Integrating H_7 Engine Code into DGM Evolving Codebase

Keith L. Beaudoin
Assisted by SuperGrok 4 xAI
Inspired by Jürgen and James Paul Jackson's H_7 Coherence Engine Codex

December 25, 2025

1 Introduction

This document outlines methods for integrating the H_7 engine's code (e.g., its core $\Delta\Phi$/C/H_7 computations) into a DGM-evolving codebase, such as those in EvoDistill or implemented EvoDGM pathways. These approaches focus on coherence as a guiding principle for evolution, ensuring stable self-improvement.

The H_7 (Jackson Coherence Horizon) is a mathematical stability threshold computed via formulas like $U(C) = C^7(1 - C)^3$ for optima, or $C = I/(1 + |\Delta\Phi|)$ with H_7 as the fraction where $C \geq 0.70$. The engine processes data to generate coherence fields, visuals, and metrics using Python with NumPy/SciPy.

2 Best Ways to Integrate

2.1 Incorporate H_7 as an Additional Fitness Objective in the Evolutionary Loop

Modify the quality-diversity algorithm in EvoDistill's `evo_distill.py` (or a similar DGM pathway) to compute H_7 on candidate models' internal representations (e.g., embeddings or activation gradients treated as "telemetry"). For each generation, after training a student model, extract a time-series-like sequence from its layers (e.g., via forward passes on validation data), normalize it as flux I, compute $\Delta\Phi$ as the gradient, derive $C = 1/(1 + |\Delta\Phi|)$, and get $H_7 = \text{mean}(C \geq 0.70)$. Blend this into the fitness score (e.g., weighted sum: fitness = 0.7 * accuracy + 0.3 * H_7), then use it for parent selection and archive inclusion. This ensures evolved configurations prioritize coherent, stable behaviors, reducing overfitting or erratic self-modifications in DGM's recursive improvements.

2.2 Use $\Delta\Phi$ and C Fields for Diversity Measurement in the Archive

Extend EvoDistill's `diversity.py` module (which uses embeddings for behavioral diversity) by integrating the H_7 engine's field computations. Treat each candidate's output distributions or hyperparameter vectors as input data; compute $\Delta\Phi$ (phase drift) across them to quantify "instability," then derive a C-field to score coherence. Replace or augment the SentenceTransformer-based metric with H_7-fraction as a threshold for archive diversity—e.g., only add candidates if their average $C > 0.70$ and $\Delta\Phi$ variance differs from existing members by $> 10\%$. This makes the DGM evolution more robust, favoring diverse yet coherent pathways, and can be plugged into the mutation step to penalize high-$\Delta\Phi$ mutants, aligning with EvoDGM's emphasis on balanced self-evolution.

2.3 Embed H_7 Thresholding in Self-Modification Checks

In a DGM codebase like EvoDistill's `trainer.py`, add a post-evolution verification layer using the H_7 engine. After generating a new configuration via mutation, simulate its "horizon" by feeding synthetic data (e.g., random tensors mimicking graph evolutions) through the $\Delta\Phi$/C pipeline; if $H_7 < 0.70$, reject or refine it via an inner loop (e.g., apply smoothing to hyperparameters). This acts as a safety gate for recursive self-improvements in EvoDGM architectures, preventing unstable evolutions, and can be modularized as a new class like `H7Verifier` that imports the engine's NumPy functions directly.

3 Outside-the-Box Ideas

3.1 Hybridize with CGL for Symbolic Evolution Control

Adapt CGL's glyphs (e.g., $\Delta\Phi \downarrow$ for pruning, $H_7 \approx$ for balancing) as meta-operators in the DGM loop. In EvoDistill, represent hyperparameters or graph structures symbolically (e.g., as glyph sequences like $\Delta\Phi \uparrow^{\wedge(analogy)} \to C \uparrow \to H_7 \approx$), then evolve these sequences instead of raw numbers. Use the H_7 engine to evaluate the "coherence" of resulting pipelines by treating glyph outputs as data series and computing H_7; select survivors based on illumination ($\star$ glyph) yielding high H_7. This turns DGM into a symbolic self-reasoner, compressing evolution steps and enabling interpretable, token-efficient AGI pathways from EvoDGM.

