependency slicing based prompt” to ensure minimal yet sufficient context (not entire repo). Focus on equivalence tests (comparing generated outputs with reference function outputs) instead of I/O pairs.

3. Test Harness Evaluation: Quality and Validity

4. Finally: The researchers instantiate this framework to construct R2E-Eval1, the first large-scale dataset of real-world code generation problems with functional correctness tests.

An example problem and test harness



```
# torchsig/utils/types.py
class SignalDesc:
  ...
class SignalCap:
  ...
```

```
# torchsig/utils/index.py
from utils.types import SignalCap, SignalDesc
import os

def _parse_sigmf(absolute_file_path: str):
 ...

def indexer(root) -> ..., SignalCap]]:
  "An indexer with classes by folders"
  non_empty_dirs = [d for d in os.listdir(root)...
  ...
  for idx, dir_name in enumerate(non_empty_dirs):
    for f in sigmf_files:
      for sig_file in _parse_sigmf(...
        ...
```

```
class TestIndexerFromFoldersSigmf(unittest.TestCase):
  def setUp(self):
    test_dir = TemporaryDirectory()

    class_dirs = ['class_x', 'class_y']
    for class_dir in self.class_dirs:
      os.makedirs(test_dir.name+”/”+class_dir))

    self.sigmf_files = {
      'class_x': ['file1.sigmf-data', 'file2.sigmf-data'],
      ...

    for class_name, files in self.sigmf_files.items():
      for file_name in files:
        ...
        with open(data_file_path, 'wb') as data_file:
          data_file.write(...)

  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)
```

R2E Scale

Motivation: We want to use LLM to not only generate code, but also to parse and understand the building environments of repositories.

Summary of what we have already:

R2E provides an initial framework supporting test harness generation. When given a random GitHub repo url, some amount of manual work needs to be done to setup the repository, and then when the user randomly picks a function to test: 1. it will firstly figure out the dependent functions that are used in the target function, generate a ***prompt*** for the LLM, ask it to generate the target function without seeing the ground truth. 2. It will secondly ask an LLM to write a test function to test the equivalence of the generated function and the original function, execute both and make sure they are equivalent.

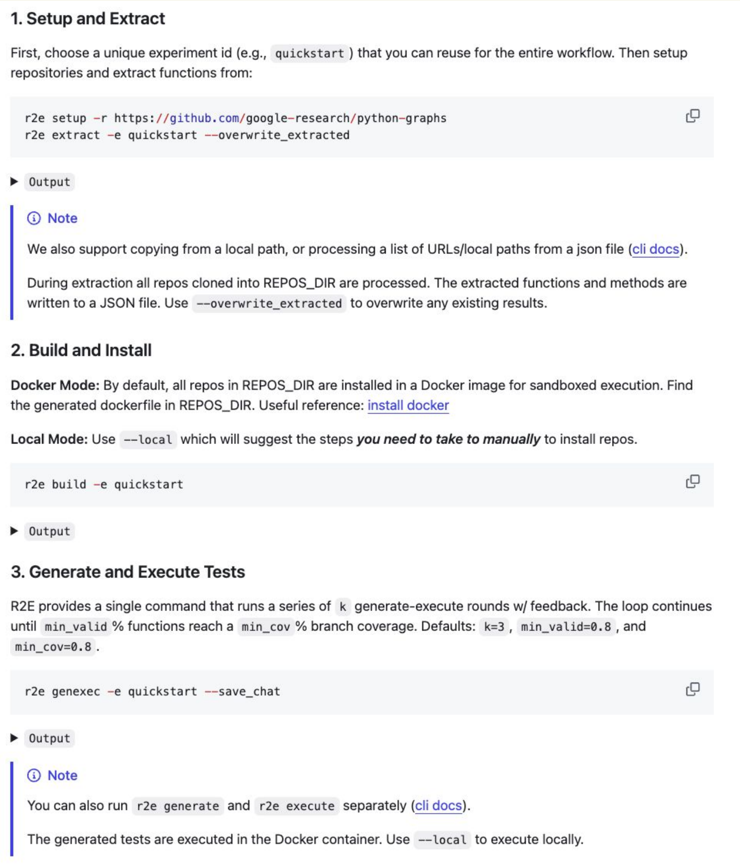
In this way, when an LLM connected to a GitHub repository, it can first setup (through docker), then try to write functions, with the test cases provided. Finally, it will debug the code until it is correct.

