AutoWeb: A Conversational and Evolutionary Multimodal Web-Building Agent

Shaobo Liang
Johns Hopkins University
sliang24@jhu.edu

Ruifeng Ji
Johns Hopkins University
rji10@jh.edu

Abstract

We present **AutoWeb**, a conversational, multimodal, and *evolutionary* web-building agent that constructs, edits, and iteratively refines websites through natural language interactions. Unlike traditional one-shot code-generation systems, AutoWeb performs incremental synthesis, maintains long-term project state, manages reusable components, and even modifies its *own interface* in response to user intent. This work extends the Web-App Agent (WAA) from the Machine Programming course into a full self-evolving system capable of interactive program synthesis, multimodal layout parsing, and UI self-modification. AutoWeb demonstrates a new paradigm in human–AI collaborative programming in which the toolchain is no longer static but continuously adapts to the user's workflow.

1 Introduction

Large Language Models (LLMs) such as Codex (1), StarCoder (2), and GPT-series models have demonstrated impressive ability to generate and manipulate code from natural language. However, most systems still follow a *single-shot* interaction paradigm: the user provides a prompt, and the model returns a code artifact. Practical software development rarely follows such a linear structure. Instead, it is iterative, conversational, and dependent on context accumulated across multiple refinements.

Recent works in agentic reasoning (3) and neural program synthesis (4) suggest that LLMs can manage multi-step reasoning processes and modify artifacts iteratively. Inspired by this, we introduce **AutoWeb**, an LLM-driven system that constructs websites over a multi-turn conversation while maintaining structured state across files, pages, components, and user preferences.

AutoWeb goes further: it is *self-evolutionary*. The system exposes a second instruction channel enabling users to modify the tool *itself*: adding new UI elements, new modalities (e.g., file upload, video input), and new interactive workflows through natural language. Thus, AutoWeb becomes a platform for creating both websites and the interface used to build them.

2 Related Works

2.1 LLMs for Code Generation

Codex (1) and StarCoder (2) established modern benchmarks for LLM-driven coding systems. These models demonstrate strong performance on text-to-code tasks but operate mainly in one-shot or few-shot settings without persistent state or cross-file reasoning. Our work builds on these capabilities while addressing the need for incremental, stateful editing.

Preprint. Under review.

2.2 Agentic Reasoning and Tool Use

ReAct (3) introduced a unified framework for reasoning + acting, enabling models to call tools based on intermediate reasoning traces. AutoWeb uses a similar pattern: the agent reasons about user requests, selects tool calls (filesystem edits, component registration, UI updates), and reflects on prior actions.

2.3 Program Synthesis and Structure-Aware Generation

Neural program synthesis research, including Yin and Neubig (4), highlights the importance of syntactic structure and incremental editing in code generation. Also relevant is Gulwani's work on example-driven synthesis (5), which emphasizes structured program transformations. AutoWeb extends these ideas into a multi-file, multi-page, interactive environment.

3 Methodology

3.1 System Overview

AutoWeb operates through an agentic loop consisting of:

- **Instruction Parsing:** The agent receives textual or multimodal instructions (wireframes).
- **Planning:** The model decomposes instructions into actionable steps.
- **Tool Invocation:** Tools such as `fs.write`, `component.register`, `page.register`, or `ui.update` are invoked.
- **State Update:** The project directory, registry files, and UI configuration are updated accordingly.
- **Reflection:** History compression maintains long-term context.

3.2 Evolutionary User Interface

A key innovation is AutoWeb's ability to modify its own UI. Users issue meta-instructions like:

> "Add a file upload panel to the interface"

AutoWeb then:

1. Updates `.waa/ui_config.json`.
2. Regenerates the frontend using `ui_builder.py`.
3. Reloads the UI, exposing new capabilities.

This allows AutoWeb to become a fully personalized tool-building environment.

3.3 Component and Page System

Components (like navbars, footers) are stored in `components/` and recorded in a structured registry. Pages are tracked in `.waa/pages.json`. This system enables:

- Multi-page site generation
- Reusable elements
- Automated inclusion through a client-side loader

3.4 Multimodal Wireframe Parsing

AutoWeb accepts JSON-encoded wireframes that describe page layout structure. The `layout_parser.py` module translates these into HTML/CSS. This enables sketch-to-code workflows similar to prior multimodal synthesis paradigms.

4 Evaluation

We evaluate AutoWeb along four axes.

4.1 Functional Correctness

Generated websites consistently rendered correctly and matched user instructions. Multi-page navigation, component loading, and CSS structure all behaved as expected.

4.2 Interactivity

The conversational loop successfully supported iterative refinement such as:

> "Make the hero section blue."
> "Add a contact form."

Targeted diffs were produced rather than full regeneration.

4.3 Self-Evolution Capability

AutoWeb successfully modified its own UI:

- Added new input panels (file upload, color pickers)
- Changed its theme (dark/light)
- Rearranged interface layout

4.4 Error Handling

When Gemini API rate limits were reached, AutoWeb generated clean JSON errors and preserved server stability.

5 Case Study

A full interactive session showed AutoWeb generating:

- An initial project scaffold
- A multi-page portfolio
- A reusable navbar component
- A wireframe-based landing page

Then, when instructed to "add video upload capability," AutoWeb rebuilt its UI with a new form field—demonstrating the evolutionary pipeline.

6 Conclusion and Future Work

AutoWeb demonstrates that LLM-driven agents can support not only dynamic web development workflows but also the self-modification of their own toolchain. This represents a shift from static code-generation tools to adaptive, co-evolving programming partners.

Future directions include:

- Integrating more expressive multimodal input (full sketches, PDFs, screenshots)
- Allowing AutoWeb to refactor or optimize its own backend logic
- Learning user preferences over long-term usage
- Extending AutoWeb to mobile app design or desktop app generation

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, and many others. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021.

[2] Raymond Li, Leandro von Werra, Thomas Wolf, and colleagues. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[3] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022.

[4] Pengcheng Yin and Graham Neubig. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[5] Sumit Gulwani. Automating String Processing in Spreadsheets Using Input-output Examples. In *Proceedings of POPL*, pages 317–330, 2011.