Myelin: A Self-Evolving Agent Knowledge System with Enforcement Hooks for Production Claude Code Deployments

Mark Teekens
AIFAIS
mark@aifais.com

Preprint, April 2026

Abstract

We present **Myelin**, a self-evolving agent knowledge system that has operated autonomously on a single Hetzner VPS for six weeks at an infrastructure cost of four euros per month. Unlike recent agent protocols that formalise lifecycle and tool-invocation semantics at specification level (A2A, Model Context Protocol, Autogenesis [1]), Myelin is a deployed single-tenant system built from bash, cron, markdown files, git, and the Claude Code CLI. It contributes three architectural commitments absent from comparable "second brain" systems: **(i) enforcement hooks** that structurally block tool invocations violating learned operational rules; **(ii) a closed learn-loop** protected by a deterministic safety gate (evo-gate) that admits self-mutations only when they pass a fixed set of structural predicates; and **(iii) a three-layer evaluation stack** separating output existence, task-specific validation, and semantic quality. We report empirical results from an eighteen-day observation window (2026-04-01 to 2026-04-18; snapshot 12:54 UTC) covering 289 task executions across 56 distinct automated tasks with a completion rate of 94.1%, an evaluation pass rate of 89.55%, and 8 evo-gate rejections preventing structurally unsafe self-mutations. A simulated fault-injection experiment (clearly labelled; Section 8) characterises evo-gate behaviour under adversarial mutations. We also reproduce FRESCO-style [2] stale-retrieval bias on our 248-file corpus: a reranked query for "agent architecture" returned a 27-day-old overview at rank 1 (score 0.90), while the canonical freshly validated document was absent from the top three. Three independent systems (Hermes-Agent [4], Claude Code itself [3], Autogenesis) have converged on overlapping architectural primitives, which we treat as evidence that the minimum viable architecture for a self-evolving single-operator agent is smaller than the current literature implies. We release the engine at `github.com/AIFAIS/myelin`.

1 Introduction

Large Language Model (LLM) agents are increasingly deployed as always-on systems that schedule themselves, author prompts, invoke external tools, and mutate persistent knowledge stores. The research response has been to specify **protocols** for agent lifecycle, tool invocation, and inter-agent communication: Agent-to-Agent (A2A), the Model Context Protocol (MCP), and the recently proposed Autogenesis Protocol [1] with its two-layer Resource Substrate + Self Evolution decomposition. These protocols formalise what a compliant agent *should* look like but leave the operational question — *what does it take to run such an agent unattended in production?* — largely unanswered.

Myelin takes the inverse approach. Rather than prescribe a protocol and wait for implementations, we describe a deployed system that has been operating autonomously on a single Hetzner VPS for six weeks and extract the architectural commitments that made it work. The commitments are deliberately small. Every knowledge unit is a markdown file with typed YAML frontmatter. Every automated action is a bash-invoked Claude Code session with a prompt file of at most fifty lines. Every mutation is a git commit. There is no database, no vector store (retrieval is added by an external tool at read-time only), no agent framework, and no ontology beyond the folder structure. What Myelin adds to this minimal substrate is the subject of this paper.

1.1 Contributions

Production architecture. A three-layer design (Section 3) combining a filesystem-native knowledge base, a cron-scheduled automation pipeline, and an enforcement layer of Claude Code hooks that structurally block tool calls violating learned rules.

Formal model. Seven definitions (Section 4) giving typed semantics to knowledge items, domain saturation, enforcement predicates, evo-gate decisions, and verification loss.

Algorithms. Six algorithms (Sections 5–6) covering safe self-mutation, exponential confidence decay, saturation-gated ingest, verification loss computation, failure ranking, and path-drift self-healing.

Propositions. Three propositions with proof sketches (Section 7) establishing bounds on decay, coverage of the evo-gate, and convergence of the saturation admission policy.

Empirical results. Eighteen days of production logs (Section 8): 289 task executions, 94.1% completion, 89.55% eval pass rate, 8 self-mutations rejected by the gate.

Case studies. SEO impact across six client sites over five months (Section 9), including honestly reported negative results.

Simulation. Ten thousand synthetic fault injections against the evo-gate (Section 10, explicitly labelled simulation) measuring detection coverage and false-positive rate across five violation classes.

Honest failure analysis. Three open failure modes (Section 11) including an empirical reproduction of FRESCO-style stale retrieval bias on our own corpus.

1.2 Non-contributions and scope

We do not claim novelty in the LLM or retrieval components themselves: Myelin uses stock Claude Code, off-the-shelf reranking (QMD 2.0.1 with Qwen3-Reranker-0.6B), and well-known statistical functions. The contribution is architectural and empirical. We also do not claim generality beyond single-tenant operator-in-the-loop deployments; the design assumes one human who reviews and curates, not a team with conflicting edits. Multi-tenant extensions are deferred to future work.

2 Background and Related Work

2.1 Agent protocols

Autogenesis [1] proposes a Resource Substrate Protocol Layer (RSPL) and Self Evolution Protocol Layer (SEPL) for agent lifecycle management. Under Autogenesis, prompts, tools, environments, and memory are modelled as versioned protocol-registered resources; a closed-loop operator interface proposes, assesses, and commits improvements. Myelin can be read as a concrete single-tenant instance of the Autogenesis spirit built from pre-existing substrate: git provides versioning, markdown frontmatter provides resource typing, the learn-loop plus evo-gate provide the assess/commit cycle.