Goal:

Extending from our motivation, the purpose of the R2E Scale project is to build a universal coding simulator (similar to VisualStudio Code) that is compatible and scalable across different codebases. On the end goal, when people want to connect a codebase to an LLM through our simulator, we want to enable software workflows like code generation, unit test generation, debugging, execution, PR raising/fixing, etc. So that people can safely test their LLM agents in our simulator. The desired workflow is given a repo and some specifications, we ideally can use an LLM to setup the repo (if the hardware requirements are met) and install all the dependencies, then define an interactive execution and debug. Simply put: given a repo, we want to make the repo executable by writing modularized functions and testing harnesses. Still very focused on building the reliable environment.

R2E Scale

Problem: Auto repository setup: A current limitation of the above approach right now is that the repository setup stage is not automated. This usually means some amount of manual work still needs to be done when a new repository is connected.



1. Setup and Extract

First, choose a unique experiment id (e.g., `quickstart`) that you can reuse for the entire workflow. Then setup repositories and extract functions from:

```
r2e setup -r https://github.com/google-research/python-graphs
r2e extract -e quickstart --overwrite_extracted
```

▶ `Output`

ⓘ Note

We also support copying from a local path, or processing a list of URLs/local paths from a json file (cli docs).

During extraction all repos cloned into REPOS_DIR are processed. The extracted functions and methods are written to a JSON file. Use `--overwrite_extracted` to overwrite any existing results.

2. Build and Install

Docker Mode: By default, all repos in REPOS_DIR are installed in a Docker image for sandboxed execution. Find the generated dockerfile in REPOS_DIR. Useful reference: install docker

Local Mode: Use `--local` which will suggest the steps ***you need to take to manually*** to install repos.

```
r2e build -e quickstart
```

▶ `Output`

3. Generate and Execute Tests

R2E provides a single command that runs a series of `k` generate-execute rounds w/ feedback. The loop continues until `min_valid` % functions reach a `min_cov` % branch coverage. Defaults: `k=3`, `min_valid=0.8`, and `min_cov=0.8`.

```
r2e genexec -e quickstart --save_chat
```

▶ `Output`

ⓘ Note

You can also run `r2e generate` and `r2e execute` separately (cli docs).

The generated tests are executed in the Docker container. Use `--local` to execute locally.

Task: Automation

We built an installer script that will automatically carry out the installation on docker images: Note we combined steps from cloning repo up to test generation all into this one script. An installation oracle function was created to incorporate LLM support into this process.

```python
# For each segment run this
outputs = run_tasks_in_parallel(
    install_repo,
    segment_urls,
    num_workers=installer_num_workers,
    timeout_per_task=1800,
    use_progress_bar=True,
    progress_bar_desc=f"Installing repos {start + 1} to {end}..."
)
```

Key features:
1. Docker images provide isolation of different environments.
2. Multiprocess parallel execution to reduce overhead.
3. Agentic support for installation and debugging.
4. Repo quality control with number of tests

Other nuances:
1. Logger documentation for success, failure ratio and detailed output
2. Disk space management with periodic pruning

R2E Scale

Workflow:

1. Repository Curation: We used SEART to search for repos with qualifying attributes, and consolidated 1300 repositories for experimentation. – we tested about 1000 repos, but not all of them are good/worth looking at

```python
def download_github_repos(start_date, end_date, output_file_name, min_stars, max_stars, max_urls):
    # Base URL for SEART API
    BASE_URL = "https://seart-ghs.si.usi.ch/api/r/download/csv"

    # Validate date inputs
    try:
        datetime.strptime(start_date, "%Y-%m-%d")
        if end_date:
            datetime.strptime(end_date, "%Y-%m-%d")
    except ValueError:
        raise ValueError("Incorrect date format, should be YYYY-MM-DD")

    # Create seart folder if it doesn't exist
    seart_folder = "seart"
    os.makedirs(seart_folder, exist_ok=True)
```

