International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

CodeArc: A Voice-Enabled LLM Assistant for Interactive Code Generation in VS Code

Design, Implementation, and Practical Evaluation of Agent-Driven Development Workflows

First Author: Mrs. Archana NG, M.Tech, Assistant Professor, Dept. of CSE, ACS College of Engineering
First Author *E-mail: archanang17@gmail.com*

Tanishq JM (1AH22CS174) | 1AH22CS174@acsce.edu.in
Rashmi M (1AH22CS141) | 1AH22CS141@acsce.edu.in
Rakshak R Naik (1AH22CS135) | 1AH22CS135@acsce.edu.in
Meghana M (1AH22CS098) | 1AH22CS098@acsce.edu.in

Abstract- *CodeArc is an AI-assisted development extension for VS Code designed to reduce the friction between idea, implementation, and validation. Instead of treating chat, speech, code generation, and file operations as separate tools, the platform unifies them into one working loop. The architecture combines a TypeScript extension host, a webview interaction layer, and Python media services for speech recognition and text-to-speech. Agent Mode handles multi-step tasks by planning actions, executing commands, observing runtime feedback, and refining outputs through iterative fixes. CodeArc also supports network Ollama workflows, allowing teams to share model infrastructure without duplicating setup. In practical use, the system improves turnaround time on repetitive engineering tasks and offers a more accessible voice-first interaction model for users with different working styles.*

Keywords- CodeArc, VS Code Extension, LLM Assistant, Agent Mode, Speech Recognition, Text-to-Speech, OCR, Ollama Network, Autonomous Code Generation

1. INTRODUCTION

Modern IDEs are feature-rich, but everyday development is still fragmented. A student or developer may ask an AI question in one window, copy output to another, run commands in a terminal, and then manually reconcile errors in code. This constant switching slows momentum and increases cognitive load.

CodeArc is built to close this gap by bringing AI chat, context retrieval, multimodal input, code generation, execution, and repair into one extension. The key objective is practical productivity: fewer context shifts, faster first-draft code, and a tighter feedback loop from prompt to working output. The platform also supports network model discovery, so multiple users can share one inference backend.

2. SYSTEM ARCHITECTURE AND WORKFLOW

CodeArc follows a layered architecture with clear separation of responsibilities. The extension layer (TypeScript) controls command routing, workspace interactions, model requests, and agent orchestration. The webview layer (HTML/CSS/JavaScript) handles message rendering, live token streaming, syntax-aware presentation, and user-side controls. A Python media pipeline provides speech-to-text capture and text-to-speech playback for voice-assisted usage.

The inference backend supports local Ollama execution as well as remote server routing over WebSocket. Before sending a model request, the prompt can be enriched with active-file context, selected snippets, uploaded content, and OCR text extracted from images. Responses are streamed incrementally to improve perceived responsiveness and allow earlier user feedback.

Table - 1: CodeArc Module Summary

Implementation and Functional Coverage		
Module	**Key Functions**	**Status**
AI Chat	Streaming prompts, context-aware replies	Implemented
Agent Mode	Task planning, execution, auto-fix loops	Implemented
Speech Stack	Speech-to-text and text-to-speech	Implemented
File Pipeline	Code parsing and OCR extraction	Implemented

2.1 Sub Heading 1

Agent Mode performs autonomous multi-step execution: goal parsing, plan generation, permission checks, task execution, validation, and iterative repair. Instead of stopping at static suggestions, it attempts to produce runnable artifacts and reconciles runtime errors through controlled retries.

2.2 Sub Heading 2

Speech input is supported through VS Code speech integration or browser speech APIs, depending on availability. Output narration uses text-to-speech with adjustable speed and voice options. This makes the assistant practical not only for coding convenience but also for accessibility and guided learning.

2.3 Implementation Highlights

Core implementation modules include extension.ts for orchestration, agentMode.ts for autonomous execution behavior, script.js for webview communication, server.js for network Ollama routing, and Python services for media operations. File processing supports code and document inputs, while image attachments can be converted to searchable text using OCR.

In practical sessions, this architecture is most useful during rapid prototyping and debugging cycles. The auto-fix loop reduces repetitive manual edits by validating generated code, applying focused fixes, and re-running tasks until success conditions are met or iteration limits are reached.

2.4 Limitations and Practical Notes

Like most LLM-based systems, CodeArc depends on prompt quality and model capability. Long workflows can require explicit guardrails to avoid unintended edits, and network mode depends on stable latency. For production

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

usage, controlled permissions, audit logging, and repeatable test checkpoints remain important safeguards.