Model Context Protocol (MCP) standardises tool access for LLM agents. Myelin consumes MCP via an external retrieval tool (QMD) but does not expose its own services as MCP servers; its surface is the filesystem and Claude Code hooks.

2.2 Self-evolving agents

Hermes-Agent [4] (NousResearch, 2026) converged independently on a near-identical architecture to Myelin: a closed learning loop, persistent memory in the filesystem, cron-scheduled subagents, and deterministic gating over self-mutations. We view this convergence as evidence for the following hypothesis:

> The minimum viable architecture for a long-running self-evolving agent is a filesystem-native knowledge store, a cron-scheduled pipeline, and a deterministic safety gate over self-mutation. Frameworks, ontologies, and protocol specifications that go beyond this are usually optimisations — sometimes premature ones.
>
> — Convergence Hypothesis

2.3 Agent harness design

Recent analysis of Claude Code's internals [3] identifies a simple `while`-loop core surrounded by permission modes (seven modes plus an ML-based classifier), context compaction (five layers), extensibility (MCP / plugins / skills / hooks), and append-oriented session storage. Myelin's enforcement layer is the cognate of Claude Code's permission system, specialised for a single operator: instead of broad policy categories it encodes learned lessons as hook predicates.

2.4 Retrieval under temporal drift

FRESCO [2] shows that existing rerankers consistently prefer older, semantically rich documents over factually recent ones on evolving knowledge tasks, closing a 27-point performance gap via instruction optimisation. Section 11 reproduces this failure mode on our corpus.

2.5 Second-brain systems

Tools such as Obsidian, Logseq, and Roam provide markdown-based note storage with graph views. Myelin shares this substrate but differs in three ways. First, the enforcement layer: a "second brain" is read by a human who may ignore it; Myelin is executed by an agent whose tool calls are blocked by hook predicates when they contradict the knowledge base. Second, the closed learn-loop: notes in Obsidian do not rewrite themselves; Myelin prompts do. Third, the evaluation stack: Myelin tasks produce outputs that are graded by subsequent tasks, with failures propagating through fail-fast pipelines.

3 System Overview

```
          [ Layer 3: Enforcement Hooks ]
                  ↑ rules        ↓ blocks
          [ Layer 2: Automation Pipeline ]
                  ↑ commits      ↓ outputs
    [ Layer 1: Knowledge Base (markdown + git) ]
```

Listing 1: The three layers of Myelin. Data flows upward (pipeline consumes knowledge), control flows downward (enforcement hooks gate tool invocations based on stored rules).

Myelin is organised in three layers, each of which is independently useful and independently replaceable.

Layer 1 — Knowledge Base. A git repository of markdown files organised into seven functional folders: `patterns/` (reusable technical knowledge), `playbook/` (causal if-then chains), `mental-models/` (operator heuristics), `decisions/` (recorded choices with rationale), `clients/` (project context), `reference/` (external systems), and `claims/` (unvalidated hypotheses). A separate `data/` directory holds append-oriented logs and derived analyses. As of the observation window the corpus contains 111 canonical files: 54 playbook entries, 21 mental models, 13 patterns, 10 decisions, 5 claims, 4 clients, and 4 reference documents.

Layer 2 — Automation Pipeline. A version-controlled `crontab.txt` schedules approximately sixty tasks per day. Each task is a prompt file of at most fifty lines

with a single responsibility, invoked by `run-task.sh` via `timeout $T claude --dangerously-skip-permissions -p "$PROMPT"`. Pipelines chain tasks sequentially with checkpoint recovery. A daily 05:50 pre-flight runs eighteen test files; failure blocks all subsequent tasks.

Layer 3 — Enforcement. Claude Code's `PreToolUse`, `SessionStart`, and `UserPromptSubmit` hooks provide injection points for Python scripts that evaluate tool invocations against learned predicates. A `SessionStart` hook compiles a dynamic failure ranking (frequency × recency) from a persistent category file and injects it into every session. A `PreToolUse` hook blocks scraping of datacenter-blocked platforms before the HTTP request fires. A separate predicate protects the global `CLAUDE.md` from silent edits.

3.1 Communication layer

Seven agents post to two Discord channels (`#nikola` for system reports, `#chat` for casual group communication). Each agent has its own webhook, avatar, and name; `discord-send.sh` auto-selects the correct agent based on the task name. A real-time bot (`discord-bot/bot.ts`, `discord.js` via `bun`, systemd-managed) provides two-way interaction: Nikola in `#nikola` responds to any message from the operator; Monica in `#chat` responds when mentioned. Both use `claude --dangerously-skip-permissions --resume` for one-hour session persistence, and both can invoke the full tool surface (Playwright browsing, SMTP/IMAP email, filesystem access, bash).

Alerts carry three severity levels: **critical** (red, with an operator mention), **high** (red, no mention), and **medium** (yellow). Daily review and system audit pipeline failures are critical; weekly pipeline failures are high; prompt lint warnings are medium.

3.2 Pre-flight

A daily 05:50 pre-flight runs eighteen structural tests before any scheduled task fires. The tests include: prompt lint (line count, required sections), config drift (prompts reference only `config.sh`-listed repos), cron integrity (every prompt file the crontab references exists), data dependencies (expected input files exist), eval-trend (pass rates not declining), evo-gate validation, IQ health, learn-loop convergence, semantic eval integrity, and a script-validity walk. A failure after one retry sets a lockfile that blocks all subsequent tasks for the day. This invariant — the system does not run on broken infrastructure — is older than any of the learning components and has prevented more than one cascading failure in earlier phases of the project.

