

Lab 073: Benchmarking de Agentes com SWE-bench

Nível: L300 Trilha: Todas as trilhas Tempo: ~90 min ? Custo: Gratuito ? Usa resultados simulados de benchmark

info "Tipo de prática" Este lab analisa resultados simulados no estilo SWE-bench. Ele ensina metodologia de benchmark e análise de trade-offs sem executar avaliações caras de agentes de codificação.

O que Você Vai Aprender

O que é o SWE-bench e por que ele é o padrão ouro para avaliar agentes de codificação Como diferentes estratégias de agentes (prompt direto, cadeia de pensamento, loop agêntico) afetam as taxas de resolução Analisar resultados de benchmark entre modelos e estratégias para encontrar a melhor configuração de agente Medir o trade-off custo-desempenho ? taxas de resolução mais altas custam mais por issue Construir um relatório comparativo de benchmark para selecionar a arquitetura de agente ideal

Introdução

O SWE-bench é um benchmark para avaliar agentes de codificação com IA em issues reais do GitHub. Cada tarefa é um bug genuíno ou solicitação de funcionalidade de repositórios populares de código aberto em Python (Django, scikit-learn, sympy, etc.). O agente deve:

1. Ler a descrição da issue 2. Navegar pela base de código 3. Escrever um patch que corrija a issue 4. Passar na suíte de testes do repositório

Variante do Benchmark | Issues | Dificuldade | Caso de Uso | SWE-bench Full | 2.294 | Mista | Avaliação abrangente | SWE-bench Lite | 300 | Subconjunto curado | Comparação rápida (usado neste lab) | SWE-bench Verified | 500 | Verificado por humanos | Avaliação padrão ouro |

O Cenário

Você é um Arquiteto de Plataforma de IA avaliando agentes de codificação para sua equipe de engenharia. Você realizou benchmark de 8 configurações de agentes em 3 modelos (GPT-4o, o3, Claude 3.5 Sonnet) e 4 estratégias (prompt direto, cadeia de pensamento, loop agêntico, mini SWE-agent). Seu dataset (swe_bench_results.csv) contém os resultados. Sua tarefa: identificar a melhor configuração de agente e entender os trade-offs de custo-desempenho. info "Dados Simulados" Este lab usa resultados simulados de benchmark que refletem as tendências publicadas no leaderboard do SWE-bench. Benchmarking real requer poder computacional significativo ? este dataset simulado permite que você aprenda a metodologia de análise sem o custo.

Pré-requisitos

Requisito | Por quê | Python 3.10+ | Executar os scripts de análise | Biblioteca pandas | Manipulação de dados | [diagram or code sample]

tip "Início Rápido com GitHub Codespaces" Open in GitHub Codespaces

Todas as dependências estão pré-instaladas no devcontainer.

? Arquivos de Apoio

note "Baixe estes arquivos antes de iniciar o lab" Salve todos os arquivos em uma pasta lab-073/ no seu diretório de trabalho.

Arquivo | Descrição | Download | broken_benchmark.py | Exercício de correção de bugs (3 bugs + auto-testes) | ? Download | swe_bench_results.csv | Dataset | ? Download |

Etapas 1: Entender as Estratégias de Agentes

Antes de analisar os resultados, entenda as quatro estratégias sendo avaliadas:

Estratégia | Como Funciona | Desempenho Esperado | Custo Esperado | Direct Prompt | Um único prompt com issue + contexto da base de código ? patch | Mais baixo | Mais baixo |

Chain of Thought | Prompt com etapas de raciocínio explícitas ? patch | Médio | Médio |
Agentic Loop | Loop multi-turno: ler código ? raciocinar ? editar ? testar ? iterar | Mais
alto | Mais alto | Mini SWE-agent | Agente leve com ferramentas de navegação de arquivos e
edição | Médio-Alto | Médio |

Métricas Principais

Métrica | Definição | Resolve Rate | % de issues em que o patch do agente passa em todos os
testes | Avg Time | Média de segundos por tentativa de issue | Avg Cost | Média de USD por
tentativa de issue |

Etapa 2: Carregar e Explorar os Resultados

O dataset possui 8 configurações de agentes testadas no SWE-bench Lite (300 issues) e
Verified (500 issues):

[diagram or code sample]

Saída esperada:

[diagram or code sample]

Etapa 3: Encontrar os Melhores e Piores Agentes

Classifique os agentes por taxa de resolução para encontrar os de melhor desempenho:

[diagram or code sample]

Saída esperada:

Posição | Agente | Modelo | Estratégia | Taxa de Resolução | Custo/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 lidera com 65% ? mas a \$5,50 por issue. Baseline GPT-4o é o mais
barato a \$0,85, mas resolve apenas 30%. A estratégia de loop agêntico supera
consistentemente outras estratégias para o mesmo modelo.

Etapa 4: Medir a Melhoria do Loop Agêntico

Quanto a estratégia de loop agêntico melhora em relação ao baseline (prompt direto) para o
mesmo modelo?

[diagram or code sample]

Saída esperada:

[diagram or code sample]

tip "Insight" O loop agêntico adiciona +20?21 pontos percentuais em todos os modelos. Essa
melhoria consistente sugere que a estratégia (navegação iterativa no código + testes)
importa tanto quanto o modelo subjacente.