2. If error occurs during the installation, we optionally pass the error output to an LLM and run the suggested fixing command with the contraction of an oracle. We limit the number of agentic loops to prevent stuck in a cycle. Sometimes LLMs would cascade on a small mistake.

```python
msg_content = f"""
Context: {context}
Last command executed: {last_command}
Output/Error: {last_output}
Oracle result: {oracle_result}

- Suggest the next command to run in the Docker container to complete the installation process.
- The repo in question is already partially installed in the Docker container at /repos/(name_of_repo). You may assume that you are CDed into this directory automatically.
- The repo has a partially installed virtual environment at `.venv`; you may assume that the virtual environment is already activated.
- The installation is complete if and only if the Oracle returns "INSTALLATION SUCCESSFUL".
- Important Note: Every shell command that you run is executed in a separate bash session in the Docker container. If you create any aliases or environment variables, make sure to s
- Your response should be a shell command for the Docker container or 'RUN ORACLE'. When you write 'RUN ORACLE', the Oracle will be consulted to determine if the installation is com
- Do not attempt to run the Oracle directly, as it is located somewhere that you cannot access. The Oracle will be automatically consulted for you if you say, 'RUN ORACLE'.
"""
#TODO: Enhance prompt engineering
response = openai_client.chat.completions.create(
    model="gpt-4-turbo",
    messages=[
        {"role": "system", "content": "You are an AI assistant helping to complete the installation process of a partially-installed repo within a Docker container. Read the followi
        {"role": "user", "content": msg_content}
    ]
)
return response.choices[0].message.content.strip().replace("```bash", "").replace("`", "").replace("\n", "")

human_intervention(context, last_command, last_output, oracle_result):
```

R2E Scale

Workflow – Continued:

3. Testing and analysis:

Example of a failed test

```
    "function_code": "def check_output(*popenargs, **kwargs):\n    \"\"\"Run command with arguments and return its output as a byte string.\n\n    If the exit code was non-zero i
    "function_name": "check_output",
    "function_complexity": null,
    "context": null,
    "test_history": {
        "history": [
            {
                "tests": {},
                "operation": "generate",
                "gen_model": "gpt-4-turbo-2024-04-09",
                "gen_date": "20240820_095007",
                "exec_stats": {
                    "output": null,
                    "error": "Error: Traceback (most recent call last):\n  File \"/repos/05sonicblue___gamezserver/.venv/lib/python3.11/site-packages/r2e_test_server/server.py\",
                }
            }
        ]
    }
},
{
    "function_id": {
```

“Exec_stats” field is used to determine the success of a test. Key-word based scanning - “Error”, if “history” field is not empty

The overall ratio of (number of tests passed)/(number of tests total), with a cutoff of 0.95 to be considered as a successful installation.

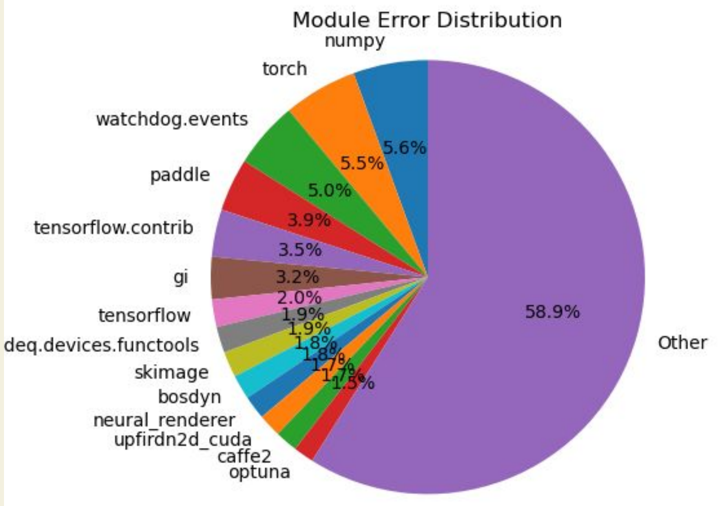
R2E Scale

Data analysis: Top failure modes

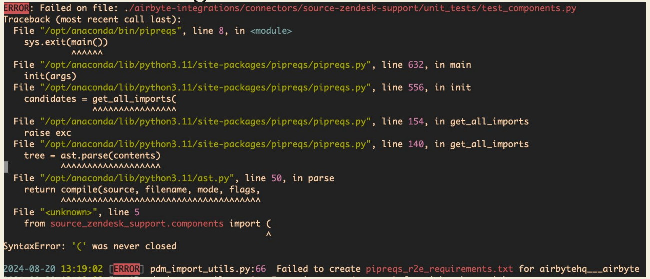
- ModuleNotFoundError: 182 occurrences
- ImportError: 164 occurrences
- AttributeError: 70 occurrences
- NameError: 28 occurrences
- SyntaxError: 27 occurrences
- FileNotFoundError: 20 occurrences
- RuntimeError: 10 occurrences
- ValueError: 8 occurrences
- KeyError: 7 occurrences
- ZeroDivisionError: 2 occurrences