4 Formal Model

We now give typed semantics to the objects Myelin manipulates. All definitions refer to a fixed universe of files $\mathcal{F}$, domains $\mathcal{D}$, sources $\mathcal{S}$, and a time axis $t \in \mathbb{R}^+$.

Definition 1. Knowledge Item. A knowledge item is a tuple

$$k = (f, c, e, d, \tau, \sigma) \in \mathcal{K}$$

where $f \in \mathcal{F}$ is the file path, $c \in [0,1]$ the confidence, $e \in \mathbb{N}$ the evidence count, $d \in \mathcal{D}$ the domain, $\tau \in \mathbb{R}^+$ the `last_validated` timestamp, and $\sigma \in \{\text{true}, \text{false}\}$ the stale flag. The frontmatter of each markdown file serialises this tuple.

Definition 2. Domain Saturation. Let $K_d = \{k \in \mathcal{K} : d_k = d\}$ and S_d the set of distinct source citations across K_d. The domain saturation is

$$\sigma_d = |K_d| / \max(|S_d|, 1).$$

We call a domain *saturated* if $\sigma_d > 3.0$. New notes targeting a saturated domain are admitted only if they demonstrably improve or contradict an existing entry.

Definition 3. Confidence Decay Function. Given current time t, the decayed confidence of item k is

$$C(k,t) = \max\bigl(0.2,\, c_k \cdot \rho^{\lfloor (t - \tau_k)/7\text{d} \rfloor}\bigr), \quad \rho = 0.95.$$

The floor at 0.2 prevents complete forgetting; the weekly exponential avoids the cliff effects of threshold-based decay.

Definition 4. Enforcement Predicate. A predicate is a function

$$\varphi : (\text{tool}, \text{args}, \text{ctx}) \to \{\text{block}, \text{warn}, \text{allow}\}.$$

The enforcement layer is a finite set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ evaluated sequentially; the first non-`allow` result determines the outcome.

Definition 5. Evo-Gate. The evo-gate is a function

$$G : \mathcal{D}\text{iff} \to \{\text{accept}, \text{revert}\}$$

mapping prompt-file diffs to admission decisions. G is defined by a conjunction of structural predicates $g_1, \ldots, g_m$ (line count, protected files, required sections, path validity, preservation of output formats). $G(\mathrm{D}) = \text{accept}$ iff $\forall i : g_{i(\mathrm{D})} = \text{pass}$. G itself is protected from self-modification.

Definition 6. Verification Loss. Let $V(t)$ be the set of automated claims verified by the operator or by subsequent tasks on day t, and $T(t)$ the set of total claims. The verification loss is

$$\mathcal{L}_{v(t)} = 1 - |V(t)|/|T(t)|.$$

Operational thresholds: $\mathcal{L}_v < 0.15$ is normal, $0.15 \leq \mathcal{L}_v < 0.30$ warrants attention, $\mathcal{L}_v \geq 0.30$ is urgent.

Definition 7. Failure Ranking. Let failures in category c occur at times $t_1, ..., t_n$ and let t be the current time. The failure score is

$$r_{c(t)} = \sum_{i=1}^{n} \exp(-\lambda(t - t_i)), \quad \lambda = \ln(2)/14 \text{ d},$$

a frequency-weighted recency metric with a fourteen-day half-life. Top-k categories by r_c are injected into every new session.

Definition 8. Chat Posting Probability. When an agent considers posting in the group channel, the probability is

$$P(\text{post}) = p_0 \cdot \gamma^k, \quad \gamma = 0.85,$$

where k is the number of consecutive agent-only messages since the last human contribution. This converts a hard silence threshold into a gradual decay, preventing reset-exploits.

5 Knowledge-Layer Algorithms

5.1 Exponential decay

Confidence decay (Definition 3) is applied daily by a maintenance task that scans all `.md` files, reads `last_validated`, computes the new confidence, and rewrites the frontmatter. A git hook resets `last_validated` to the commit timestamp when a human-authored edit touches the file (auto-commits by the agent do not reset). This ensures that the decay reflects *operator attention*, not mechanical re-saving.

Algorithm 1 — Confidence Decay

```
input : corpus K, current day t
output: updated K with decayed confidences
for each k in K:
  if manual commit in last 7d for k:
    tau_k ← commit timestamp
  w ← floor((t - tau_k) / 7 days)
  c_new ← max(0.2, c_k · 0.95^w)
  if t - tau_k >  90d: sigma_k ← true
  if t - tau_k > 180d: mark archive_candidate
  write frontmatter(k, c_new)
```

5.2 Saturation-gated ingest

The `/ingest` skill implements Definition 2 by computing σ_d per domain before admitting a candidate note. Saturated domains ($\sigma_d > 3$) require the candidate to demonstrably improve or contradict an existing entry; otherwise the candidate is discarded as redundant. This guards against an *echo chamber* failure mode where a growing corpus filters input more aggressively than it learns from it.

Algorithm 2 — Saturation-Gated Ingest

```
input : candidate note n, target domain d
output: admit | reject | update existing
K_d ← { k in K : domain(k) = d }
S_d ← { source(k) : k in K_d, distinct }
sigma_d ← |K_d| / max(|S_d|, 1)
if sigma_d <= 3.0:
  admit(n)
else:
  for each k in K_d:
    if overlaps(n, k):
      if improves(n, k) or contradicts(n, k):
        update(k, n)
        return
  reject(n, reason="saturated")
```

The `improves` and `contradicts` predicates are currently evaluated by the operator during `/ingest`; we deliberately do not automate this step because false negatives (rejecting a genuinely new insight) are much costlier than false positives (admitting a slightly redundant note).

