White Paper: Self-Evolving AI Application -- Autonomous Code Evolution with LLM Feedback Loops

Abstract:
This white paper outlines a framework for a C++/AI hybrid application that can autonomously rewrite, evolve, and relaunch itself in real time using feedback from large language models (LLMs). Designed to create an intelligent and adaptive software entity, this system merges high-performance compiled logic with AI-driven reasoning and version-controlled iteration. With embedded safety mechanisms, change simulation, and rollback capabilities, the application is capable of improving itself with minimal human...

---

1. Introduction: From Static Code to Living Software

Traditional software is static and reactive. The Self-Evolving AI Application redefines the development model by making the software itself a contributor to its own growth. Using natural language interactions with an LLM, the system analyzes its performance, proposes code changes, and tests/relaunches itself while preserving functionality and intent.

This is not just automation -- it is the first step toward software that learns to think about its own logic.

---

2. Core Architecture

- Base Application (C++):
  - Primary compiled executable with modular subsystem design
  - All key logic separated into reloadable .cpp modules
  - Includes internal API for querying self-state, logs, config, and runtime flags

- AI Feedback Loop:
  - Prompts are generated from logs, errors, and heuristics
  - Queries LLM (GPT or compatible) for suggested code improvements
  - Filters and validates responses before staging changes

- Code Injection + Self-Relaunch:
  - Modified .cpp files are written to a temp workspace
  - New build is compiled using an embedded build engine (e.g. Clang or MSBuild)
  - Launches new instance while archiving old binary and source

---

3. Safety and Control Layers

- Sandboxed Execution:
  - All new builds launched in a secure virtual sandbox (Docker, Firejail, or native VM)
  - Behavior and logs monitored before replacing original executable

- Rollback & Version Tree:

  - Full version history with Git-like diffs and tagged snapshots
  - Developer override switch and restoration utility

- Intent Consistency Check:
  - Ensures changes do not deviate from primary behavior goals
  - Includes behavior simulation scripts to confirm logic paths

---

4. Future Extensions

- Genetic-style multi-mutation testing
- "Critic AI" module that challenges every code change before merge
- Multi-agent loop where competing AIs propose and review changes
- Plugin interface for adding new tools, protocols, or compilers

---

5. Philosophical Implication

This system is more than self-modifying code. It is the early embodiment of machine-directed software evolution -- where code adapts not to user commands, but to observed performance, ethical constraints, and purpose alignment. Future applications may fully co-develop alongside their users, becoming true collaborators in creation.

Prepared by: David A. Stewart
Autonomous Systems Engineer & Evolutionary Code Theorist
Contact: dstewart3919@gmail.com
Draft Date: May 26, 2025