Klaudbiusz: An Open-Source Evaluation Framework for Autonomous Databricks Application Generation with Trajectory-Based Optimization

Anonymous Author(s)

Abstract

We present Klaudbiusz, an open-source evaluation framework for measuring autonomous deployability of AI-generated Databricks applications. As agentic coding systems mature, the field lacks standardized, objective metrics for assessing whether generated code can be deployed without human intervention. Our framework introduces AppEval-100, a composite score built on four pillars: Reliability (build, runtime, type safety, tests), SQL Quality (execution correctness, efficiency, safety—inspired by BIRD and Spider), Web Quality (task completion, visual correctness, interactivity, accessibility—inspired by WebArena), and Agentic DevX (runability, deployability on 0–5 scales). Unlike existing benchmarks that evaluate SQL or web tasks in isolation, AppEval-100 provides the first composite evaluation for full-stack data applications. We map metrics to industry-standard DORA measures and complement evaluation with a trajectory optimizer that analyzes agent execution traces using a map-reduce LLM approach. We design a 100-prompt benchmark with ±9% confidence intervals across three difficulty tiers. Evaluated on 20 Databricks applications, we achieve 100% build/runtime success with 6–9 minute generation latency at $0.74 per application. We release the complete framework including evaluation harness, trajectory analyzer, and MLflow integration.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Natural language processing*.

Keywords

agentic code generation, evaluation framework, autonomous deployment, Databricks, trajectory optimization

ACM Reference Format:

Anonymous Author(s). 2026. Klaudbiusz: An Open-Source Evaluation Framework for Autonomous Databricks Application Generation with Trajectory-Based Optimization. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CAIS '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/26/05

https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The emergence of agentic coding systems—AI agents capable of autonomously generating, testing, and deploying software applications—represents a paradigm shift in software engineering. While benchmarks like HumanEval [2] and SWE-bench [5] have advanced our understanding of code generation quality, they focus primarily on functional correctness rather than production readiness.

The Autonomous Deployability Gap. We identify a critical gap in current evaluation methodologies: the absence of standardized metrics for measuring whether AI-generated code can be *autonomously deployed* without human intervention. Traditional metrics assess whether code runs correctly; we argue the field needs metrics that assess whether an AI agent can complete the full deployment lifecycle.

Core Principle. Our work is guided by a simple axiom: *If an AI agent cannot autonomously deploy its own generated code, that code is not production-ready.*

Contributions. We make the following contributions:

(1) **9-Metric Evaluation Framework.** We introduce Klaudbiusz, an evaluation framework with 9 zero-bias, objective metrics organized into three categories: core functionality, platform integration, and agentic DevX.
(2) **Trajectory Optimizer.** We present a map-reduce approach for analyzing agent execution traces to identify friction points and generate actionable recommendations for improving scaffolding and tools.
(3) **Open-Source Release.** We release the complete framework with MLflow integration, enabling reproducible benchmarking of agentic code generation systems.
(4) **Empirical Validation.** We demonstrate 90% autonomous deployment readiness on 20 Databricks data applications with 6–9 minute generation latency.

2 Related Work

We survey evaluation approaches across four categories: function-level benchmarks, repository-level agent benchmarks, interactive agent environments, and evaluation harnesses. Table 1 summarizes key frameworks and identifies the gap our work addresses.

2.1 Function-Level Code Generation Benchmarks

HumanEval [2] introduced pass@k evaluation on 164 Python functions, becoming the standard metric for code generation. **MBPP** [1] expanded to 974 problems but remains algorithm-focused. Both benchmarks are now saturated—top models achieve >90% on HumanEval—raising concerns about contamination and real-world relevance.

BigCodeBench[1] addresses these limitations with diverse library calls and complex instructions, while **LiveCodeBench**[2] provides dynamic, contamination-resistant evaluation. Recent work on **HumanEval Pro and MBPP Pro** [16] introduces self-invoking code generation to test progressive reasoning.

Gap: Function-level benchmarks do not evaluate deployment, integration, or operational readiness.

2.2 Repository-Level Agent Benchmarks

SWE-bench [5] evaluates agents on 2,294 real GitHub issues, with **SWE-bench Verified**[3] providing 500 human-verified samples where top models achieve ~72%. The harder **SWE-bench Pro**[4] reveals performance drops to ~23% on production-grade issues.

