International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 04, April 2026)

Intellicode: An AI-Powered, Privacy-Preserving, Browser-Based Coding Playground Using WebLLM and Monaco Editor

Mahesh Hasabe[1], Prem Sakhare[2], Swapnil Chavan[3], Ritesh Patil[4], Harshal Patil[5]

[1]*Assistant Professor,*[2,3,4,5]*Student, Computer Science & Engineering, Sant Gajanan Maharaj College of Engineering, Mahagaon, Maharashtra, India.*

***Abstract*—This paper describes Intellicode, a browser-based coding playground that puts a full-featured code editor, live multi-language execution, and an AI chat assistant together in a single web page—no server, no install. The AI assistant runs entirely inside the browser using WebLLM, which loads a quantized large language model (LLM) via WebGPU. Code editing is handled by Monaco Editor, the engine underlying VS Code. Execution goes through the Judge0 API, which supports over 60 languages. State is managed with Zustand, the UI is built in React and TypeScript, and everything is bundled with Vite. This paper covers the architecture, the technology choices, design trade-offs, performance realities, and where the project can go next.**

***Keywords*—AI-assisted coding, WebLLM, Monaco Editor, Judge0, in-browser LLM, browser IDE, code review, TypeScript, React, Zustand, Vite, client-side execution**

I. Introduction

AI developer tools have changed how engineers write and debug code, but nearly all of them still depend on a local runtime, a package manager, a server, or some combination of the three. That is a genuine barrier for students, for anyone working across multiple machines, and for anyone in an environment where software installation is not permitted.

Intellicode removes that friction. It is a coding playground that runs entirely in the browser: a professional editor, live code execution across dozens of languages, and an AI assistant that performs its inference locally without sending anything to an external API. Open a tab, start coding.

This paper is organized as follows: Section II covers related work. Section III describes the system architecture. Section IV goes through the technology stack and performance. Section V outlines use cases. Section VI compares Intellicode against similar platforms. Section VII addresses current limitations. Section VIII describes planned improvements, and Section IX concludes.

II. Background And Related Work

A. Evolution of Online Code Editors

Browser-based coding tools started as syntax-highlighted text areas with a Run button. JSFiddle (2010) and CodePen (2012) made the concept popular for front-end work.

Replit, CodeSandbox, and StackBlitz later extended it to full-stack development, package management, and collaborative editing [10]. Nearly all of these platforms depend on server-side infrastructure, which introduces latency, running costs, and data privacy trade-offs.

B. AI-Assisted Development Tools

LLM-driven tools—GitHub Copilot, Amazon CodeWhisperer, Tabnine—have demonstrated real productivity gains: code completions, bug detection, and function generation from natural language [9]. The trade-off is that every interaction goes through a cloud API, which is a problem for proprietary codebases or limited-bandwidth environments.

C. WebLLM and In-Browser AI

WebLLM, developed by the MLC AI team, runs quantized LLMs directly in the browser via WebGPU [1]. Model weights are downloaded once and cached in IndexedDB; subsequent sessions skip the download and start in seconds. The result is near-native inference speed with no network dependency and no data leaving the device.

D. Judge0 API for Code Execution

Judge0 is an open-source code execution system supporting over 60 programming languages [2]. It exposes a REST API that accepts source code, compiles and runs it in a sandboxed container, and returns stdout, stderr, and execution metadata. Competitive programming platforms and online judges use it widely for its reliability and language breadth.

III. System Architecture And Design

A. Architectural Overview

Intellicode is a single-page application with no custom backend. All logic runs in the browser. There are four core subsystems: Monaco Editor for code editing, Judge0 for execution, WebLLM for AI assistance, and Zustand for state management. The UI is built with React 18 and TypeScript, bundled by Vite, and styled with Tailwind CSS.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 04, April 2026)

B. Frontend Layer

React and TypeScript together provide component-driven UI with compile-time type safety. Vite handles development (fast hot module replacement) and production (optimized bundles). The interface has three panes: the Monaco Editor on the left, an output console below it, and an AI chat panel on the right.

C. Monaco Editor Integration

Monaco Editor is the open-source core of Visual Studio Code [3]. In Intellicode it provides syntax highlighting for all supported languages, auto-completion and parameter hints, inline error diagnostics, and the keyboard shortcuts VS Code users already know. It loads as an npm package and renders as a React component wired directly into Zustand state.

