

Lab 073: Agent Benchmarking with SWE-bench

Level: L300 Path: All paths Time: ~90 min ? Cost: Free ? Uses mock benchmark results
info "Practice type" This lab analyzes mock SWE-bench-style results. It teaches benchmarking methodology and trade-off analysis without running expensive coding-agent evaluations.

What You'll Learn

What SWE-bench is and why it's the gold standard for evaluating coding agents How different agent strategies (direct prompt, chain-of-thought, agentic loop) affect resolve rates
Analyze benchmark results across models and strategies to find the best agent configuration
Measure the cost-performance trade-off ? higher resolve rates cost more per issue Build a benchmark comparison report for selecting the right agent architecture

Introduction

SWE-bench is a benchmark for evaluating AI coding agents on real-world GitHub issues. Each task is a genuine bug or feature request from popular open-source Python repositories (Django, scikit-learn, sympy, etc.). The agent must:

1. Read the issue description 2. Navigate the codebase 3. Write a patch that fixes the issue 4. Pass the repository's test suite

Benchmark Variant | Issues | Difficulty | Use Case | SWE-bench Full | 2,294 | Mixed | Comprehensive evaluation | SWE-bench Lite | 300 | Curated subset | Quick comparison (used in this lab) | SWE-bench Verified | 500 | Human-verified | Gold-standard evaluation |

The Scenario

You are an AI Platform Architect evaluating coding agents for your engineering team. You've benchmarked 8 agent configurations across 3 models (GPT-4o, o3, Claude 3.5 Sonnet) and 4 strategies (direct prompt, chain-of-thought, agentic loop, mini SWE-agent). Your dataset (swe_bench_results.csv) contains the results. Your job: identify the best agent configuration and understand the cost-performance trade-offs.

info "Mock Data" This lab uses mock benchmark results that mirror published SWE-bench leaderboard trends. Real benchmarking requires significant compute ? this mock dataset lets you learn the analysis methodology without the cost.

Prerequisites

Requirement | Why | Python 3.10+ | Run the analysis scripts | pandas library | Data manipulation |

[diagram or code sample]

tip "Quick Start with GitHub Codespaces" Open in GitHub Codespaces

All dependencies are pre-installed in the devcontainer.

? Supporting Files

note "Download these files before starting the lab" Save all files to a lab-073/ folder in your working directory.

File | Description | Download | broken_benchmark.py | Bug-fix exercise (3 bugs + self-tests) | ? Download | swe_bench_results.csv | Dataset | ? Download |

Step 1: Understand Agent Strategies

Before analyzing results, understand the four strategies being benchmarked:

Strategy | How It Works | Expected Performance | Expected Cost | Direct Prompt | Single prompt with issue + codebase context ? patch | Lowest | Lowest | Chain of Thought | Prompt with explicit reasoning steps ? patch | Medium | Medium | Agentic Loop | Multi-turn loop: read code ? reason ? edit ? test ? iterate | Highest | Highest | Mini SWE-agent | Lightweight agent with file navigation and edit tools | Medium-High | Medium |

Key Metrics

Metric | Definition | Resolve Rate | % of issues where the agent's patch passes all tests |
Avg Time | Average seconds per issue attempt | Avg Cost | Average USD per issue attempt |

Step 2: Load and Explore the Results

The dataset has 8 agent configurations tested on SWE-bench Lite (300 issues) and Verified (500 issues):

[diagram or code sample]

Expected output:

[diagram or code sample]

Step 3: Find the Best and Worst Agents

Rank agents by resolve rate to find the top performers:

[diagram or code sample]

Expected output:

Rank	Agent	Model	Strategy	Resolve Rate	Cost/Issue
1	Agentic o3	o3	agentic_loop	65.0%	\$5.50
2	Agentic Claude	claude-3.5-sonnet	agentic_loop	56.0%	\$3.20
3	Agentic GPT-4o	gpt-4o	agentic_loop	50.0%	\$2.50
4	Baseline o3	o3	direct_prompt	45.0%	\$3.00
5	CoT GPT-4o	gpt-4o	chain_of_thought	40.0%	\$1.20
6	Mini SWE-agent	gpt-4o	mini_swe_agent	35.0%	\$1.80
7	Baseline Claude	claude-3.5-sonnet	direct_prompt	35.0%	\$0.95
8	Baseline GPT-4o	gpt-4o	direct_prompt	30.0%	\$0.85

[diagram or code sample]

tip "Insight" Agentic o3 leads at 65% ? but at \$5.50 per issue. Baseline GPT-4o is cheapest at \$0.85 but resolves only 30%. The agentic loop strategy consistently outperforms other strategies for the same model.