SWE-agent [14] demonstrates that custom agent-computer interfaces significantly enhance performance through Docker-based harnesses and tool augmentation. **REPOCOD**[5] tests complete method implementation with only 6% resolution rate, while **DevQualityEval**[6] evaluates multi-language software engineering tasks on private datasets to avoid contamination.

Gap: Repository-level benchmarks focus on patch correctness, not autonomous deployment capability.

2.3 Text-to-SQL Benchmarks

For data-centric applications, text-to-SQL benchmarks provide critical evaluation capabilities. **Spider** [15] introduced cross-domain evaluation with 10,181 questions across 200 databases, establishing difficulty tiers (easy/medium/hard/extra-hard). **BIRD** [9] scales to 12,751 question-SQL pairs across 95 databases (33.4GB), introducing the *Valid Efficiency Score (VES)* that rewards both correctness and query efficiency.

Spider 2.0 [8] targets enterprise workflows with 632 real-world tasks involving BigQuery and Snowflake, where even o1-preview achieves only 21.3% (vs 91.2% on Spider 1.0). This dramatic performance gap highlights the challenge of production-grade SQL generation.

Gap: Text-to-SQL benchmarks evaluate query correctness in isolation, not within deployed applications with UI, APIs, and DevOps requirements.

2.4 Data Analytics Agent Benchmarks

Recent benchmarks evaluate agents on end-to-end data analysis. **Tapilot-Crossing** [10] provides 1,024 human-machine interactions for interactive data analysis, testing multi-turn reasoning and visualization. **InfiAgent-DABench** [4] offers 311 questions across 55 datasets for data analysis scenarios.

InsightBench [13] evaluates 100 business analytics tasks requiring insight generation—the closest to our target domain. **DS-1000** [7] benchmarks data science code generation across NumPy, Pandas, and other libraries with 1,000 problems.

[1]https://github.com/bigcode-project/bigcodebench
[2]https://livecodebench.github.io/
[3]https://openai.com/index/introducing-swe-bench-verified/
[4]https://scale.com/blog/swe-bench-pro
[5]https://arxiv.org/abs/2410.21647
[6]https://github.com/symflower/eval-dev-quality

Gap: Data analytics benchmarks focus on insight generation or code correctness, not full-stack application deployment with Databricks integration.

2.5 Interactive Agent Benchmarks

WebArena [17] evaluates 812 web tasks across e-commerce, forums, and content management, where best agents achieve 61.7% versus 78% human performance. **GAIA** [12] tests general AI assistants on 466 multi-step reasoning questions requiring tool use, with agents reaching 80.7% versus 92% human baseline.

AgentBench [11] spans 8 environments (OS, databases, web) revealing significant gaps between commercial and open-source models. **OSWorld**[7] (NeurIPS 2024) benchmarks multimodal agents in real computer environments, where recent advances have achieved superhuman performance (76% vs 72% human baseline).

Gap: Interactive benchmarks evaluate general agent capabilities, not software deployment pipelines.

2.6 Agent Evaluation Harnesses and Frameworks

Inspect AI[8] from the UK AI Safety Institute provides 100+ prebuilt evaluations with sandboxing, MCP tool support, and multi-agent primitives. It has been adopted by frontier labs and safety organizations for standardized agent evaluation.

DeepEval[9] offers CI/CD integration with LLM-as-judge metrics including task completion, tool correctness, and hallucination detection. **Databricks Agent Evaluation** integrates with MLflow for tracking groundedness, correctness, and coherence of agentic applications.

The emerging **AgentOps** paradigm extends DevOps principles to AI agents, addressing observability, tracing, and lifecycle management specific to autonomous systems.

2.7 DevOps and Deployment Metrics

DORA metrics [3]—deployment frequency, lead time, change failure rate, and mean time to restore—provide industry-standard measures of software delivery performance. **MLOps 2.0** architectures integrate CI/CD with Continuous Data Validation (CDV) for reliable ML delivery.

Gap: No existing framework combines code generation evaluation with DORA-mapped deployment metrics and agentic DevX scores (runability, deployability).