5.3 Failure-rank injection

`Algorithm 3` populates the `SessionStart` context with the top-k failure categories by r_c. Two empirical properties matter. First, the fourteen-day half-life reflects the observation that a failure stops being predictive after a month; longer windows silted the session prompt with resolved issues. Second, re-ranking happens only at session start, not after every failure; intra-session ranks would be noisy.

Algorithm 3 — Failure-Rank Injection

```
input : failure log F, current time t, k ← 5
output: top-k failure messages for session injection
for each category c in F:
  times ← { t_i : failure t_i in c }
  r_c ← sum(exp(-λ(t - t_i)) for t_i in times)
sort categories by r_c descending
return top-k as context injection
```

5.4 Chat silence decay

Myelin deploys seven agents in a group Discord channel. Unmoderated agent chat converges quickly into a feedback loop in which agents respond to each other with diminishing information content. A hard silence threshold ("at most N messages without a human reply") has two failure modes: agents time their posts to just-miss the threshold, and transitions across the boundary are perceived as abrupt. We instead implement a stochastic admission rule matching Definition 8.

Algorithm 4 — Chat Admission (per agent, per tick)

```
input : messages since last human M_h, base prob p_0
= 0.25,
        decay gamma = 0.85
output: post | stay silent
k ← |M_h|
p ← p_0 · gamma^k
u ← uniform(0, 1)
if u < p: post
else:     stay silent
```

At $k = 0$ (just after a human message) the posting probability is $p_0 = 0.25$; at $k = 10$ it is $p_0 \cdot \gamma^{10} \approx 0.049$; at $k = 20$ it is approximately 0.010. Agents thus become progressively quieter without ever being fully silenced, and the system never has a "reset" boundary that can be gamed.

5.5 Inbox deduplication

Discord-triaged action items flow into `brain/inbox/`. The triage prompt is called multiple times per day; naïve appending produces duplicate items. We deduplicate against a rolling window of the previous seven days using a cheap token-overlap measure.

Algorithm 5 — Inbox Deduplication

```
input : new item n, history H (last 7 days)
output: admit | skip
t_n ← tokens(n.subject + n.body)
for each h in H:
  overlap ← |t_n ∩ tokens(h)| / |t_n ∪ tokens(h)|
  if overlap > 0.7: return skip
write n to inbox/
return admit
```

6 Pipeline-Layer Algorithms

6.1 Evo-gate

The evo-gate is the single most important component of Myelin's self-evolution: it is the line between the learn-loop modifying itself and the learn-loop destroying itself. The gate validates *structure* only — line counts, required sections, path references, protected-file invariants — which is deliberately weaker than a quality assessment. Section 11.2 discusses this limitation. Over the observation window the gate accepted 27 of 35 proposed mutations and reverted 8.

Algorithm 6 — Evo-Gate

```
input : diff D over server/prompts/
output: accept | revert
failures ← [ ]
for each file f in D:
  if f in PROTECTED:       failures += "protected"
  if lines(f) > 50:        failures += "lines"
  if missing(f, SECTIONS): failures += "sections"
  if missing(f, DATEVAR):  failures += "datevar"
  if stale_ref(f, REPOS):  failures += "path"
append (now, pass?, failures) to evo-gate-log
if failures = [ ]:
  git tag "evo-YYYYMMDD-HHMM"
  git commit --allow-empty -m "evo: " + diff_summary
  return accept
git checkout -- server/prompts/   # data/ kept
return revert
```

6.2 Path-drift self-healing

A recurring failure mode is *eval-path drift*: a learn-step renames a prompt's output file, the eval checker still looks for the old name, and the task reports `FAIL` despite producing correct output. This is a schema-mismatch between producer (mutated prompt) and consumer (unchanged eval). Algorithm 5 detects and heals this automatically.

Algorithm 7 — Path-Drift Self-Heal

```
input : recent eval failures E, current prompts P
output: candidate prompt edits
for each failure e in E where exit_code = 0 and eval
= FAIL:
  expected ← extract_path(e.eval_config)
  actual   ← find_output_since(e.timestamp)
  if expected ≠ actual and similar(expected, actual)
> 0.8:
    propose_edit(P[e.task], rewrite output path)
submit proposals to evo-gate
```

On 2026-04-07 six tasks failed with `file_exists` checks after an unrelated learn-02 mutation changed output naming; on 2026-04-08 five of the six healed automatically through this mechanism (83% recovery). The sixth case (`tenderned-scout`) involved a fundamentally wrong eval path and required manual intervention — a real limitation rather than a soft failure.

6.3 Fail-fast pipelines

A separate invariant prevents cascading failures. If any step in a pipeline returns a non-zero exit code, `run-pipeline.sh` halts and subsequent steps do not execute. This sacrifices throughput for correctness — we would rather miss a day's digest than build one on bad data.

6.4 Three-layer evaluation stack

Evaluation happens in three successive layers with different costs and different false-positive profiles.

Layer 1 — Structural eval. `eval-check.sh` runs after every task and performs cheap deterministic checks: did the expected output file appear, is it valid JSON, is it non-empty. Extracted from the prompt file directly (`## Output` sections name the expected paths). Cost: milliseconds. False-positive risk: schema mismatch between prompt and eval. This is the layer most vulnerable to path drift (Algorithm 7).

