SWE-Opt: A Benchmark for Evaluating LLMs on Performance-Oriented Repository-Level Optimization

Ezgi Sarıkayak[1 2] Wenchao Gu[2] Hesham Ghonim[2] Chunyang Chen[2]

[1] Siemens Research, Munich [2] Technical University of Munich

NeurIPS 2026 · Evaluations & Datasets Track

SIEMENS

Paper Code Data

Motivation - Can LLMs Optimize Code Like Experts?

Expert optimizations across 102 performance-optimizing pull requests

15×
runtime speedup

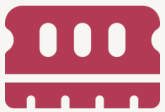
171×
peak-memory reduction

619×
TWMU reduction
time-weighted memory usage

Can LLMs match these expert improvements?

Motivation · Challenges of Performance Optimization

Challenge: Performance optimization is



Multi-metric

Runtime and memory

Code can get faster but use more memory.



Workload-dependent

Gains vary across workloads

A change can help on large inputs and do nothing on small ones.



Noise-sensitive

Noise can hide or fake gains

Timings change from run to run.

SWE-Opt · What We Do Differently

Multi-metric

Faster, but at what memory cost?

Runtime
Execution time

Peak memory
Maximum memory usage

TWMU
Time-weighted memory usage

Workload sweeps

Fast on which inputs?

✓ **Coverage:** Exercise the optimized path

✓ **Sensitivity:** Expose performance differences

✓ **Workload Grid:** ≤ 24 configurations

✓ **Feasibility:** Within time and memory limits

Noise-aware

Real gain, or just noise?

Calibrate
Set repetitions and sampling rate

Warm up
Remove cold-start effects

Adapt
Repeat until 90% CI ≤ 10%

Validate
Keep effects ≥ 2× the noise

Beyond correctness and speed: SWE-Opt adds memory, workloads and noise-aware measurement.

SWE-Opt · Dataset and Performance Test

Dataset

102
tasks from performance optimizing PRs

3
repositories

pandas · scikit-learn · xarray

Each task has

Given to the LLM

- Task description
- Full repository

Used for evaluation

- **Performance test**
- Unit tests
- Gold patch (expert)

Parameterized performance test

Example · pandas PR #50778: Speed up Series.replace

PR50778Scenario · shortened

```
class PR50778Scenario(PerfScenario):

  def setup(self):
    arr = self.rng.standard_normal(N)
    ser = pd.Series(arr)
    to_replace = self.rng.choice(arr, R)
    # ... build replace_dict

  def run(self):
    return ser.replace(replace_dict)
```

One test → 6 workloads

	R 1K	R 5K
N 100K	1	2
N 500K	3	4
N 1M	5	6

N = series length
R = values replaced

Metrics · How Often and How Much?

Performance metrics
measured on every workload

Runtime
execution time

Peak memory
maximum memory used

TWMU
Time-weighted memory usage

For each metric, two numbers:



Pass rate

How often?

% of the 102 tasks

with a stable gain ≥ 2× the noise



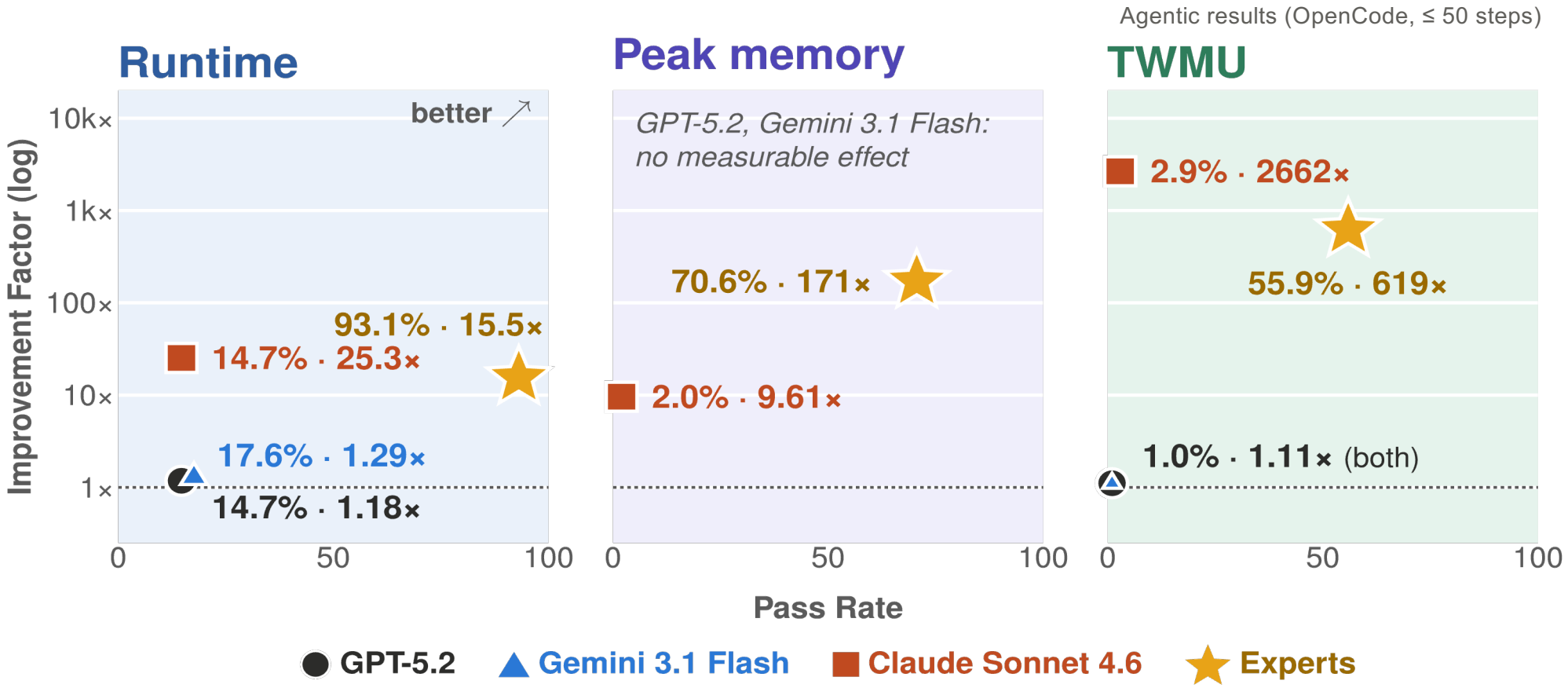
Improvement factor (IF)