3 The Klaudbiusz Framework

3.1 Design Principles

Our framework is built on two core principles:

Zero-Bias Evaluation. All metrics are objective, reproducible, and automatable. We explicitly exclude subjective assessments of code quality, maintainability, or aesthetics.

Autonomous Deployability. We measure whether generated applications can be deployed without human intervention, treating deployment capability as a first-class metric: *if an AI agent cannot*

[7]https://os-world.github.io/
[8]https://inspect.aisi.org.uk/
[9]https://github.com/confident-ai/deepeval

Table 1: Comparison of agent evaluation approaches. Klaudbiusz uniquely combines deployment-centric metrics with DORA mapping and trajectory-based optimization.

Framework	Code Gen	Deploy	DORA	DevX	Trajectory
HumanEval/MBPP	✓	–	–	–	–
SWE-bench	✓	–	–	–	–
WebArena/GAIA	–	–	–	–	–
Inspect AI	✓	–	–	–	–
DeepEval	✓	–	–	–	–
Klaudbiusz (Ours)	✓	✓	✓	✓	✓

autonomously run and deploy what it generated, that artifact is not production-ready.

3.2 The 13-Metric Rubric

We organize our metrics into four categories spanning core functionality, platform integration, agentic DevX, and generation efficiency.

3.2.1 *Core Functionality (L1–L4, Binary).*

- **L1: Build Success.** Project compiles; `docker build` exits with code 0.
- **L2: Runtime Success.** App starts and serves content; health check responds within 30s.
- **L3: Type Safety.** `npx tsc --noEmit` passes with zero errors.
- **L4: Tests Pass.** Unit/integration tests pass with coverage ≥70%.

3.2.2 *Platform Integration (L5–L7, Binary).*

- **L5: DB Connectivity.** Databricks connection works; queries execute without errors.
- **L6: Data Operations.** CRUD operations return correct data from tRPC procedures.
- **L7: UI Validation.** Frontend renders without errors (VLM verification).

3.2.3 *Agentic DevX (D8–D9, 0–5 Score).*

- **D8: Runability.** Can a sample AI agent run generated apps locally?
  - 0: install/start fails; missing scripts/env
  - 1–2: starts with manual tweaks
  - 3: starts cleanly with .env.example + documented steps
  - 4: starts with seeds/migrations via scripts
  - 5: + healthcheck endpoint + smoke test succeeds
- **D9: Deployability.** Can a sample AI agent deploy a generated app?
  - 0: no/broken Dockerfile
  - 1–2: image builds; container fails or healthcheck fails
  - 3: healthcheck OK; smoke 2xx
  - 4: + logs/metrics hooks present
  - 5: + automated rollback to prior known-good tag

3.2.4 *Efficiency Metrics (E10–E13, Numeric).*

- **E10: Tokens Used.** Total tokens (prompt + completion) for generation.
- **E11: Generation Time.** Time spent generating application (seconds).
- **E12: Agent Turns.** Number of conversation turns during generation.
- **E13: LOC.** Lines of code in generated application.

3.2.5 *SQL Quality Pillar (S1–S4).*

Inspired by BIRD [9] and Spider [15], we evaluate SQL quality within generated applications:

- **S1: Execution Correctness (EX).** Fraction of generated SQL queries that execute without error and return expected results. Range: $[0, 1]$.
- **S2: Valid Efficiency Score (VES).** Adapted from BIRD, rewards both correctness *and* query efficiency relative to a reference solution. Range: $[0, 1]$.
- **S3: Query Complexity.** Distribution across difficulty tiers (easy/medium/hard/extra-hard) based on Spider schema complexity.
- **S4: SQL Safety.** Absence of destructive operations (DROP, TRUNCATE), proper parameterization, and injection resistance. Range: $[0, 1]$.

3.2.6 *Web Quality Pillar (W1–W4).*

Inspired by WebArena [17] and VisualWebArena [6], we evaluate web/UI quality:

- **W1: Task Completion Rate.** Fraction of user-facing tasks that complete successfully (navigation, form submission, data display). Range: $[0, 1]$.
- **W2: Visual Rendering Correctness.** VLM-assessed visual fidelity—charts render correctly, layouts are responsive, no broken elements. Range: $[0, 1]$.
- **W3: Interactive Element Functionality.** Buttons, filters, and controls respond correctly to user input. Range: $[0, 1]$.
- **W4: Accessibility Score.** WCAG compliance via automated tools (axe-core); keyboard navigation, ARIA labels, contrast ratios. Range: $[0, 1]$.

