

Lab 073 : Benchmarking d'agents avec SWE-bench

Niveau : L300 Parcours : Tous les parcours Durée : ~90 min ? Coût : Gratuit ? Utilisez des résultats de benchmark simulés

info "Type de pratique" Ce lab analyse des résultats simulés de type SWE-bench. Il enseigne la méthodologie de benchmark et l'analyse des compromis sans exécuter d'évaluations coûteuses d'agents de codage.

Ce que vous apprendrez

Ce qu'est SWE-bench et pourquoi c'est le standard de référence pour évaluer les agents de codage Comment différentes stratégies d'agent (prompt direct, chaîne de pensée, boucle agentique) affectent les taux de résolution Analyser les résultats de benchmark à travers les modèles et stratégies pour trouver la meilleure configuration d'agent Mesurer le compromis coût-performance ? des taux de résolution plus élevés coûtent plus cher par issue Construire un rapport comparatif de benchmark pour choisir la bonne architecture d'agent

Introduction

SWE-bench est un benchmark pour évaluer les agents de codage IA sur de vrais problèmes GitHub. Chaque tâche est un véritable bug ou une demande de fonctionnalité provenant de dépôts Python open-source populaires (Django, scikit-learn, sympy, etc.). L'agent doit :

1. Lire la description de l'issue 2. Naviguer dans le code source 3. Écrire un correctif qui résout le problème 4. Passer la suite de tests du dépôt

Variante du benchmark | Issues | Difficulté | Cas d'usage | SWE-bench Full | 2 294 | Mixte | Évaluation complète | SWE-bench Lite | 300 | Sous-ensemble organisé | Comparaison rapide (utilisé dans ce lab) | SWE-bench Verified | 500 | Vérifié par des humains | Évaluation de référence |

Le scénario

Vous êtes un Architecte plateforme IA évaluant des agents de codage pour votre équipe d'ingénierie. Vous avez testé 8 configurations d'agents avec 3 modèles (GPT-4o, o3, Claude 3.5 Sonnet) et 4 stratégies (prompt direct, chaîne de pensée, boucle agentique, mini SWE-agent). Votre jeu de données (swe_bench_results.csv) contient les résultats. Votre mission : identifier la meilleure configuration d'agent et comprendre les compromis coût-performance.

info "Données simulées" Ce lab utilise des résultats de benchmark simulés qui reflètent les tendances publiées du classement SWE-bench. Le benchmarking réel nécessite des ressources de calcul significatives ? ce jeu de données simulé vous permet d'apprendre la méthodologie d'analyse sans le coût.

Prérequis

Exigence | Raison | Python 3.10+ | Exécuter les scripts d'analyse | Bibliothèque pandas | Manipulation des données |

[diagram or code sample]

tip "Démarrage rapide avec GitHub Codespaces" Open in GitHub Codespaces

Toutes les dépendances sont pré-installées dans le devcontainer.

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-073/ de votre répertoire de travail.

Fichier | Description | Téléchargement | broken_benchmark.py | Exercice de correction de bugs (3 bugs + auto-tests) | ? Télécharger | swe_bench_results.csv | Jeu de données | ? Télécharger |

Étape 1 : Comprendre les stratégies d'agent

Avant d'analyser les résultats, comprenez les quatre stratégies testées :

Stratégie | Fonctionnement | Performance attendue | Coût attendu | Prompt direct | Un seul prompt avec issue + contexte du code ? correctif | La plus basse | Le plus bas | Chaîne de pensée | Prompt avec étapes de raisonnement explicites ? correctif | Moyenne | Moyen | Boucle agentique | Boucle multi-tours : lire le code ? raisonner ? modifier ? tester ? itérer | La plus haute | Le plus élevé | Mini SWE-agent | Agent léger avec outils de navigation et d'édition de fichiers | Moyenne-haute | Moyen |

Métriques clés

Métrique | Définition | Taux de résolution | % d'issues où le correctif de l'agent passe tous les tests | Temps moyen | Secondes moyennes par tentative de résolution | Coût moyen | USD moyen par tentative de résolution |

Étape 2 : Charger et explorer les résultats

Le jeu de données contient 8 configurations d'agents testées sur SWE-bench Lite (300 issues) et Verified (500 issues) :

[diagram or code sample]

Sortie attendue :

[diagram or code sample]

Étape 3 : Trouver les meilleurs et les pires agents

Classez les agents par taux de résolution pour trouver les plus performants :

[diagram or code sample]

Sortie attendue :

Rang | Agent | Modèle | Stratégie | Taux de résolution | Coût/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 est en tête à 65 % ? mais à 5,50 \$ par issue. Baseline GPT-4o est le moins cher à 0,85 \$ mais ne résout que 30 %. La stratégie de boucle agentique surpasse systématiquement les autres stratégies pour le même modèle.

Étape 4 : Mesurer l'amélioration agentique

Combien la stratégie de boucle agentique améliore-t-elle par rapport à la référence (prompt direct) pour le même modèle ?

[diagram or code sample]

Sortie attendue :

[diagram or code sample]

tip "Insight" La boucle agentique ajoute +20 à 21 points de pourcentage quel que soit le modèle. Cette amélioration constante suggère que la stratégie (navigation itérative du code + tests) compte autant que le modèle sous-jacent.

Étape 5 : Analyser les compromis coût-performance

Les agents plus performants coûtent plus cher. Calculez le coût par issue résolue pour trouver l'option la plus efficace :

[diagram or code sample]

[diagram or code sample]

warning "La courbe des coûts" Passer de 30 % à 65 % de taux de résolution (amélioration de 2,2x) coûte 5,50 \$ contre 0,85 \$ par tentative (6,5x plus cher). Les rendements décroissants interviennent fortement ? évaluez si l'amélioration marginale du taux de

résolution justifie le coût pour votre cas d'usage.

