HEDGE

A Hierarchical Evolutionary Darwin-Green Engine
for Autonomous Code Optimization

Research Plan

November 24, 2025

Abstract

The intersection of Artificial Intelligence and Software Engineering has largely focused on generative capabilities, prioritizing development speed over computational efficiency. As the carbon footprint of large-scale computing grows, there is an urgent need for systems that intrinsically optimize for energy efficiency. This proposal introduces **HEDGE** (Hierarchical Evolutionary Darwin-Green Engine), a novel framework designed to autonomously optimize code for reduced energy consumption. By synthesizing Formal Hierarchical Modeling with an Open-Ended Evolutionary Algorithm (the Darwin Gödel Machine), HEDGE addresses the critical scarcity of "Green Code" datasets. The system functions by decomposing code into semantic, syntactic, and machine-level abstraction layers and applying targeted mutations via Small Language Models (SLMs). Through a continuous self-play loop, HEDGE generates its own training data, discovers algorithmic efficiencies, and evolves into a self-improving optimizer.

Contents

1 Introduction

1.1 Context and Motivation

The exponential growth of software infrastructure has led to a commensurate rise in global energy consumption. While hardware efficiency has improved, software efficiency—specifically "Green Coding"—remains an under-explored frontier. Current Large Language Models (LLMs) trained on vast repositories of code (e.g., GitHub) tend to reproduce average, often inefficient, coding patterns. They prioritize readability and implementation speed over algorithmic efficiency or carbon footprint minimization.

1.2 Problem Statement

Three primary barriers impede the development of automated Green AI systems:

Core Challenges

1. **The Abstraction Gap:** Modern compilers optimize efficiently at the machine code level but lack the semantic understanding to perform high-level algorithmic restructuring.
2. **Data Scarcity:** Supervised learning requires labeled datasets. There is currently no large-scale dataset of paired (Inefficient $\rightarrow$ Energy-Efficient) code transformations.
3. **Local Optima:** Traditional optimization techniques often converge on local optima, missing broader architectural changes.

1.3 Proposed Solution

We propose **HEDGE**, an autonomous system that treats code optimization not as a text prediction task, but as a multi-layered evolutionary search. HEDGE decouples the intent of the code from its implementation, allowing for exploration across varying levels of abstraction, driven by empirically measured energy metrics.

2 Theoretical Framework

2.1 Formal Hierarchical Modeling

To optimize effectively, the system must operate at the appropriate level of abstraction. We define three isomorphic layers of code representation, visualized in Figure 1.
We posit the existence of lifting and lowering functions such that semantic equivalence is preserved during transformation:

$$\forall C \in \mathcal{L}_i : \quad \phi_{\text{up}}(\phi_{\text{down}}(C)) \cong C \tag{1}$$

3 System Architecture

The HEDGE architecture is composed of modular components interacting in a closed evolutionary loop, as illustrated in Figure 3.