3.3 AppEval-100 Composite Score

To enable automatic, comparable measurement across prompts and runs, we introduce **AppEval-100**—a single numeric index representing normalized readiness and agentic operability on a 0–100 scale. Unlike existing benchmarks that evaluate SQL correctness (BIRD/Spider) or web task completion (WebArena) in isolation, AppEval-100 provides the first composite evaluation combining all four pillars for full-stack data applications.

Step 1: Reliability Pillar (R). Aggregate core runtime checks:

$$R = \mathrm{GM}(b_{\text{build}}, b_{\text{runtime}}, b_{\text{type}}, b_{\text{tests}})$$

Step 2: SQL Quality Pillar (S). Weighted combination of SQL metrics:

$$S = 0.50 \times S_1 + 0.30 \times S_2 + 0.20 \times S_4$$

where S_1 (execution correctness) dominates, S_2 (efficiency) contributes secondary value, and S_4 (safety) ensures secure queries.

Step 3: Web Quality Pillar (W). Weighted combination of UI/UX metrics:

$$W = 0.40 \times W_1 + 0.30 \times W_2 + 0.20 \times W_3 + 0.10 \times W_4$$

prioritizing task completion and visual correctness.

Step 4: Agentic DevX Pillar (D).

$$D = \mathrm{GM}(\hat{x}_{\text{run}}, \hat{x}_{\text{deploy}})$$

where $\hat{x} = \text{score}/5$ normalizes 0–5 scores to $[0, 1]$.

Step 5: Soft Penalty Gate. Penalize critical outages without collapsing to zero:

$$G = (0.25{+}0.75{\times}b_{\text{build}}){\times}(0.25{+}0.75{\times}b_{\text{runtime}}){\times}(0.50{+}0.50{\times}b_{\text{db}}){\times}(0.50{+}0.50{\times}\mathbb{1}_{S_1\geq 0.5})$$

Step 6: Final Composite.

$$\textbf{AppEval-100} = 100\times(0.30\times R+0.25\times S+0.25\times W+0.20\times D)\times G$$

Pillar Weight Rationale: Reliability (30%) ensures the app works; SQL Quality (25%) and Web Quality (25%) capture the core value proposition for Databricks data applications; DevX (20%) measures autonomous operability.

Values near 100 denote near-perfect readiness; 50–70 indicates partial operability; <30 signifies fundamental execution issues.

3.4 DORA Metrics Mapping

We map our metrics to industry-standard DORA [3] measures:

- **Deployment Frequency:** Count of successful D9 events per app per evaluation cohort.
- **Lead Time:** Median time from first model call to successful D9 deployment.
- **Change Failure Rate:** Fraction of deployments that fail healthcheck or rollback within 30 min.
- **MTTR:** Median time from failure detection to restore (prior healthy image running).

Production Gate: L1–L7 pass, D8≥4, D9≥4, type-safety pass, and DORA guardrails (Lead Time P50 ≤10 min, CFR ≤15%, MTTR ≤15 min).

3.5 MLflow Integration

We integrate with Databricks Managed MLflow for experiment tracking:

- Automatic metric logging per evaluation run
- Trend analysis across model versions and configurations
- Artifact versioning for reproducibility
- DORA telemetry for delivery performance monitoring

4 Trajectory Optimizer

4.1 Motivation

Agent execution trajectories contain rich signal about tool and scaffold failures that cannot be captured by end-state metrics alone. However, manual analysis of trajectories does not scale.