Etapa 5: Analisar Trade-offs de Custo-Desempenho

Agentes mais capazes custam mais. Calcule o custo por issue resolvida para encontrar a
opção mais eficiente:

[diagram or code sample]

[diagram or code sample]

warning "A Curva de Custo" Ir de 30% para 65% de taxa de resolução (melhoria de 2,2x) custa
\$5,50 vs. \$0,85 por tentativa (6,5x mais caro). Os retornos decrescentes aparecem com força
? avalie se a melhoria marginal na taxa de resolução justifica o custo para o seu caso de
uso.

Etapa 6: Construir o Relatório de Benchmark

[diagram or code sample]

? Exercício de Correção de Bugs

O arquivo lab-073/broken_benchmark.py contém 3 bugs que produzem análises de benchmark incorretas. Você consegue encontrar e corrigir todos?

Execute os auto-testes para ver quais falham:

[diagram or code sample]

Você deverá ver 3 testes falhando. Cada teste corresponde a um bug:

Teste | O que verifica | Dica | Teste 1 | Seleção do melhor agente | Deve encontrar o agente com a *maior* taxa de resolução, não a menor | Teste 2 | Custo médio por issue resolvida | Deve dividir pela contagem de *resolvidas*, não pelo total de issues | Teste 3 | Comparação agêntico vs. baseline | Deve filtrar por *modelo* antes de comparar estratégias |

Corrija todos os 3 bugs e execute novamente. Quando você vir All passed!, está pronto!

? Verificação de Conhecimento

question "Q1 (Múltipla Escolha): O que o SWE-bench mede sobre um agente de codificação?"

- A) Quão rápido ele consegue gerar código - B) Se ele consegue resolver issues reais do GitHub produzindo patches que passam nos testes - C) Quantas linhas de código ele consegue escrever por minuto - D) Se ele consegue explicar código para um humano

??? success "? Revelar Resposta" Correto: B) Se ele consegue resolver issues reais do GitHub produzindo patches que passam nos testes

O SWE-bench avalia agentes em sua capacidade de corrigir bugs genuínos e implementar funcionalidades de repositórios reais de código aberto. O agente deve produzir um patch, e o patch deve passar na suíte de testes existente do projeto para contar como "resolvido".

question "Q2 (Múltipla Escolha): Por que a estratégia de loop agêntico supera o prompt direto?"

- A) Ela usa uma janela de contexto maior - B) Ela itera: lendo código, raciocinando, editando e testando em um loop - C) Ela usa um modelo mais caro - D) Ela tem acesso à internet

??? success "? Revelar Resposta" Correto: B) Ela itera: lendo código, raciocinando, editando e testando em um loop

O loop agêntico dá ao agente múltiplos turnos para explorar a base de código, formar hipóteses, escrever patches, executar testes e revisar. Isso espelha como desenvolvedores humanos trabalham ? raramente uma tentativa única resolve um bug complexo.

question "Q3 (Execute o Lab): Qual configuração de agente tem a maior taxa de resolução?"

Execute a análise da Etapa 3 no ? swe_bench_results.csv e verifique o ranking.

??? success "? Revelar Resposta" Agentic o3 ? 65%

O agente AG05 ("Agentic o3") usando o modelo o3 com a estratégia agentic_loop resolve 195 de 300 issues (65,0%), a maior de todas as 8 configurações.

question "Q4 (Execute o Lab): Qual configuração de agente tem a menor taxa de resolução?"

Verifique a parte inferior do ranking da Etapa 3.

??? success "? Revelar Resposta" Baseline GPT-4o ? 30%

O agente AG01 ("Baseline GPT-4o") usando o modelo gpt-4o com a estratégia direct_prompt resolve apenas 90 de 300 issues (30,0%). O prompt direto sem iteração produz o menor desempenho.

question "Q5 (Execute o Lab): Quantos pontos percentuais o loop agêntico melhora em relação ao baseline para o mesmo modelo?"

Execute a análise da Etapa 4 para calcular a melhoria agêntica por modelo.

??? success "? Revelar Resposta" +20pp para GPT-4o (30%?50%), +20pp para o3 (45%?65%),

+21pp para Claude (35%?56%)

O loop agêntico adiciona consistentemente 20?21 pontos percentuais de taxa de resolução sobre o baseline de prompt direto, independentemente do modelo subjacente. Isso demonstra que a arquitetura do agente importa tanto quanto a capacidade do modelo.

Resumo

Tópico | O que Você Aprendeu | SWE-bench | Benchmark padrão ouro usando issues reais do GitHub e suítes de testes | Taxa de Resolução | Métrica principal ? % de issues em que o patch do agente passa nos testes | Loop Agêntico | +20pp de melhoria sobre prompt direto para qualquer modelo | Trade-off de Custo | 65% de taxa de resolução custa 6,5x mais por tentativa do que 30% | Modelo vs. Estratégia | A estratégia (loop agêntico) contribui tanto quanto a escolha do modelo | Análise de Benchmark | Como classificar, comparar e relatar sobre configurações de agentes |

Próximos Passos

Lab 035 ? Avaliação de Agentes com Azure AI Eval SDK (avaliação personalizada além do SWE-bench) Lab 038 ? Otimização de Custos de IA (gerenciando o custo de loops agênticos)

Lab 040 ? AutoGen Multi-Agente (construindo loops agênticos com AutoGen) Lab 074 ? Foundry Agent Service (operação e monitoramento de frotas de agentes)

Phase 3 · Simulação / Caminho Real

Simulação Caminho Real

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: Exclua implantações, endpoints, índices, traces e dados de teste após a validação.

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

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		