(348 total repos)

Module not found error distribution



Example of modules not found error:
airbytehq/airbyte is a repo for which even basic dependencies like requests, yaml and pytest are not detected
There's a syntax error in one of the files. This causes pipreqs to terminate without installing



```
ERROR: Failed on file: ./airbyte-integrations/connectors/source-zendesk-support/unit_tests/test_components.py
Traceback (most recent call last):
  File "/opt/anaconda/bin/pipreqs", line 8, in <module>
    sys.exit(main())
             ^^^^^^
  File "/opt/anaconda/lib/python3.11/site-packages/pipreqs/pipreqs.py", line 632, in main
    init(args)
  File "/opt/anaconda/lib/python3.11/site-packages/pipreqs/pipreqs.py", line 556, in init
    candidates = get_all_imports(
                 ^^^^^^^^^^^^^^^^
  File "/opt/anaconda/lib/python3.11/site-packages/pipreqs/pipreqs.py", line 154, in get_all_imports
    raise exc
  File "/opt/anaconda/lib/python3.11/site-packages/pipreqs/pipreqs.py", line 140, in get_all_imports
    tree = ast.parse(contents)
           ^^^^^^^^^^^^^^^^^^^
  File "/opt/anaconda/lib/python3.11/ast.py", line 50, in parse
    return compile(source, filename, mode, flags,
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "<unknown>", line 5
    from source_zendesk_support.components import (
                                                  ^
SyntaxError: '(' was never closed

2024-08-20 13:19:02 [ERROR] pdm_import_utils.py:66  Failed to create pipreqs_r2e_requirements.txt for airbytehq___airbyte
```

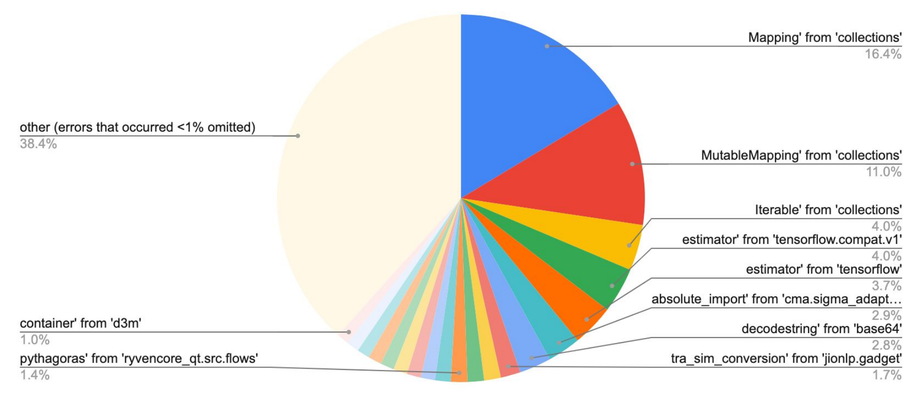
R2E Scale

Data analysis: Top failure modes

- ModuleNotFoundError: 182 occurrences
- ImportError: 164 occurrences
- AttributeError: 70 occurrences
- NameError: 28 occurrences
- SyntaxError: 27 occurrences
- FileNotFoundError: 20 occurrences
- RuntimeError: 10 occurrences
- ValueError: 8 occurrences
- KeyError: 7 occurrences
- ZeroDivisionError: 2 occurrences

(348 total repos)

Import Errors Breakdown:



In a separate experiment: How the statistics change when import errors are excluded: Success rate went from 7% to 14%

When all directories are added to sys.path, import conflicts would be error inducing.

R2E Scale

Data analysis: Top failure modes

- ModuleNotFoundError: 182 occurrences
- ImportError: 164 occurrences
- AttributeError: 70 occurrences
- NameError: 28 occurrences
- SyntaxError: 27 occurrences
- FileNotFoundError: 20 occurrences
- RuntimeError: 10 occurrences
- ValueError: 8 occurrences
- KeyError: 7 occurrences
- ZeroDivisionError: 2 occurrences

(348 total repos)

Out of 31 successes of 490 tests in one experiment: Most successful installations (23/31) achieved 1.0 pass rate, the distribution skews towards 1.0

Success Ratio: 0.96, ID: ['facebookresearch']

Success Ratio: 0.98, ID: ['eric-mingjie']

