AlphaStack: Autonomous Code Generation via Multi-Agent Systems with Iterative Self-Healing

AlphaStack Team

Abstract

We introduce AlphaStack, a novel approach to autonomous code generation using multi-agent systems with iterative self-healing and comprehensive validation across diverse programming paradigms. By separating planning and correction concerns, AlphaStack achieves high success rates in generating production-ready codebases. Our system features an intelligent multi-agent architecture, comprehensive code generation capabilities, and a Docker-based validation framework. We evaluate AlphaStack on a custom benchmark of 40 programming challenges across CUDA, Go, Rust, and TypeScript, demonstrating its effectiveness in handling complex software engineering tasks.

Introduction

The generation of complete, production-ready codebases from natural language descriptions remains a significant challenge in AI-assisted software development. While current models excel at generating code snippets, they often struggle with multi-file projects, dependency management, and build configurations.

AlphaStack addresses these challenges through an intelligent multi-agent architecture that includes a Planning Agent for error analysis and a Correction Agent for executing fixes. The system employs iterative self-healing to automatically detect and resolve dependency conflicts, build errors, and test failures. Furthermore, AlphaStack utilizes Docker-based validation to ensure that generated projects are not only syntactically correct but also functional in isolated environments.

Methodology

Multi-Agent Architecture
AlphaStack's core innovation lies in its multi-agent system:
- **Planning Agent:** Analyzes errors and generates comprehensive fix strategies using tool-augmented reasoning. It maintains a cache of the project structure to enable efficient planning.
- **Correction Agent:** Executes the fixes proposed by the Planning Agent. It validates code changes before application and uses language-specific parsers to prevent syntax errors.

Iterative Self-Healing
The system operates in a loop of generation, validation, and correction. If a build or test fails, the Planning Agent analyzes the error logs, and the Correction Agent applies the necessary fixes. This process continues until the project builds and passes all tests, or a maximum number of iterations is reached.

Docker-Based Validation
To ensure reproducibility and security, all generated projects are validated within Docker containers. This provides isolated build and test environments with resource management (configurable CPU/memory limits).

Architecture

The architecture of AlphaStack is designed to streamline the flow from natural language input to a production-ready project. The process involves blueprint generation, multi-file code generation, dependency resolution, Docker configuration, and iterative validation.

```
graph LR
    A[Natural Language Input] --> B[AI Analysis & Blueprint]
    B --> C[Multi-File Code Generation]
    C --> D[Dependency Resolution]
    D --> E[Docker Configuration]
    E --> F[Build Validation]
    F --> G{Build Success?}
    G -->|No| H[Planning Agent]
    H --> I[Correction Agent]
    I --> F
    G -->|Yes| J[Test Execution]
    J --> K{Tests Pass?}
    K -->|No| H
    K -->|Yes| L[Production-Ready Project]
```

Figure 1: AlphaStack Architecture Diagram (Mermaid Source)

Results

We evaluated AlphaStack using models such as GPT-5.2, Claude Sonnet 4.6, GLM-5, and MinimaxM2.5 on two key benchmarks: HumanEval (Pass@1 %) and MDDP Score.

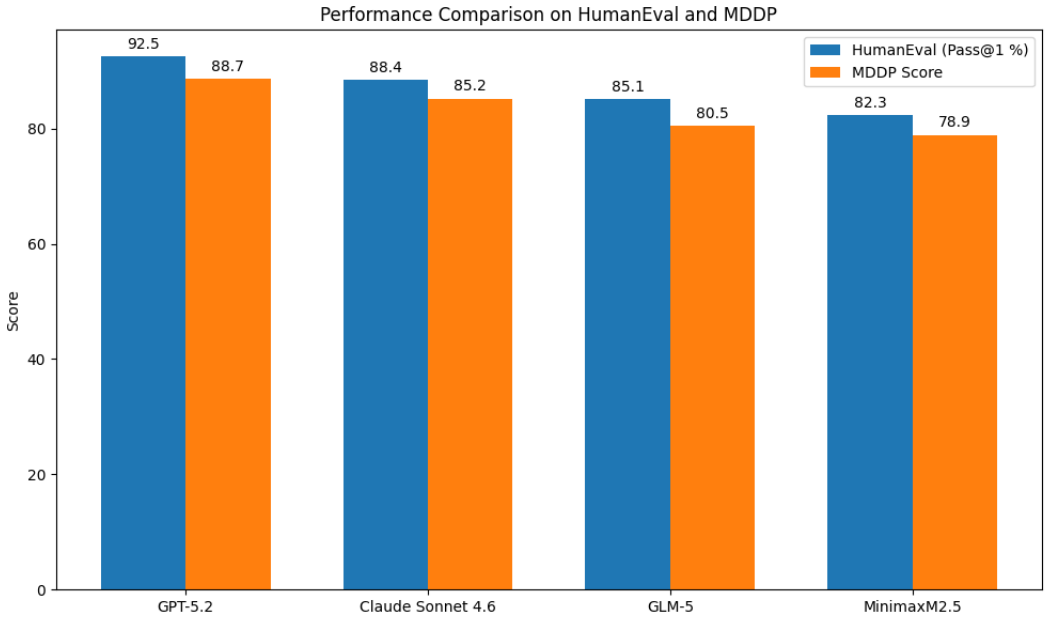

Figure 2: Performance Comparison on HumanEval and MDDP

Conclusion

AlphaStack presents a robust solution for autonomous project generation. By leveraging multi-agent systems and iterative self-healing, it effectively bridges the gap between natural language requirements and functional, production-ready code. Future work will focus on expanding language support and

integrating more advanced reasoning capabilities into the Planning Agent.