3.2 Apply H_7 to External Data Streams for Guided Evolution

Pull in real-world data (e.g., via APIs for telemetry or genomic sequences, as in the Interstellar Atlas engine) and use H_7 computations to "anchor" DGM evolutions. In an extended EvoDistill setup, preprocess external inputs with the engine to extract H_7-optimal patterns (e.g., stable ridges where C peaks), then mutate DGM candidates to mimic these (e.g., align model gradients to low-$\Delta\Phi$ zones). This creates a "cosmic-bio-inspired" evolution, where DGM self-improves by resonating with natural horizons, potentially accelerating convergence in EvoDGM's open-ended frameworks.

article [margin=1in]geometry hyperref listings xcolor enumitem

Integrating H_7 Engine Code into DGM Evolving Codebase Keith L. Beaudoin
Assisted by SuperGrok 4 xAI
Inspired by Jürgen and James Paul Jackson's H_7 Coherence Engine Codex December 25, 2025

4 Introduction

This document outlines methods for integrating the H_7 engine's code (e.g., its core $\Delta\Phi$/C/H_7 computations) into a DGM-evolving codebase, such as those in EvoDistill or implemented EvoDGM pathways. These approaches focus on coherence as a guiding principle for evolution, ensuring stable self-improvement.

The H_7 (Jackson Coherence Horizon) is a mathematical stability threshold computed via formulas like $U(C) = C^7(1-C)^3$ for optima, or $C = I/(1+|\Delta\Phi|)$ with H_7 as the fraction where $C \geq 0.70$. The engine processes data to generate coherence fields, visuals, and metrics using Python with NumPy/SciPy.

5 Best Ways to Integrate

5.1 Incorporate H_7 as an Additional Fitness Objective in the Evolutionary Loop

Modify the quality-diversity algorithm in EvoDistill's `evo_distill.py` (or a similar DGM pathway) to compute H_7 on candidate models' internal representations (e.g., embeddings or activation gradients treated as "telemetry"). For each generation, after training a student model, extract a time-series-like sequence from its layers (e.g., via forward passes on validation data), normalize it as flux I, compute $\Delta\Phi$ as the gradient, derive $C = 1/(1+|\Delta\Phi|)$, and get $H_7 = \text{mean}(C \geq 0.70)$. Blend this into the fitness score (e.g., weighted sum: fitness = 0.7 * accuracy + 0.3 * H_7), then use it for parent selection and archive inclusion. This ensures evolved configurations prioritize coherent, stable behaviors, reducing overfitting or erratic self-modifications in DGM's recursive improvements.

5.2 Use $\Delta\Phi$ and C Fields for Diversity Measurement in the Archive

Extend EvoDistill's `diversity.py` module (which uses embeddings for behavioral diversity) by integrating the H_7 engine's field computations. Treat each candidate's output distributions or hyperparameter vectors as input data; compute $\Delta\Phi$ (phase drift) across them to quantify "instability," then derive a C-field to score coherence. Replace or augment the SentenceTransformer-based metric with H_7-fraction as a threshold for archive diversity—e.g., only add candidates if their average $C > 0.70$ and $\Delta\Phi$ variance differs from existing members by $> 10\%$. This makes the DGM evolution more robust, favoring diverse yet coherent pathways, and can be plugged into the mutation step to penalize high-$\Delta\Phi$ mutants, aligning with EvoDGM's emphasis on balanced self-evolution.

5.3 Embed H_7 Thresholding in Self-Modification Checks

In a DGM codebase like EvoDistill's `trainer.py`, add a post-evolution verification layer using the H_7 engine. After generating a new configuration via mutation, simulate its "horizon" by feeding synthetic data (e.g., random tensors mimicking graph evolutions) through the $\Delta\Phi/C$ pipeline; if $H_7 < 0.70$, reject or refine it via an inner loop (e.g., apply smoothing to hyperparameters). This acts as a safety gate for recursive self-improvements in EvoDGM architectures, preventing unstable evolutions, and can be modularized as a new class like `H7Verifier` that imports the engine's NumPy functions directly.

