AppBench: Benchmarking AI-Generated Web Applications

Brendan McLaughlin, Belinda Mo, Abhinav Lalwani, Ethan Hellman

Department of Computer Science, Stanford

PROBLEM

As web app generation rapidly advances, there remains a gap in comprehensive evaluation frameworks which effectively assess technical quality + adherence to user specifications and requirements. Our project creates a benchmark that assesses AI-generated web applications across multiple evaluation axes and difficulty levels. We aim to answer the following questions:

1. How can we construct a dataset to evaluate the ability of frontier models to build high quality web apps?

2. How can we build a system that conducts fair evaluations of web application quality & alignment with user specifications?

DATASET

We design a high-quality, high-diversity, and challenging benchmark dataset that:

a) Must be feasible to autonomously evaluate
b) Must maximally span the diversity of real-world web apps

To start, we write 9 app-generation user queries, using 6 evaluation axes:

1. **UI Complexity** – Visual and structural sophistication
2. **Feature Coverage & Functionality** – Breadth of supported capabilities
3. **State Management** – Handling of user interactions and data persistence
4. **API Integration** – Ability to connect with external services
5. **Cross-Page Functionality** – Navigation and multi-page interactions
6. **Data Processing** – Handling and transformation of structured data

There are 5 difficulty levels, ranging from **L0** (basic static apps) to **L4** (advanced interactive app with complex integration)

Example prompts:

- L0: "Generate a simple landing page with a header, centered welcome message, and footer."
- L2: "Create a page that includes a search box; when a user enters a city name and clicks a button, fetch and display basic weather data from an the OpenWeatherMap API using this api key: <redacted api key>"
- L4: "Design a web app with real-time stock price updates, interactive line charts displaying trends, and a form for submitting trade requests with validation and detailed error messages."

EVALUATORS

Each example in the APP-bench dataset has a corresponding evaluation script that flexibly draws from a library of evaluation tools. The evaluation tools are all implemented on top of Playwright. At the most granular level, the evaluator script can invoke a tool that intercepts and breaks API network requests to test how the user interface handles errors. At the most general level, the evaluator script can invoke a language model-powered browser agent to navigate ambiguity on the web app.

Example - Mobile Landing Page UI: Score: 6.8/7 Notes:
✓ Search input field found in mobile view.
✓ Search button found in mobile view.
✓ Page title found in mobile view.
Clear, well-organized layout. Search functionality prominently accessible. Minor overlap issue with *'Edit with lovable'* button slightly detracts from UX.

EXPERIMENTS

We evaluated 7 different agents (Replit, Bolt, Cursor Agent, Loveable, Claude Sonnet 3.7, GPT 4.5, and OpenHands CodeAct 2.1) across our benchmark dataset. We analyzed performance both by individual agent and by agent type (Closed-Source, Zero-Shot, and Open-Source).

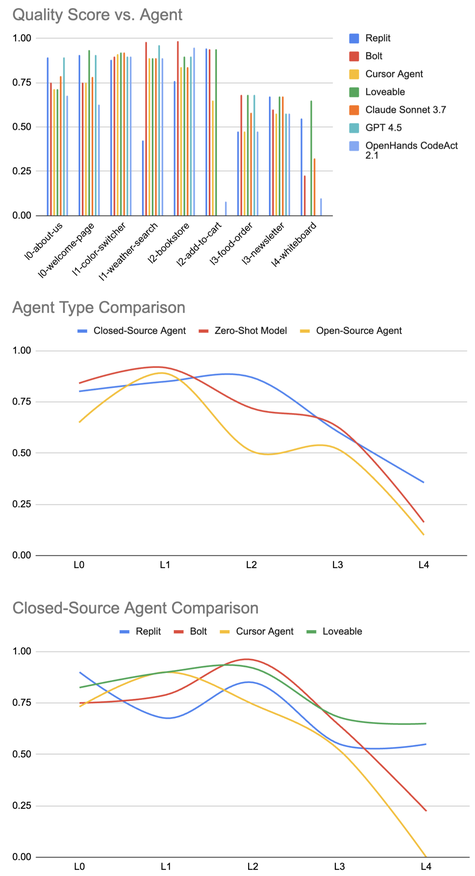
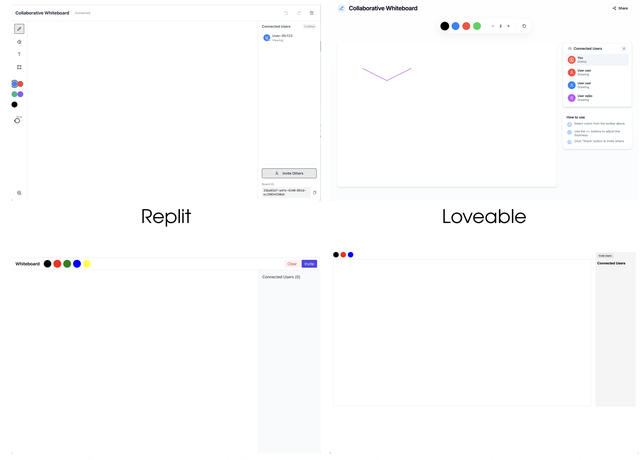


Figure 1. Comparison Charts

QUALITATIVE RESULTS OF AGENT GENERATIONS

Table 1. *

Qualitative Comparison of 4 Evaluated Code Agents



DISCUSSION

Key findings from our experiments:

- Zero-shot models demonstrate surprisingly competitive performance compared to closed-source agents, despite their simpler architecture and lower compute budget.
- Loveable and Bolt emerge as the top-performing agents across our test set.
- All agent types show stronger performance (>75% quality score) up to difficulty level L2.
- Performance degradation begins at L3, with a sharp decline at L4 across all agent types. 2 of 7 agents were unable to generate a compileable application for 1 L4 and 1 L3 task.
- The gap between agent types narrows at higher difficulty levels, suggesting fundamental limitations in current app generation capabilities.

FUTURE WORK

- Add additional prompt-evaluator pairs for higher coverage across the dataset
- Improve evaluator script quality by adding additional tooling capabilities and stricter criteria
- Experiment with improving an evaluator agent that may run more accurately
- Do more fine-grained experiments to quantitatively show model's performance across the defined axes