Step 4: Measure the Agentic Improvement

How much does the agentic loop strategy improve over the baseline (direct prompt) for the same model?

[diagram or code sample]

Expected output:

[diagram or code sample]

tip "Insight" The agentic loop adds +20?21 percentage points across all models. This consistent improvement suggests that the strategy (iterative code navigation + testing) matters as much as the underlying model.

Step 5: Analyze Cost-Performance Trade-offs

More capable agents cost more. Calculate the cost per resolved issue to find the most efficient option:

[diagram or code sample]

[diagram or code sample]

warning "The Cost Curve" Going from 30% to 65% resolve rate (2.2x improvement) costs \$5.50 vs. \$0.85 per attempt (6.5x more expensive). Diminishing returns kick in hard ? evaluate whether the marginal resolve rate improvement justifies the cost for your use case.

Step 6: Build the Benchmark Report

[diagram or code sample]

? Bug-Fix Exercise

The file lab-073/broken_benchmark.py contains 3 bugs that produce incorrect benchmark analysis. Can you find and fix them all?

Run the self-tests to see which ones fail:

[diagram or code sample]

You should see 3 failed tests. Each test corresponds to one bug:

Test | What it checks | Hint | Test 1 | Best agent selection | Should find the agent with the *highest* resolve rate, not lowest | Test 2 | Average cost per resolved issue | Should divide by *resolved* count, not total issues | Test 3 | Agentic vs. baseline comparison | Should filter by *model* before comparing strategies |

Fix all 3 bugs, then re-run. When you see All passed!, you're done!

? Knowledge Check

question "Q1 (Multiple Choice): What does SWE-bench measure about a coding agent?"

- A) How fast it can generate code - B) Whether it can resolve real GitHub issues by producing patches that pass tests - C) How many lines of code it can write per minute - D) Whether it can explain code to a human

??? success "? Reveal Answer" Correct: B) Whether it can resolve real GitHub issues by producing patches that pass tests

SWE-bench evaluates agents on their ability to fix genuine bugs and implement features from real open-source repositories. The agent must produce a patch, and the patch must pass the project's existing test suite to count as "resolved."

question "Q2 (Multiple Choice): Why does the agentic loop strategy outperform direct prompting?"

- A) It uses a larger context window - B) It iterates: reading code, reasoning, editing, and testing in a loop - C) It uses a more expensive model - D) It has access to the internet
??? success "? Reveal Answer" Correct: B) It iterates: reading code, reasoning, editing, and testing in a loop

The agentic loop gives the agent multiple turns to explore the codebase, form hypotheses, write patches, run tests, and revise. This mirrors how human developers work ? rarely does a one-shot attempt solve a complex bug.

question "Q3 (Run the Lab): Which agent configuration has the highest resolve rate?"

Run the Step 3 analysis on ? swe_bench_results.csv and check the ranking.

??? success "? Reveal Answer" Agentic o3 ? 65%

Agent AG05 ("Agentic o3") using the o3 model with the agentic_loop strategy resolves 195 out of 300 issues (65.0%), the highest of all 8 configurations.

question "Q4 (Run the Lab): Which agent configuration has the lowest resolve rate?"

Check the bottom of the Step 3 ranking.

??? success "? Reveal Answer" Baseline GPT-4o ? 30%

Agent AG01 ("Baseline GPT-4o") using the gpt-4o model with the direct_prompt strategy resolves only 90 out of 300 issues (30.0%). Direct prompting with no iteration yields the lowest performance.

question "Q5 (Run the Lab): How many percentage points does the agentic loop improve over the baseline for the same model?"

Run the Step 4 analysis to compute the agentic improvement per model.

??? success "? Reveal Answer" +20pp for GPT-4o (30%?50%), +20pp for o3 (45%?65%), +21pp for Claude (35%?56%)

The agentic loop consistently adds 20?21 percentage points of resolve rate over the direct-prompt baseline, regardless of the underlying model. This demonstrates that agent architecture matters as much as model capability.