Layer 2 — Task-specific eval. For high-value tasks (`site-health` must list all eight monitored sites; `reflection` must have a `grounding` score and a `tomorrow_actions` field; `gsc-analyze` must cite meta tags) we add field-level validation. These checks are hand-written per task. Cost: milliseconds. False-positive risk: a schema change that is genuinely correct but does not match the validator yet.

Layer 3 — Semantic eval. `semantic-eval.py` runs on tasks with declared semantic tests. Pure Python, no LLM call: content length thresholds, required keyword presence, structural-format regexes. Cost: sub-second. Any LLM-judge-based evaluation is explicitly kept in observer mode (see `observer-mode-llm-judge.md`) and not wired into blocking decisions.

The decision to keep LLM judges out of the blocking path is a deliberate architectural choice: an LLM judge that blocks is a source of noise that can cascade into a stuck pipeline. Observers that log without blocking let us correlate judge outputs against downstream reality before promoting them.

6.5 Session-start briefing

Algorithm 8 — Session-Start Briefing

```
input : failure log F, memory M, day context D_y
output: BRIEFING.md injected into every session
R ← rank_failures(F, k=5, lambda=ln2/14d)  # Alg 3
L ← read(MEMORY.md)               # behavioural rules
P ← read(mark-denkpatronen.md)    # operator model
D ← read(data/day-context-yday.md)
write BRIEFING.md: stats + L + R + P + D
export CLAUDE_CONTEXT ← BRIEFING.md
```

This algorithm is the entry point for all the others. It composes static operator knowledge (rules, model of the

operator's preferences), dynamic ranked failures, and the previous day's observable context into a single briefing file that Claude Code reads at the start of every session. It is why an agent launched on the VPS at 06:00 knows the same things the operator's local Claude Code session knows at 14:00.

7 Propositions

Proposition 1. Decay Bound. For any knowledge item k with initial confidence c_0, and any time $t \geq \tau_k$,

$$C(k,t) \in [0.2, c_0].$$

Proof sketch. $C(k,t) = \max(0.2, c_0 \cdot 0.95^w)$ with $w \geq 0$. Since $0 < 0.95 < 1$, the term $c_0 \cdot 0.95^w$ is non-increasing in w and bounded above by c_0 (at $w = 0$). The `max` with 0.2 bounds the output from below. Therefore $C(k,t) \in [0.2, c_0]$ for all $t \geq \tau_k$. □

Proposition 2. Evo-Gate Structural Coverage. Let $\Phi_G = \{g_1, \ldots, g_m\}$ be the structural predicates of the evo-gate and $\mathcal{V}_S$ the set of diffs violating at least one predicate in Φ_G. Then for all $D \in \mathcal{V}_S$, $G(D) =$ revert.

Proof sketch. G is defined as a conjunction: $G(D) =$ accept iff $\forall i : g_{i(D)} =$ pass. If $D \in \mathcal{V}_S$ there exists i with $g_{i(D)} =$ fail, so the conjunction is false and $G(D) =$ revert. □

Remark. Proposition 2 gives *structural* coverage. It says nothing about *semantic* coverage: a diff that changes `threshold ≥ 6` to `threshold ≥ 5` violates no structural predicate but may degrade output quality. We discuss this gap — which is the open problem of quality regression detection in self-evolving agents — in Section 11.2.

Proposition 3. Saturation Admission Convergence. Assume new sources arrive IID across domains with uniform probability $1/|\mathcal{D}|$, and the saturation policy admits candidates iff $\sigma_d \leq 3$ or an `improves`/`contradicts` condition is met. Then in expectation, as the corpus size $N = |\mathcal{K}|$ grows, the ratio

$$\max_d \sigma_d / \min_d \sigma_d \to 1.$$

Proof sketch. Let $N_d^{(t)}$ be the number of notes in domain d at time t. Under IID arrival with uniform domain probability, $E\left[N_d^{(t)}\right] = N^{(t)}/|\mathcal{D}|$. The admission policy blocks growth in N_d once $\sigma_d > 3$ while unrestricted domains continue to grow. The hard cap flattens the distribution, and in the limit $t \to \infty$ the admission ratio across domains tends to the minimum of the cap and the arrival rate, equalising σ_d across d. □

Remark. The "IID with uniform domain probability" assumption is unrealistic in practice — real ingest streams are bursty and domain- biased by what the operator reads on a given week. Proposition 3 is therefore a *convergence in expectation* result, useful as a sanity check rather than a guarantee. Empirically (Section 8) we observe domains with σ_d in the range 1.0–4.2 over the observation window.

8 Empirical Evaluation

8.1 Setup

All results in this section are extracted from Myelin's production logs over the eighteen-day observation window 2026-04-01 to 2026-04-18. Three log files are the source of truth: `data/evo-log.jsonl` (one entry per task execution), `data/eval-log.jsonl` (one entry per eval check), and `data/evo-gate-log.jsonl` (one entry per self-mutation attempt). Log files are append-only and under version control; the full logs are available in the public repository.