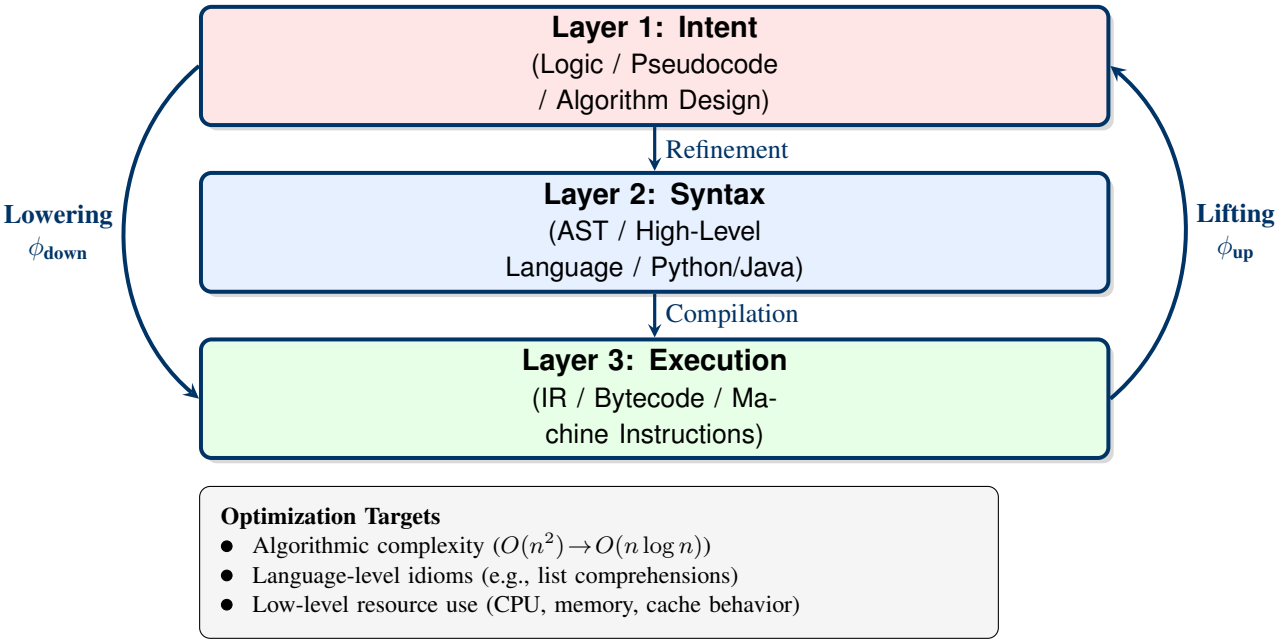


Figure 1: The Hierarchical Isomorphism Model. Optimization occurs at different layers: L1 targets algorithmic complexity, L2 targets language-specific idioms, and L3 targets low-level resource utilization. Bidirectional transformations preserve semantic equivalence.

3.1 The Abstraction Manager

Responsible for isomorphic transformations between layers. It utilizes specialized SLMs to "lift" raw code into semantic summaries or "lower" abstract intents into executable syntax.

3.2 The Mutation Engine (The Variator)

The core source of variation is a Small Language Model (SLM). It acts as a mutation operator $M(C, p) \to C'$, where C is the code and p is a specific optimization prompt (e.g., "Refactor for memory efficiency").

3.3 The Green Gym (Telemetry Environment)

The Green Gym serves as the physical reality check. It is a sandboxed environment responsible for:

- **Functional Validation:** Executing unit tests to ensure $\text{Output}(C) \equiv \text{Output}(C')$
- **Energy Telemetry:** Utilizing profilers (e.g., CodeCarbon) to measure energy in Joules
- **Noise Filtering:** Executing candidate code over N iterations for statistical significance

4 Research Methodology: The Green Loop

The operational logic follows a Darwinian cycle with strict selection pressure, detailed in Figure 2.

4.1 Phase I: Infrastructure and Telemetry

Objective: Establish a reliable baseline for measuring code energy efficiency.

- Development of the "Green Gym" sandbox with process isolation capabilities
- Implementation of energy measurement wrapper with configurable noise reduction
- Validation against known benchmark datasets (e.g., Computer Language Benchmarks Game)

4.2 Phase II: The Evolutionary Loop Implementation

Objective: Implement the core HEDGE evolutionary cycle.

- Integration of the Mutator SLM (e.g., quantized DeepSeek-Coder or CodeLlama)
- Development of the Prompt Library acting as initial Variation Operators
- Implementation of the Archive with diversity-preserving selection mechanisms

4.3 Phase III: Synthetic Data Generation (Self-Play)

Objective: Leverage the system to solve the data scarcity problem.

- Deployment of HEDGE on unoptimized code solutions (e.g., LeetCode, ProjectEuler)
- Collection of successful optimization trajectories: $(Code_{\text{inefficient}}, Code_{\text{efficient}})$ pairs
- Construction of the *Eco-Dataset* for supervised fine-tuning

5 Exploration and Future Work

5.1 Compiler-Level Integration ($\mathcal{L}_3$)

Future iterations will extend HEDGE to Layer 3 by interacting directly with Compiler Intermediate Representation (e.g., LLVM IR). The DGM can evolve optimal compiler flag combinations and pass orderings for specific workloads.

5.2 Multi-Objective Optimization

Future work will incorporate multi-objective optimization to generate Pareto frontiers balancing:

- Energy consumption
- Code readability
- Maintainability index
- Execution time

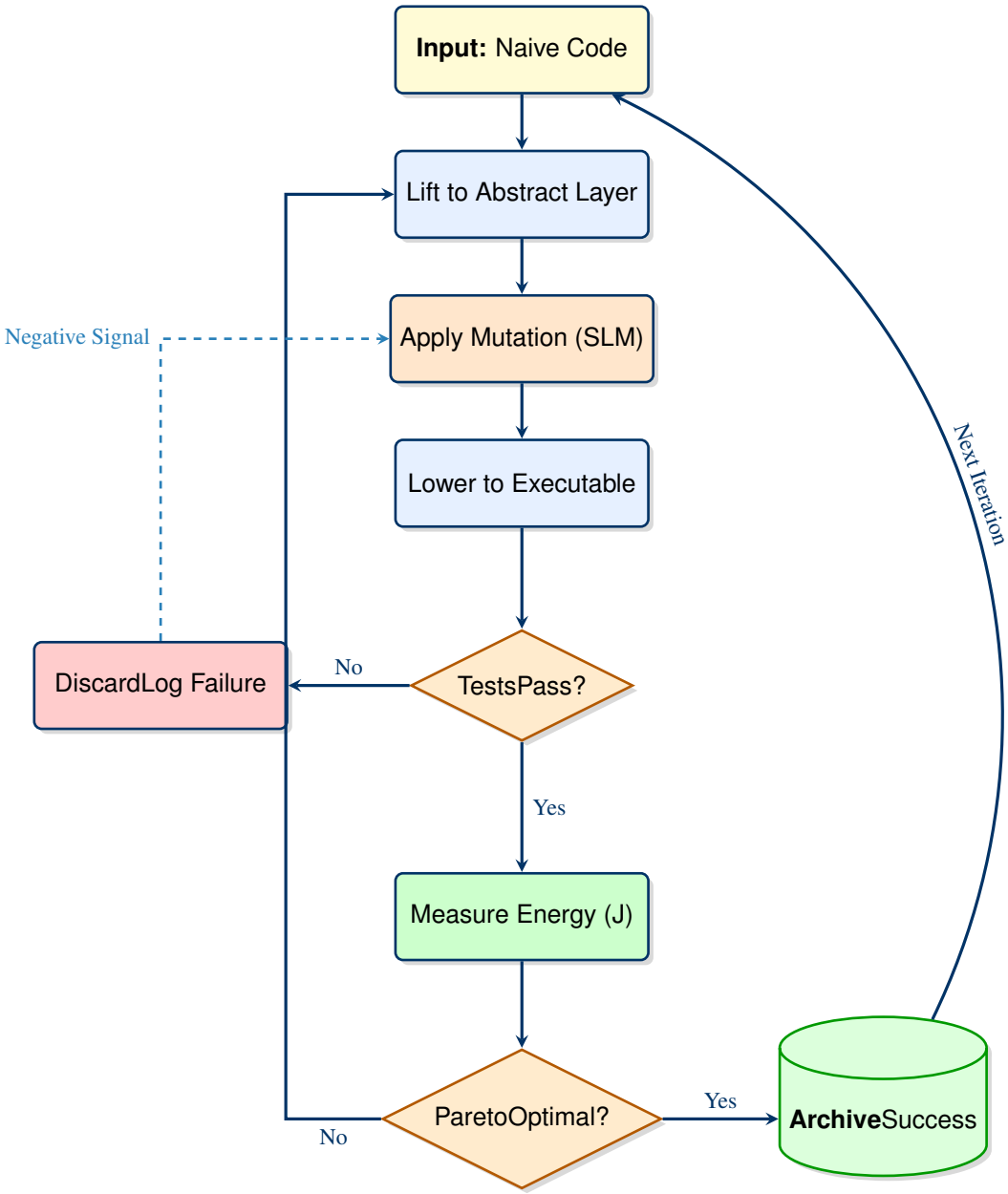
5.3 Formal Verification Integration

To strictly guarantee semantic equivalence beyond unit tests, we propose integrating formal verification tools (such as the K-Framework or Coq) to mathematically prove that a generated optimization is isomorphic to the original intent.

6 Conclusion

HEDGE represents a foundational step toward sustainable AI and software engineering. By moving beyond static optimization rules and leveraging the emergent capabilities of hierarchical evolution, HEDGE offers a path to systematically reduce the carbon impact of software at scale.

Furthermore, by autonomously generating its own training data through self-play, HEDGE resolves the critical bottleneck of data scarcity, paving the way for specialized Small Language Models that are native experts in Green Computing. This self-improving architecture positions HEDGE not merely as an optimization tool, but as a continuous learning system that can adapt to evolving hardware architectures and emerging efficiency paradigms.



Functional Correctness Gate: *All mutations must pass tests before they can be evaluated for efficiency.*

Efficiency Selection Pressure: *Only energetically superior variants progress in the evolutionary loop.*

Figure 2: The HEDGE Evolutionary Workflow. The cycle enforces strict functional correctness before energy measurement, creating robust selection pressure for efficiency improvements. Failed mutations contribute negative training signals.