Summary

Topic | What You Learned | SWE-bench | Gold-standard benchmark using real GitHub issues and test suites | Resolve Rate | Primary metric ? % of issues where the agent's patch passes tests | Agentic Loop | +20pp improvement over direct prompting for any model | Cost Trade-off | 65% resolve rate costs 6.5x more per attempt than 30% | Model vs. Strategy |

Strategy (agentic loop) contributes as much as model choice | Benchmark Analysis | How to rank, compare, and report on agent configurations |

Next Steps

Lab 035 ? Agent Evaluation with Azure AI Eval SDK (custom evaluation beyond SWE-bench) Lab

038 ? AI Cost Optimization (managing the cost of agentic loops) Lab 040 ? AutoGen

Multi-Agent (building agentic loops with AutoGen) Lab 074 ? Foundry Agent Service (operating and monitoring agent fleets)

Phase 3 · Simulation / Real path

Simulation Real path

Preserve the existing free simulation and its fixtures. Estimated cost: US\$0.

Reuse the same acceptance criteria against the documented hosted service. Before running, record a service-specific estimate and a hard spending cap. After validation: Delete hosted deployments, endpoints, indexes, traces, and test data after validation.

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

[Download generated PDF](#) [Download generated ZIP](#)

Supporting file: broken_benchmark.py

```
#!/usr/bin/env python3 """Bug-fix: Broken Benchmark Analyzer - Lab 073. Fix 3 bugs.""" import io, pandas as pd def best_agent(results): # Bug #1: Returns agent with lowest resolve rate instead of highest return results.loc[results["resolve_rate_pct"].idxmin(), "agent_name"] def avg_cost_per_resolve(results): # Bug #2: Divides total cost by total issues instead of resolved return results["avg_cost_usd"].sum() / results["total_issues"].sum() def agentic_vs_baseline(results, model): # Bug #3: Doesn't filter by model agentic=results[results["strategy"]=="agentic_loop"]["resolve_rate_pct"].mean() baseline=results[results["strategy"]=="direct_prompt"]["resolve_rate_pct"].mean() return agentic - baseline def run_tests(): csv="agent_id,agent_name,model,total_issues,resolved,resolve_rate_pct,avg_cost_usd,strategy\nA1,Base GPT,gpt-4o,100,30,30.0,0.85,direct_prompt\nA2,Agent GPT,gpt-4o,100,50,50.0,2.50,agentic_loop\nA3,Base o3,o3,100,45,45.0,3.00,direct_prompt\nA4,Agent o3,o3,100,65,65.0,5.50,agentic_loop" r=pd.read_csv(io.StringIO(csv));p,f=0,0 b=best_agent(r);exp="Agent o3" if b==exp:print(f"OK T1: {b}");p+=1 else:print(f"FAIL T1: {b} (exp {exp})");f+=1 c=avg_cost_per_resolve(r);exp=sum([0.85,2.50,3.00,5.50])/sum([30,50,45,65]) if abs(c-exp)<0.001:print(f"OK T2: {c:.4f}");p+=1 else:print(f"FAIL T2: {c:.4f} (exp {exp:.4f})");f+=1 d=agentic_vs_baseline(r,"gpt-4o");exp=20.0 if abs(d-exp)<0.1:print(f"OK T3: {d}");p+=1 else:print(f"FAIL T3: {d} (exp {exp})");f+=1 print(f"\n{'All passed!' if f==0 else f'{f} failed'}") if __name__=="__main__": print("Tests:\n");run_tests()
```

Supporting file: swe_bench_results.csv

```
agent_id,agent_name,model,benchmark,total_issues,resolved,resolve_rate_pct,avg_time_sec,avg_cost_usd,strategy AG01,Baseline GPT-4o,gpt-4o,swe-bench-lite,300,90,30.0,45,0.85,direct_prompt AG02,CoT GPT-4o,gpt-4o,swe-bench-lite,300,120,40.0,68,1.20,chain_of_thought AG03,Agentic GPT-4o,gpt-4o,swe-bench-lite,300,150,50.0,120,2.50,agentic_loop AG04,Baseline o3,o3,swe-bench-lite,300,135,45.0,90,3.00,direct_prompt AG05,Agentic o3,o3,swe-bench-lite,300,195,65.0,180,5.50,agentic_loop AG06,Baseline
```

Claude,claude-3.5-sonnet,swe-bench-lite,300,105,35.0,55,0.95,direct_prompt AG07,Agentic
Claude,claude-3.5-sonnet,swe-bench-lite,300,168,56.0,145,3.20,agentic_loop AG08,Mini
SWE-agent,gpt-4o,swe-bench-verified,500,175,35.0,95,1.80,mini_swe_agent