8.2 Main results

Metric	Value
Observation window	18 days
Task executions	289
Unique task types	55
Completion rate (exit 0)	94.1%
Timeouts (exit 124)	10 (3.5%)
Eval failures (exit 2)	5 (1.8%)
Runtime errors (exit 1)	1 (0.4%)
Eval pass rate	89.55% (240/268)
Evo-gate pass rate	77.14% (27/35)
Mutations blocked	8
Total compute	12.9 h
Mean compute / day	43.0 min
Git commits	527 (29.3/day)
Evo-tags created	21
Knowledge files curated	111

Table 1: Production metrics over 18 days of unsupervised operation. All numbers are extracted directly from Myelin's append-only JSONL logs; no synthetic data is included in this table.

8.3 Task-family distribution

Of the 289 executions, the dominant families are the `system-audit` pipeline (48 executions across three stages), the `daily-review` pipeline (47 executions), and the `learn-loop` (34 executions). The long tail includes scouts (freelance, consultant, tenderned, hoofdkraan, outbound), research tasks (GitHub Trending, HN frontpage, Twitter), and maintenance jobs (inbox processing, evidence decay). Fifty-five distinct task types cover the full operational surface.

Family	Runs
system-audit-*	48
daily-review-*	47
learn-loop-*	34
gsc-*	19
maintenance-*	13
site-health	11
weekly-*	11
github-trending	10
hn-frontpage	10
discord-*	9

Table 2: Top ten task families by execution count over the observation window. Extracted from `evo-log.jsonl`.

The scheduling asymmetry is deliberate. `system-audit` and `daily-review` are three-step pipelines that run once per day; each day therefore contributes three executions per family. Scouts run on selective weekdays (e.g. `tenderned-scout` ran 6 times, reflecting its Tuesday/Thursday/Saturday cadence).

8.4 Daily throughput

Date	Runs	Date	Runs	Date	Runs
04-01	11	04-07	15	04-13	16
04-02	11	04-08	20	04-14	21
04-03	21	04-09	19	04-15	6
04-04	9	04-10	33	04-16	14
04-05	21	04-11	16	04-17	28
04-06	5	04-12	16	04-18	2

Table 3: Daily task executions across the observation window. Weekend dips (2026-04-04, 2026-04-06, 2026-04-15) reflect reduced scheduling for non-critical jobs. 2026-04-18 is partial (data cut at noon).

Median daily throughput is 16 runs; peak was 33 (2026-04-10, coinciding with the weekly GSC pipeline on Friday evening).

8.5 Self-mutation analysis

The learn-loop proposed 35 prompt mutations over the window. The evo-gate accepted 27 (77.1%) and reverted 8. Rejection reasons, taken verbatim from `evo-gate-log.jsonl`, cluster into two causes: *line-count violations* (mutations that accreted content beyond the 50-line ceiling; 5 cases across `outbound-scout`, `tenderned-scout`, `freelance-scout`, `consultant-nl-scout`, `seo-scanner`, `review-02-act`), and *path-reference drift* (2 cases on `learn-01-observe` and `review-01-collect` where renamed directories broke references). No rejected mutation caused a cascading failure in the observation window because `evo-gate.sh` reverts prompt changes while keeping data/ changes, allowing the data side of a run to be recorded even when the prompt side is rolled back.

8.6 Domain saturation snapshot

Applying Definition 2 to the corpus at `2026-04-18` yields the following per-domain saturation. Domains with $\sigma_d > 3.0$ are flagged; `/ingest` routes to those domains require a demonstrable improvement over existing entries.

Domain	Notes	Sources	σ_d
automation	31	14	2.21
seo	12	6	2.00
ai-engineering	8	5	1.60
vakkennis	9	6	1.50
meta	13	3	4.33
workflow	4	3	1.33
business	6	4	1.50
leads	3	2	1.50
data	10	8	1.25

Table 4: Per-domain saturation across the canonical corpus. Values > 3.0 indicate saturated domains (higher admission bar applies). `meta` is the one domain currently at ceiling.

The `meta` domain (containing notes about Myelin itself and operator-behavioural rules) is saturated because the operator is also the primary source for those notes; this is an expected fixed point. The automation and seo domains are close to the threshold and are the most interesting for future ingest calibration.

8.7 Reliability

Of 289 executions, 272 returned exit code 0 (94.1%). The remaining 5.6% decompose as: 10 timeouts (scouts running longer than their timeout budget under slow network), 5 eval failures (schema mismatches between producer and consumer that the path-drift heal fixed within the next day), and one runtime error. No failure in the window propagated to downstream pipeline steps because of the fail-fast invariant.

9 Case Studies

Myelin is primarily deployed in support of **AIFAIS**, a one-person consultancy managing six client websites alongside the agency's own site. The operator's time constraint is the hard motivation for the system. This section reports impact across five months of SEO and analytics data. All numbers are extracted from Google Search Console exports and Vercel Analytics; raw exports are archived in the `data/raw/` directory of the Myelin repository.

9.1 Case 1: AIFAIS repositioning (January 2026)

On 2026-01-13 a commit — named in the log as "reposition as Claude-powered AI Automation Micro-Agency" — changed the agency's primary positioning. The effect was visible within three weeks: daily impressions rose from approximately 40 to 200 (+400%), and a new query cluster emerged around "ai automatisering", "contract checker", and "factuur maken". Myelin's post-commit monitoring pipeline surfaced the causal pattern five days later. The pattern now lives in the playbook as *po-*

sitionering-indexatie.md with confidence = 0.9 and five supporting evidence entries.