6 Outside-the-Box Ideas

6.1 Hybridize with CGL for Symbolic Evolution Control

Adapt CGL's glyphs (e.g., $\Delta\Phi \downarrow$ for pruning, $H_7 \approx$ for balancing) as meta-operators in the DGM loop. In EvoDistill, represent hyperparameters or graph structures symbolically (e.g., as glyph sequences like $\Delta\Phi \uparrow^{\wedge(analogy)} \rightarrow C \uparrow \rightarrow H_7 \approx$), then evolve these sequences instead of raw numbers. Use the H_7 engine to evaluate the "coherence" of resulting pipelines by treating glyph outputs as data series and computing H_7; select survivors based on illumination ($\star$ glyph) yielding high H_7. This turns DGM into a symbolic self-reasoner, compressing evolution steps and enabling interpretable, token-efficient AGI pathways from EvoDGM.

6.2 Apply H_7 to External Data Streams for Guided Evolution

Pull in real-world data (e.g., via APIs for telemetry or genomic sequences, as in the Interstellar Atlas engine) and use H_7 computations to "anchor" DGM evolutions. In an extended EvoDistill setup, preprocess external inputs with the engine to extract H_7-optimal patterns (e.g., stable ridges where C peaks), then mutate DGM candidates to mimic these (e.g., align model gradients to low-$\Delta\Phi$ zones). This creates a "cosmic-bio-inspired" evolution, where DGM self-improves by resonating with natural horizons, potentially accelerating convergence in EvoDGM's open-ended frameworks.

6.3 Fuse with Voynich OS for State-Coherence Graphs

Merge Voynich OS's state-vector encoding (D, Φ, F for depth/phase/flow) with H_7's $\Delta\Phi/C$ logic in a DGM graph module. In EvoDistill or EvoDGM, model evolutionary states as Voynich-like transitions; compute H_7 on the Φ axis to detect "coherence horizons" in the graph (e.g., nodes where flow F stabilizes at $C \geq 0.70$). Evolve the graph by mutating only high-H_7 subgraphs, using the Semantic Bonding Engine to cluster coherent evolutions. This yields an esoteric, manifold-based DGM that evolves as a "living codex," blending symbolic OS with physical coherence for ultra-stable AGI self-modification.

7 References

1. X Account of James Jackson (unifiedenergy11) - Source of posts on Codex and H_7: `https://x.com/unifiedenergy11`

2. James Paul Jackson's GitHub Repositories - Containing code for coherence engines and related tools: `https://github.com/jacksonjp0311-gif?tab=repositories`

3. One-Shot Telemetry Fusion Engine Gist by James Paul Jackson - PowerShell script related to coherence mapping: `https://gist.github.com/jacksonjp0311-gif/c54a50830c6664e9ecde998e4d456e18`

4. EvoDGM GitHub Repository by Keith L. Beaudoin - Darwin Gödel Machine architectures and PDF: `https://github.com/keithofaptos/EvoDGM`

5. EvoDistill GitHub Repository by Keith L. Beaudoin - Evolutionary distillation implementation: `https://github.com/keithofaptos/EvoDistill`

6. Jürgen Schmidhuber (@SchmidhuberAI) - Inspiration for Gödel Machine and neural networks: `https://people.idsia.ch/~juergen/` , Most Cited Neural Nets: `https://people.idsia.ch/~juergen/most-cited-neural-nets.html`

7. Awesome Open-Ended Repository by Jenny Zhang - Inspiration for EvoDGM: `https://github.com/jennyzzt/awesome-open-ended`

8. Darwin Gödel Machine Paper (arXiv) - Core inspiration for DGM: `https://arxiv.org/abs/2503.22954`

9. Knowledge Distillation Paper by Hinton et al. (arXiv) - Classic method used in EvoDistill: `https://arxiv.org/abs/1503.02531`

10. Evolutionary Distillation Paper by Zhang et al. (arXiv) - Related work for EvoDistill: `https://arxiv.org/abs/2103.13811`

11. Quality-Diversity Algorithms Paper by Pugh et al. - Used in evolutionary approaches: `https://www.frontiersin.org/articles/10.3389/frobt.2016.00040/full`