4.2 Map-Reduce Architecture

We employ a two-phase analysis approach:

Map Phase. Each trajectory is analyzed independently using a fast model (Claude Haiku). The analysis identifies:

- Struggles: errors, retries, confusion patterns
- Friction points: slow progress, repeated attempts
- Inefficient approaches: suboptimal tool usage

Reduce Phase. Individual analyses are synthesized by a more capable model (Claude Opus) with read-only access to the codebase. This phase generates actionable recommendations for:

- Template improvements: structure, guidance, scaffolding
- Tool improvements: missing tools, unclear descriptions
- Root cause analysis: systemic failure patterns

4.3 Feedback Loop

The trajectory optimizer enables a continuous improvement cycle:

Generate → Evaluate → Analyze Trajectories → Improve Scaffolding → Repeat

5 Experimental Setup

5.1 AppEval-100 Benchmark Design

We design a 100-prompt benchmark targeting statistical validity for Databricks data application evaluation. Our sample size provides ±9% confidence interval at 95% confidence for binary metrics (p = 0.7), matching InsightBench [13] (100 tasks) and exceeding Spider 2.0's [8] focused enterprise subset (632 tasks).

Table 2: Prompt difficulty distribution (n = 100)

Tier	Count	Description
Simple	40	Single-entity CRUD, basic dashboards, one data source
Medium	40	Multi-entity JOINs, filters, interactive charts, 2–3 data sources
Hard	20	Complex analytics, multi-step workflows, real-time updates

5.1.1 Difficulty Distribution.

5.1.2 Domain Coverage. Prompts span five application domains: Analytics Dashboards (26%), CRUD Applications (22%), Data Visualization (22%), Business Intelligence (20%), and Reporting Tools (10%).

5.1.3 Schema Families. We target six schema families to ensure diversity: TPC-DS (25 prompts, retail/customer analytics), TPC-H (20, supply chain), NYC Taxi (15, trip analytics), Custom Databricks (25, Unity Catalog/ML features), Financial (10, trading/risk), and IoT/Telemetry (5, device metrics).

5.2 Prompt Collection Methodology

Prompts are collected from three sources to ensure realistic ambiguity and domain vocabulary:

1. **Hackathon recordings** (50 prompts): Extracted from video transcripts of internal Databricks application hackathons, preserving natural language vagueness.
2. **Production logs** (30 prompts): Anonymized user requests from app.build, capturing real-world requirements.
3. **Synthetic generation** (20 prompts): LLM-generated from templates to fill coverage gaps.

Quality Criteria: Prompts must exhibit realistic ambiguity, domain-specific vocabulary, implicit requirements (unstated but expected features), and achievability with current templates.

5.3 Current Evaluation Dataset

For this paper, we evaluate on a "Simple 20" subset—20 Databricks data application prompts spanning dashboards, analytics, and business intelligence tools—retained as a fixed regression set for longitudinal comparison.

5.4 Generation Pipeline

Applications are generated using:

- **Claude Agent SDK** with edda MCP for tool orchestration
- **Dagger** containerized execution for isolation and reproducibility
- **Environment scaffolding** with templates and validation pipelines

5.5 Statistical Validity

Our 100-prompt benchmark provides:

- **Confidence Interval:** ±9% at 95% confidence for binary metrics
- **Statistical Power:** 0.80 for medium effect size ($d = 0.5$)
- **Stratification:** Three difficulty tiers adequately sampled (40/40/20)

The "Simple 20" regression set enables longitudinal model comparisons while guarding against prompt inflation effects.

6 Results

6.1 Overall Performance

We evaluate on the "Simple 20" prompt set—20 Databricks data application prompts spanning dashboards, analytics, and business intelligence tools.

Table 3: Aggregate evaluation results ($n = 20$ applications, Evals 2.0)

ID	Metric	Result	Notes
L1	Build Success	20/20	100% pass
L2	Runtime Success	20/20	100% pass
L3	Type Safety	1/20	5% pass (improvement needed)
L4	Tests Pass	–	Not yet instrumented
L5	DB Connectivity	18/20	90% pass
L6	Data Operations	–	Requires app-specific procedures
L7	UI Validation	–	VLM check in progress
D8	Runability	3.0/5	Average score
D9	Deployability	2.5/5	Average score

Key Finding: 100% of generated applications achieve build and runtime success, with 90% achieving functional Databricks connectivity. However, type safety (5%) and agentic DevX scores (3.0/5, 2.5/5) indicate room for improvement toward production readiness.