Success Ratio: 0.99, ID: ['subbarayudu-j', 'maqp', 'nexb', 'uclnlp', 'opendevops-cn', 'markqvist']

Success Ratio: 1.00, ID: ['jiesutd', 'maratyszcza', 'musicmancorley', 'fabiocaccamo', 'csawtelle', 'dreamoftheredchamber', 'wuduhren', 'blockchain-etl', 'hips', 'dynobo', 'pyqt', 'moderngl', 'juanpotato', 'seikur0', 'vinta', 'chris7', 'merkremont', 'plasma-disassembler', 'gameboy12615', 'pyside', 'amimo', 'wuziheng', 'coldmanck']

As we can see successful installations show pretty promising results

R2E Scale

Data analysis: Top failure modes

- ModuleNotFoundError: 182 occurrences
- ImportError: 164 occurrences
- AttributeError: 70 occurrences
- NameError: 28 occurrences
- SyntaxError: 27 occurrences
- FileNotFoundError: 20 occurrences
- RuntimeError: 10 occurrences
- ValueError: 8 occurrences
- KeyError: 7 occurrences
- ZeroDivisionError: 2 occurrences

(348 total repos)

Why pipreqs isn't installing missing dependencies:

Syntax Errors

- Syntax errors can cause pipreqs to terminate early (see prev. slide)

Version Conflicts

- Docker containers use Python 3.11.5 for parallel installation. Which sometimes causes incompatibility issues.
  - Example: removal of ABC classes from collections breaks import statements like `from collections ….` (should be `from collections.abc`)

R2E Scale

Problem Summary:

Repo quality issues:
The most irksome property of the repos is that they are very miscellaneous with varying quality. Some examples of problematic features:

1. Blank tests/low number of original tests can produce false negatives, simply discard

2. Python 2 syntax in older repos. Also other dependency errors that are hard to debug. Discard

Challenge of Multiprocessing:
When number of workers is too big (>20) Unix HTTP Connection Pool error occurs at random. We fixed it by simply limiting the number of workers to relatively smaller (~10).
Logging tend to be a little messy during the parallel processing of multiple repositories.

During installation, the success rate is still low, and the failure modes are heavily skewed towards missing modules, import errors, and attribute not found errors.
Ideas on fixing:
Fork and locally enhance pipreqs to avoid early termination
Resolve import conflicts on sys.path

The terminal goal here is to yet bump up success ratio during automation.

R2E Multilingual

Motivation:
The above mentioned workflow now only implemented in with python language. The key challenge is to implemented a dependent function tracker, basically a call graph analyzer for different languages.

Right now we are exploring in C. Still in the early phase of development.

Some starting steps:
1. Collect install commands to cover different type of build systems (CMake, build.sh, or custom build systems)

2. Analyze failure modes and experiment with reliable fixes (e.g. adding agent support Oracle as well?)

R2E Pyperf

This repository is still private, so I'll only include text description. Might be a little dry.

Motivation:
Still based on LLM code optimization, we want to benchmark how well can LLM optimize a repository. And we are currently developing that benchmark.

Workflow:
1. Problem curation: We generate commit history mapping of repositories, and find commit histories related to performance enhancement.
Then we want to test how the amount of optimization achieved by the performance enhancing changes in contrast to before.
Changes that achieve good optimization (e.g. > 2 times speedup)

2. Test Harness Creation: Similar to the R2E Scale Project, we prompted LLM to create a test harness for interesting functions to analyze performance enhancement. Dependency slicing is also employed here to reduce overhead and provide sufficient context. A very interesting feature is called quickcheck - we run the test harness locally to troubleshoot before execution. Should any bugs occur, an agentic loop is utilized to improve the tests. If the tests pass locally, we then move on to execution on a VM.
3. Finally the prob
optimization analy
Some improveme
1. Prompt enginee

Repo	Target API	Commit A^	Commit B	Time A (s)	Time B (s)	Speedup	Opt %
pylint	run_pylint	v:2.14.5	v:latest	123.760	106.930	1.157	16.21 (0.70)
pylint	run_pylint	v:2.15.7	v:latest	116.150	106.930	1.086	8.75 (0.60)
pillow	im.save	a51d3bcd: Speed	main	3.066	0.385	7.964	87.45 (0.33)

2. Better automation

Acknowledgement

Manish Shetty
Tianjun Zhang
Naman Jain
Vijay Kethenaboyina
Koushik Sen