D. Code Execution via Judge0

Pressing Run reads the code and selected language from Zustand, then sends an authenticated request to Judge0's /submissions endpoint. The payload includes source code, language ID, and any stdin provided. Results are polled asynchronously; stdout, stderr, and compile errors all appear in the output console.

E. AI Assistant via WebLLM

WebLLM downloads a quantized LLM on first use and caches it in IndexedDB. From that point on, inference is local and works offline. The assistant handles five tasks: explaining code, catching bugs, suggesting improvements, translating between programming languages, and answering general programming questions. Nothing leaves the browser tab—the full inference loop runs client-side.

F. State Management with Zustand

Zustand is a lightweight, hook-based state library for React [4]. It holds the current editor content, selected language, execution results and errors, AI chat history, and loading flags for both execution and inference. Centralizing this state keeps the three panes in sync without prop drilling or the boilerplate that comes with Redux.

IV. Technical Analysis

A. Technology Stack

TABLE I
TECHNOLOGY STACK SUMMARY

Layer	Technology	Version	Purpose
Language	TypeScript	5.x	Type-safe JS across the codebase
UI Framework	React.js	18.x	Component-driven rendering
Build Tool	Vite	5.x	Fast HMR and production bundling
Editor	Monaco Editor	Latest	VS Code-grade in-browser editing

B. Performance Considerations

Running both Monaco and a WebLLM model in the same tab places real demands on memory and GPU. Four specifics worth noting:

Model loading. The first-time setup requires downloading model weights, ranging from 500 MB to 4 GB depending on the model. After that, the IndexedDB cache brings startup time down to seconds.

WebGPU utilization. WebLLM uses TVM compilation [6] to produce optimized GPU kernels, reaching inference speeds comparable to native execution on a mid-range desktop GPU.

Monaco lazy loading. The editor loads asynchronously so the page becomes interactive before the AI model finishes initializing.

Execution latency. Judge0 API calls typically add 500 ms to 2 s of network round-trip, depending on server load and geographic proximity.

V. Use Cases And Applications

A. Education and Learning

Students can write, run, and get AI explanations for code without installing anything. Instructors can share links to code snippets for classroom demonstrations.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 04, April 2026)

The AI assistant functions as an on-demand tutor—explaining concepts, catching errors, and walking through logic at the student's pace.

B. Code Review and Interview Preparation

Developers preparing for technical interviews can solve algorithm problems, test across languages, and get immediate AI feedback. The assistant can comment on time and space complexity, suggest alternative approaches, and explain why a given solution works or falls short.

C. Rapid Prototyping and Code Migration

Intellicode eliminates environment setup for quick experiments. The AI assistant can translate code between languages—useful when a team migrates from JavaScript to TypeScript or from Python 2 to Python 3—producing a working equivalent with a brief explanation of what changed.

D. Accessibility in Restricted Environments

In corporate or educational settings where software installation is not permitted, Intellicode works in any modern browser without administrator rights. That makes it available in many places where other tools cannot be deployed at all.

VI. Comparative Analysis

TABLE II
PLATFORM COMPARISON

Feature	Intellicode	Replit	CodeSandbox	JSFiddle
Backend Required	No	Yes	Yes	Partial
AI Assistant	In-browser (WebLLM)	Cloud AI	Cloud AI	None
Code Execution	Judge0 API	Server-side	Server-side	Browser-only
Editor Quality	Monaco (VS Code)	Monaco	Monaco	CodeMirror
Languages	60+	50+	JS/TS focused	HTML/CSS/JS
AI Privacy	Fully local	Cloud-dependent	Cloud-dependent	N/A
Open Source	Yes	No	Partial	No
Offline Capable	Yes (post-cache)	No	No	Limited

Intellicode's main advantage is the combination of a fully client-side architecture and a locally-running AI assistant. Replit offers stronger collaboration features; GitHub Codespaces provides deeper cloud integration. Intellicode trades those capabilities for zero setup, no data leaving the browser, and no dependency on an external service staying online.

VII. Limitations And Challenges

A. WebGPU Browser Requirements

WebLLM requires WebGPU support, available in Chrome 113+, Edge 113+, and experimental Firefox and Safari builds. Older browsers and non-GPU devices see degraded AI performance. Smaller, CPU-optimized models via WebAssembly partially address this gap.

B. Model Download Size

The initial model download ranges from 500 MB to 4 GB. Smaller quantized variants (3-bit or 4-bit) reduce the barrier for users on slower connections, but there is no avoiding some upfront cost on first use.