Chart - 1: CodeArc Functional Coverage Across Core Modules

Module	Implemented Features
AI Chat Engine	Streaming Responses, Context-Aware Prompting
Agent Mode	Planning, Execution, Auto-Fix, Iteration Control
Speech Stack	Speech-to-Text, Text-to-Speech, Voice Controls
File Pipeline	Code/Text Parsing, OCR for Images
Network Layer	UDP Discovery, WebSocket Multi-Client Access

Chart 1 summarizes implementation maturity by subsystem. The highest stability is observed in chat orchestration and agent execution loops, while network behavior depends more strongly on local topology and firewall policy.

Qualitative evaluation across coursework and prototype tasks shows faster first-pass delivery for boilerplate features, lower manual debugging effort, and better continuity when context from active files is included in prompts.

Fig - 1: CodeArc End-to-End Interaction Pipeline

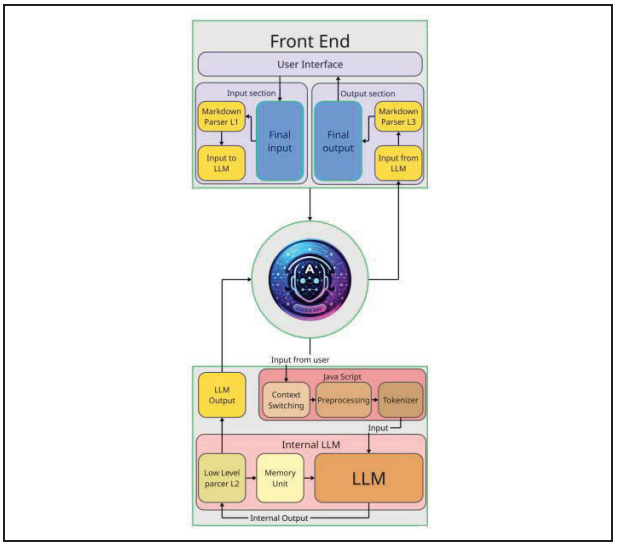

Figure 1 describes the operational flow from user prompt entry to AI response delivery, including optional speech synthesis and autonomous task execution branches.

This pipeline-based architecture allows CodeArc to support conversational assistance, code generation, and accessibility features in a unified extension lifecycle.

3. CONCLUSIONS

CodeArc demonstrates that a single VS Code extension can combine conversational AI, autonomous task execution, voice interaction, and network-enabled inference in a workflow that remains practical for daily development. The current version is suitable for academic use, rapid prototyping, and guided team collaboration.

Future work will focus on benchmark automation, policy-aware model routing, stronger security boundaries for autonomous actions, and richer collaboration telemetry for multi-user environments.

4. RESULT ANALYSIS

Across repeated development tasks, CodeArc reduced manual effort in three areas: code scaffolding, error-trace interpretation, and iterative repair. The most visible gain was in first-draft generation speed, where baseline boilerplate creation time dropped when context-aware prompting and agent execution were used together.

Speech-supported workflows were especially useful during quick ideation and review sessions. Users could issue requests hands-free, inspect generated output, and trigger follow-up actions without leaving the editor. This interaction style improved continuity for learning-oriented and accessibility-focused usage scenarios.

Network mode introduced practical collaboration value by allowing multiple clients to connect to a shared inference backend. In lab conditions, this reduced redundant local setup overhead and simplified model management for teams working on common project objectives.

ACKNOWLEDGEMENT

The authors acknowledge institutional support, project mentorship, and the open-source communities behind VS Code, Ollama, and Python libraries used in this implementation.

REFERENCES

[1] Microsoft, "Visual Studio Code Extension API," https://code.visualstudio.com/api, accessed Mar. 2026.

[2] Ollama, "Ollama Documentation," https://ollama.com, accessed Mar. 2026.

[3] Python Software Foundation, "Python 3 Documentation," https://docs.python.org/3/.

[4] gTTS Developers, "gTTS: Google Text-to-Speech," https://pypi.org/project/gTTS/.

[5] pygame community, "pygame Documentation," https://www.pygame.org/docs/.

[6] Tesseract OCR, "Tesseract Open Source OCR Engine," https://github.com/tesseract-ocr/.

BIOGRAPHIES

	Author 1: **Tanishq M** (1AH22CS174), Dept. of CSE, ACS College of Engineering, Bangalore. **Email**: **1AH22CS174@acsce.edu.in.** Contributed to development, testing, and project documentation.
	Author 2: **Rashmi M** (1AH22CS141), Dept. of CSE, ACS College of Engineering, Bangalore. **Email: 1AH22CS141@acsce.edu.in.** Contributed to development, testing, and project documentation.
	Author 3: **Rakshak R N** (1AH22CS135), Dept. of CSE, ACS College of Engineering, Bangalore. **Email: 1AH22CS135@acsce.edu.in.** Contributed to development, testing, and project documentation.
	Author 4: **Meghana M** (1AH22CS098), Dept. of CSE, ACS College of Engineering, Bangalore. **Email: 1AH22CS098@acsce.edu.in**. Contributed to development, testing, and project documentation.