9.2 Case 2: SEO push and GEO content (February 2026)

A coordinated set of commits on 2026-02-17 added pillar pages, `FAQPage` and `AggregateRating` structured data, cross-links, and national targeting. The effect over the next four weeks: daily impressions rose from approximately 150 to 450 ($\times$ 3). A subsequent commit on 2026-02-18 introduced GEO-pakket content and `llms.txt`; this produced an entirely new query cluster around "geo seo agency" contributing 183 impressions.

9.3 Case 3: Sitemap compression across five sites

The playbook entry *sitemap-crawlbudget-focus.md* codifies a compression pattern validated across five sites: where the domain authority is lower than the sitemap size implies, shrinking the sitemap to known-performing pages focuses crawl budget. Outcomes are summarised below.

Site	Before	After	Reduction
AIFAIS	1098	186	−83%
MyWestApp	4300	100	−98%
TTMCards	5900	675	−89%
Schadeassistentie	120	85	−29%
Curit	75	55	−27%

Table 5: Sitemap compression across five sites, April 2026. All numbers from Google Search Console; the pattern lives in `playbook/sitemap-crawlbudget-focus.md`.

MyWestApp's extreme case (4300 to 100 URLs) reflects a city×sector template that generated 2856 URL combinations with approximately zero clicks. The reduction was accompanied by a clicks-per-impression improvement on mobile (CTR 2.67% mobile versus 0.21% desktop) as remaining URLs surfaced for their target queries.

9.4 Case 4: Honest negative result

On 2026-03-24 Myelin generated six SEO fix proposals (meta descriptions, title edits, positional improvements) which were implemented and merged. Day-10 monitoring showed five of six on positive trajectories. Day-17 monitoring revealed the reversal: **zero fixes improved, four had no effect, one got worse, one had insufficient data to judge**. The most stark case was `curit.nl /kennisbank/blijvende-invaliditeit`: click-through rate fell from 7.09% to 1.1% (a 84% degradation) at a stable position.

We report this negative result because the pattern it surfaces is, for a self-evolving system, more interesting than the wins. Myelin now carries the following corrective note in its playbook:

> "The current approach of meta-fixes and content-optimisation delivers insufficient measurable result. Larger interventions (content depth, backlinks, structured data) should be considered alongside title and description tweaks."

The pattern feeds back into the ingest layer: future proposals of "cheap meta fixes" are now ranked below proposals with deeper interventions.

9.5 Case 5: Weekly impact rollup

Site	Clicks	Imp.	CTR	Pos.
aifais.com	23	1610	1.43%	25.4
schadeassistentie.nl	22	1123	1.96%	24.5
curit.nl	15	4748	0.32%	34.7
mywestapp.com	3	456	0.66%	26.2
vakscout.nl	0	266	0.00%	55.2

Table 6: Week-over-week Google Search Console metrics across all five AIFAIS sites, week of 2026-04-10. Impressions are a 7-day sum; position is the weighted average over queries with ≥1 impression. Source: GSC export imported into `data/raw/`.

The spread across sites reflects their lifecycle stage: aifais.com (established, undergoing repositioning), schadeassistentie.nl (established, recent restructuring), curit.nl (established, narrow niche with high impression count per click), mywestapp.com (sitemap recently pruned from 4300 to 100 URLs with expected impression dip), and vakscout.nl (four weeks old, building authority). Myelin's weekly pipeline produces this table automatically from GSC exports and posts it to Discord every Friday at 18:30.

9.6 Case 6: AI-referral channel emergence

Over Q1 2026, Myelin's analytics pipeline detected and tracked a new referral channel: large language model chat interfaces. Traffic attribution over three months shows 78 visitors (16% of external referrals) originating from ChatGPT (70), Perplexity (4), Claude (2), Copilot (1), and Gemini (1). The pattern is documented as *ai-citatie-content* and drives a specific content-optimisation track (llms.txt, structured FAQ content, citable claims).

10 Simulation Study

This section reports **simulated** results. Unlike Sections 8 and 9 which extract figures from production logs, the numbers here come from fault injection against Myelin's code running on synthetic inputs. We describe the methodology and clearly label results to prevent confusion with production metrics.

10.1 Evo-gate fault injection

We generated $N = 10000$ synthetic prompt diffs under five violation classes and ran each through `evo-gate.sh` to measure detection coverage and rejection latency.

Class	Perturbation	N
`lines`	append 5 lines past 50	2500
`protected`	touch `reflection.md`	1500
`path`	rename referenced dir	2000
`sections`	delete `## Output` header	2000
`datevar`	remove `$(date +%F)`	2000

Table 7: Simulated violation classes, synthetic generation rule. These are not production mutations; they exist only to probe gate behaviour.

Since the evo-gate is structural and deterministic, 100% recall on the injected classes is expected by construction (Proposition 2). The more informative measurements are:

False-positive rate on valid diffs. We additionally injected 2000 structurally valid diffs (line-count changes *within* the 50-line ceiling, section re-ordering that preserves required headers, path references to repos in `config.sh`). The gate rejected 0 of 2000; the observed false-positive rate is 0.00% (95% Wilson interval $[0.00\%, 0.18\%]$).

Gate latency. Over the 12000 total diff runs, median wall-clock evaluation time was 73 ms (interquartile range 48–110 ms). The gate does not meaningfully contribute to pipeline runtime.

Cascading violation handling. A diff violating multiple predicates (e.g. both `lines` and `path`) is recorded with all violations in `evo-gate-log.jsonl`. Inspection of 500 cascading cases confirmed that all violations were logged, enabling diagnosis without repeated runs.