Étape 6 : Construire le rapport de benchmark

[diagram or code sample]

? Exercice de correction de bugs

Le fichier lab-073/broken_benchmark.py contient 3 bugs qui produisent une analyse de benchmark incorrecte. Pouvez-vous les trouver et les corriger tous ?

Exécutez les auto-tests pour voir lesquels échouent :

[diagram or code sample]

Vous devriez voir 3 tests échoués. Chaque test correspond à un bug :

Test | Ce qu'il vérifie | Indice | Test 1 | Sélection du meilleur agent | Devrait trouver

l'agent avec le taux de résolution le *plus élevé*, pas le plus bas | Test 2 | Coût moyen

par issue résolue | Devrait diviser par le nombre de *résolues*, pas le total des issues |

Test 3 | Comparaison agentique vs référence | Devrait filtrer par *modèle* avant de

comparer les stratégies |

Corrigez les 3 bugs, puis relancez. Quand vous voyez All passed!, c'est terminé !

? Vérification des connaissances

question "Q1 (Choix multiple) : Que mesure SWE-bench concernant un agent de codage ?"

- A) La vitesse à laquelle il peut générer du code - B) S'il peut résoudre de vraies issues

GitHub en produisant des correctifs qui passent les tests - C) Combien de lignes de code il

peut écrire par minute - D) S'il peut expliquer du code à un humain

??? success "? Révéler la réponse" Correct : B) S'il peut résoudre de vraies issues GitHub

en produisant des correctifs qui passent les tests

SWE-bench évalue les agents sur leur capacité à corriger de vrais bugs et implémenter des

fonctionnalités provenant de véritables dépôts open-source. L'agent doit produire un

correctif, et le correctif doit passer la suite de tests existante du projet pour être

compté comme « résolu ».

question "Q2 (Choix multiple) : Pourquoi la stratégie de boucle agentique surpasse-t-elle le prompt direct ?"

- A) Elle utilise une fenêtre de contexte plus grande - B) Elle itère : lire le code,

raisonner, modifier et tester dans une boucle - C) Elle utilise un modèle plus coûteux - D)

Elle a accès à internet

??? success "? Révéler la réponse" Correct : B) Elle itère : lire le code, raisonner,

modifier et tester dans une boucle

La boucle agentique donne à l'agent plusieurs tours pour explorer le code, formuler des

hypothèses, écrire des correctifs, exécuter les tests et réviser. Cela reflète la façon

dont les développeurs humains travaillent ? il est rare qu'une tentative en un seul coup

résolve un bug complexe.

question "Q3 (Exécutez le lab) : Quelle configuration d'agent a le taux de résolution le plus élevé ?"

Exécutez l'analyse de l'étape 3 sur ? swe_bench_results.csv et vérifiez le classement.

??? success "? Révéler la réponse" Agentic o3 ? 65 %

L'agent AG05 (« Agentic o3 ») utilisant le modèle o3 avec la stratégie agentic_loop résout

195 des 300 issues (65,0 %), le plus élevé des 8 configurations.

question "Q4 (Exécutez le lab) : Quelle configuration d'agent a le taux de résolution le plus bas ?"

Vérifiez le bas du classement de l'étape 3.

??? success "? Révéler la réponse" Baseline GPT-4o ? 30 %

L'agent AG01 (« Baseline GPT-4o ») utilisant le modèle gpt-4o avec la stratégie

direct_prompt ne résout que 90 des 300 issues (30,0 %). Le prompt direct sans itération produit la performance la plus basse.

question "Q5 (Exécutez le lab) : De combien de points de pourcentage la boucle agentique améliore-t-elle par rapport à la référence pour le même modèle ?"

Exécutez l'analyse de l'étape 4 pour calculer l'amélioration agentique par modèle.

??? success "" Révéler la réponse" +20pp pour GPT-4o (30 %?50 %), +20pp pour o3 (45 %?65 %), +21pp pour Claude (35 %?56 %)

La boucle agentique ajoute systématiquement 20 à 21 points de pourcentage de taux de résolution par rapport à la référence en prompt direct, quel que soit le modèle sous-jacent. Cela démontre que l'architecture de l'agent compte autant que la capacité du modèle.

Résumé

Sujet | Ce que vous avez appris | SWE-bench | Benchmark de référence utilisant de vraies issues GitHub et suites de tests | Taux de résolution | Métrique principale ? % d'issues où le correctif de l'agent passe les tests | Boucle agentique | +20pp d'amélioration par rapport au prompt direct pour tout modèle | Compromis de coût | 65 % de taux de résolution coûte 6,5x plus cher par tentative que 30 % | Modèle vs stratégie | La stratégie (boucle agentique) contribue autant que le choix du modèle | Analyse de benchmark | Comment classer, comparer et rendre compte des configurations d'agents |

Prochaines étapes

Lab 035 ? Évaluation des agents avec le SDK Azure AI Eval (évaluation personnalisée au-delà de SWE-bench) Lab 038 ? Optimisation des coûts IA (gérer le coût des boucles agentiques)

Lab 040 ? Multi-agents AutoGen (construire des boucles agentiques avec AutoGen) Lab 074 ?

Foundry Agent Service (exploiter et surveiller des flottes d'agents)

Phase 3 · Simulation / Parcours Réel

Simulation Parcours Réel

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: Supprimez déploiements, points de terminaison, index, traces et données de test après validation.

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

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\n
nA1,Base GPT,gpt-4o,100,30,30.0,0.85,direct_prompt\n
A2,Agent GPT,gpt-4o,100,50,50.0,2.50,agentic_loop\n
A3,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

```