How much?

baseline ÷ patched

Both matter: a large factor on very few tasks is not broad progress.

Finding 1 · A Large Gap to Experts



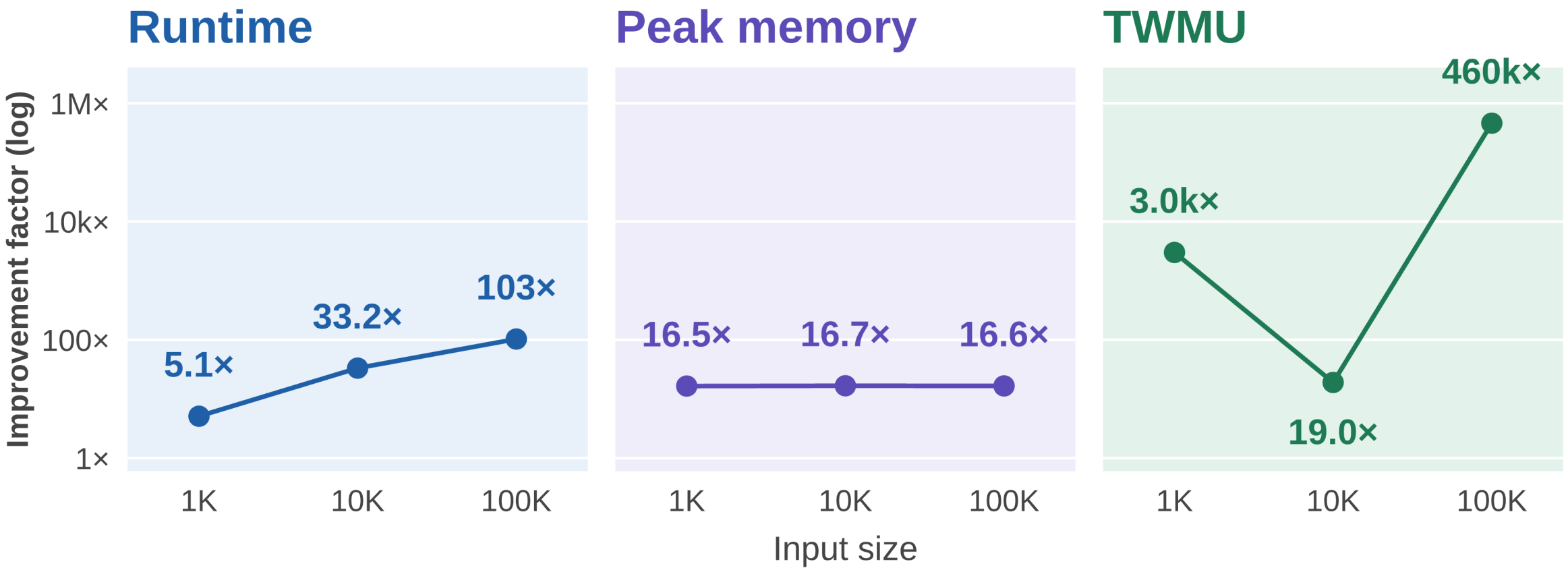
Runtime: large, but on few tasks
Claude S. 4.6 25.3× on 15 tasks
Experts 15.5× on 95 tasks

Peak memory: 18× smaller gains
Claude S. 4.6 9.6× on 2 tasks
Experts 171× on 72 tasks

TWMU: isolated wins only
Claude S. 4.6 2,662× on 3 tasks
Experts 619× on 57 tasks

Finding 2 · Workloads Matter

One Claude Sonnet 4.6 patch (xarray PR #7578, stride 10), measured at three input sizes.



Same patch, different input sizes, different gains.
One workload is not enough to judge a patch. (Claude Sonnet 4.6 Patch on xarray PR #7578)

Takeaways

Speed: Far from experts

The best agent speeds up 17.6% of tasks; expert patches speed up 93.1%.

Memory: A blind spot

Two of three models never reduce peak memory, the best agent reaches 2.9% of tasks (TWMU).

Workloads matter

One patch ranges from 5× to 103× runtime gain across input sizes
Measure a grid, not one input.

Paper Code Data

SIEMENS

Contact: ezgi.sarikayak@siemens.com

Supported by BMFTR, ITEA4 project GreenCode (16IS24070G)