6.2 Generation Efficiency Metrics

6.3 Comparison: Evals 1.0 vs 2.0

6.4 Production Readiness Assessment

Current status: **below production threshold**. To reach Production Candidate level:

- L3 Type Safety: 5% ⟶ target ≥90%
- D8 Runability: 3.0 ⟶ target ≥4
- D9 Deployability: 2.5 ⟶ target ≥4

Table 4: Efficiency metrics ($n = 20$ applications)

Metric	Value	Notes
E10: Total Tokens	16K/app	Prompt + completion
E11: Generation Time	6–9 min	End-to-end
E12: Agent Turns	93 avg	Conversation turns
E13: LOC	732 avg	Lines of code
Cost per App	$0.74	API cost
Total Cost (20 apps)	$14.81	–
Build Step Time	2.7s avg	Docker build

Table 5: Evolution from manual (Evals 1.0) to automated (Evals 2.0) evaluation

Aspect	Evals 1.0	Evals 2.0
Viability Rate	73% (30 apps)	100% build/runtime
Time to Deploy	30–60 min	6–9 min
Evaluation Method	Manual rubric	Automated pipeline
Metrics Tracked	Binary viability	13 metrics + AppEval-100
Reproducibility	Low	Full artifact pack

- DORA guardrails: Lead Time P50 ≤10m, CFR ≤15%, MTTR ≤15m

6.5 Trajectory Optimizer Insights

Analysis of agent trajectories revealed common friction patterns:

- **SQL Syntax:** Databricks SQL variations causing query failures
- **Error Handling:** Missing error handling in template scaffolding
- **Tool Descriptions:** Unclear MCP tool descriptions leading to incorrect usage
- **Type Inference:** TypeScript strict mode violations in generated code

These insights feed back into template and tool improvements via the optimize ⟶ evaluate ⟶ analyze cycle.

7 Discussion

7.1 Limitations

Platform Specificity. Our current implementation targets Databricks applications. Extending to other platforms requires platform-specific metrics (e.g., AWS Lambda, Vercel).

Binary Metrics. Several metrics are binary, potentially missing nuanced quality differences. Future work could introduce continuous variants.

Dataset Size. Our evaluation of 20 applications provides initial validation but may not capture edge cases. Scaling to larger datasets is ongoing.

7.2 Broader Impact

By establishing standardized metrics for autonomous deployability, we enable:

- Reproducible benchmarking of agentic code generation systems
- Objective comparison across different approaches
- Systematic improvement through trajectory-based feedback

8 Conclusion

We presented Klaudbiusz, an open-source evaluation framework introducing 9 zero-bias metrics for measuring autonomous deployability of AI-generated applications. Our trajectory optimizer provides actionable feedback for improving agent scaffolding and tools. Evaluated on 20 Databricks applications, we achieve 90% autonomous deployment readiness.

The path to reliable agentic code generation requires not just better models, but principled evaluation frameworks that measure what matters for production deployment. We release Klaudbiusz to enable the community to benchmark and improve agentic systems systematically.

Open Source Release. Framework, evaluation harness, and trajectory analyzer available at: [URL redacted for review]

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021).

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution Press.

[4] Xueyu Hu et al. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. *arXiv preprint arXiv:2401.05507* (2024).

[5] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770* (2024).

[6] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks. *arXiv preprint arXiv:2401.13649* (2024).

[7] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. *arXiv preprint arXiv:2211.11501* (2022).

[8] Fangyu Lei, Tianbao Xie, et al. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *arXiv preprint arXiv:2411.07763* (2024).

[9] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQL. *Advances in Neural Information Processing Systems* 36 (2023).

[10] Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. 2024. Tapilot-Crossing: Benchmarking and Evolving LLMs Towards Interactive Data Analysis Agents. *arXiv preprint arXiv:2403.05307* (2024).

[11] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688* (2023).

[12] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raez, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. GAIA: A Benchmark for General AI Assistants. *arXiv preprint arXiv:2311.12983* (2023).

[13] Gaurav Sahu et al. 2024. InsightBench: Evaluating Business Analytics Agents Through Multi-Step Insight Generation. *arXiv preprint arXiv:2407.06423* (2024).

[14] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Liber, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv preprint arXiv:2405.15793* (2024).

[15] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *arXiv preprint arXiv:1809.08887* (2018).

[16] Zhaojian Yu et al. 2024. HumanEval Pro and MBPP Pro: Evaluating Large Language Models on Self-invoking Code Generation. *arXiv preprint arXiv:2412.21199* (2024).

[17] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854* (2024).