C. Judge0 API Rate Limits

The public Judge0 API has rate limits that become a problem for users who run code frequently. A production deployment would need a self-hosted Judge0 instance or a paid tier to guarantee reliable execution throughput.

D. No Collaboration Features

Intellicode is currently single-user. Adding real-time collaborative editing via WebRTC and CRDTs (such as Yjs) would require a coordination server, which conflicts with the purely client-side design. It is doable but would require rethinking part of the architecture.

E. Stateless Chat History

AI chat history exists only for the current session. A page refresh clears it. Multi-session workflows are not supported in the current implementation unless a save-and-load mechanism is added explicitly.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 04, April 2026)

VIII. Future Work And Enhancements

TABLE III
PLANNED ENHANCEMENTS

Enhancement	Description	Priority
Collaborative Editing	Real-time multi-user editing via WebRTC and CRDTs (e.g., Yjs)	High
Model Selection	Let users pick from multiple WebLLM models by capability or size	High
AI Code Generation	Generate entire functions from natural language descriptions	High
Persistent Snippets	Save and reload code via localStorage or cloud sync	Medium
File System Emulation	Multi-file projects with a virtual file tree (Origin Private File System API)	Medium
Mobile Optimization	Responsive layout for tablet and mobile	Medium
Offline Mode	Full offline support after initial model and asset caching	Medium
Version History	Edit history with diff view	Low
Plugin System	Community extensions for language support, themes, and tools	Low

The three highest-priority items are collaborative editing, model selection, and AI code generation. Yjs-based real-time collaboration via WebRTC would open Intellicode to classrooms and pair-programming sessions. Letting users choose a model—trading download size for capability—would accommodate a wider range of hardware. Generating functions from natural language descriptions is the natural next step for browser-based AI tooling and something users will expect as the space matures.

IX. Conclusion

Intellicode demonstrates that the browser is capable enough to host a real development environment—professional editor, multi-language execution, and a locally-running AI assistant—without any backend infrastructure.

WebGPU inference, Monaco editing, and the Judge0 execution API have all matured to the point where this is a working tool, not a proof of concept.

The privacy argument is straightforward: nothing leaves the tab. For students, that means no account required. For developers, it means proprietary code stays off external servers. For anyone in a restricted environment, it means access to a capable coding tool through a standard browser.

As WebGPU support broadens, quantization techniques improve [7][8], and browser APIs expand, the distance between browser-based and desktop development will keep shrinking. Intellicode is built to grow with that shift.

Acknowledgment

The authors—Prof. M. K. Hasabe, Sakhare Prem Sachin, Patil Harshal Mohan, Chavan Swapnil Sambhaji, and Patil Ritesh Sarjerao Department of Computer Science and Engineering, Sant Gajanan Maharaj College of Engineering, Mahagaon, for his guidance and support throughout this work. The authors also acknowledge the open-source communities behind WebLLM, Monaco Editor, Judge0, Zustand, Vite, and Tailwind CSS.

REFERENCES

[1] MLC AI Team, "WebLLM: High-Performance In-Browser LLM Inference Engine," 2024. [Online]. Available: https://webllm.mlc.ai/

[2] Judge0, "Judge0 CE - Open Source Online Code Execution System," 2024. [Online]. Available: https://judge0.com/

[3] Microsoft, "Monaco Editor: The Code Editor that Powers VS Code," 2024. [Online]. Available: https://microsoft.github.io/monaco-editor/

[4] Zustand Contributors, "Zustand: A Small, Fast and Scalable State-Management Solution," 2024. [Online]. Available: https://zustand-demo.pmnd.rs/

[5] Vite Contributors, "Vite: Next Generation Frontend Tooling," 2024. [Online]. Available: https://vitejs.dev/

[6] T. Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in Proc. 13th USENIX Symp. OSDI, 2018.

[7] Y. Shi et al., "Efficient LLM Inference on GPUs via Compilation," arXiv:2406.01705, 2024.

[8] P. Liang et al., "Holistic Evaluation of Language Models (HELM)," arXiv:2211.09110, 2022.

[9] GitHub Copilot Team, "GitHub Copilot: Your AI Pair Programmer," 2023. [Online]. Available: https://github.com/features/copilot

[10] Replit, "Replit — The World's Most Used Collaborative Online IDE," 2024. [Online]. Available: https://replit.com/