10.2 Retrieval-bias ablation

We selected 10 conceptual queries across agent architecture, context management, SEO, and brain self-reference, issued them against the production QMD index (248 files, BM25 + Qwen3-Reranker-0.6B), and recorded the rank of the canonical `how-myelin-works.md` document (`last_validated` within one week of the query). Results:

Query	Canonical in top-3
agent architecture	no
how does Myelin work	no (0 results)
knowledge base design	yes
self-evolving agent	yes
context management	no
enforcement hooks	yes
learn loop design	yes
prompt mutation safety	no
fail-fast pipelines	yes
three-layer architecture	yes

Table 8: Canonical-document recall at $k = 3$ for ten conceptual queries against the production index. `how-myelin-works.md` is the designated canonical target; other top-3 results in failure cases were 17–27 days older.

Recall@3 for the canonical document was 6/10 = 60%. Four failure cases exhibited the FRESCO pattern: the rank-1 result was a March overview or a daily context log matching on vocabulary but not intent. We emphasise that this simulation runs against *the real index*; it is "simulated" in the sense that we constructed the query set and canonical-target assignment, not in the sense that we generated synthetic documents.

11 Failure Modes

A system that reports only its wins is not a production system. We describe three failure modes that remain open at submission time.

11.1 Stale retrieval bias

As shown in Section 10.2 and reported in the project playbook as *retrieval-stale-bias.md* (created 2026-04-18), our hybrid retrieval layer reproduces FRESCO [2]'s finding on our own corpus. For the query "agent architecture", the top reranked result was a March overview document (`patterns/kennisbronnen.md`, `last_validated` 2026-03-22) scoring 0.90; the canonical fresh target (`playbook/how-myelin-works.md`, `last_validated` 2026-04-12, `evidence` = 9, `confidence` = 0.95) did not appear in the top three. The decay information lives in Myelin's frontmatter but is not read by the reranker.

We have not yet implemented a mitigation. The playbook lists four candidate interventions ordered by cost: query prepending with a `last_validated` filter, reranker prompt tuning with a recency instruction, post-retrieval filtering on the `stale` flag, and periodic retrieval-eval to catch regressions. None has been trialled at the time of writing.

11.2 Evo-gate quality gap

The evo-gate enforces structural invariants (Proposition 2) but not semantic ones. A mutation changing `score >= 6` to `score >= 5` in a scout prompt is structurally valid and accepted; whether it yields better or worse leads is only visible after the next production run. Reflection cross-validates evo-tags against downstream failures with a one-

day lag (Section 6.2), but subtle regressions can persist longer. A genuine quality-regression test suite, probably built on held-out "golden" inputs per prompt, remains future work.

11.3 Daily-log retrieval pollution

Daily context logs (`data/day-context-*.md`) and research scans (`data/research-*.md`) share terminology with canonical patterns and outrank them in BM25 queries. In our corpus `day-context-2026-04-09.md` ranked first for "context management" at score 0.86, displacing `playbook/context-pollution.md`. The mitigation documented in the playbook is to split the QMD collection into `brain-patterns` (patterns + playbook + mental-models) and `brain-journal` (data + inbox), so that default retrieval draws only from canonical knowledge; journal retrieval becomes an explicit opt-in.

11.4 Retrieval mitigations, ordered by cost

The playbook entry for stale bias lists four candidate mitigations ordered by their implementation and operational costs.

Mitigation	Scope	Cost
`last_validated` prefix in query	per call	low
Filter `stale: true` in post-retrieval	per call	low
Collection split (`patterns` vs `journal`)	global	medium
Reranker prompt-tune (FRESCO-style)	global	high

Table 9: Retrieval bias mitigations under consideration. None has been deployed yet; this is future work.

We have not yet decided which to trial first. The low-cost options are attractive but do not address the mechanism (reranker does not read frontmatter); the global options address the mechanism but require changes to the retrieval substrate.

11.5 Evidence for the convergence hypothesis

Beyond the three systems cited in Section 2.2, further evidence for the convergence hypothesis has accumulated during the paper's drafting. The filesystem-native pattern has appeared independently in several open-source agent projects in 2026; the evo-gate pattern (a deterministic structural gate over self-mutations) is present in Hermes-Agent's release notes and in the Claude Code analysis paper's description of permission modes.

If the hypothesis is true, the implication for the agent-protocol research agenda is uncomfortable: the substrate (git + cron + markdown + shell) may already be sufficient; the protocol layer may be largely notation. We do not have the data to validate this strongly, and we acknowledge the self-selection risk (we chose to build with primitives; we are likely to see those primitives succeed). The claim is a hypothesis, not a conclusion.

12 Discussion

Three design commitments matter more than any single algorithm in Myelin.

Filesystem-native storage. Making git the source of truth turns every mutation into an auditable event, removes an entire class of database failures, and makes the system trivially self-hosting. A corollary: any tool that requires its own storage layer should be considered suspect until proven irreplaceable. Myelin uses exactly one external storage service (Supabase, for a client CRM) and one external retrieval tool (QMD, read-only on top of the filesystem).

Enforcement has teeth. A rule recorded in a wiki that the agent *should* follow is not a rule. The step from "we wrote down that datacenter IPs are blocked on LinkedIn" to "the tool call is structurally prevented by a Python hook that reads `config.sh`" is what converts passive documentation into active policy. Six weeks of production operation suggest the translation is worthwhile: the hook has fired and blocked actions several times, each avoiding a failure mode that earlier manual-rule-only deployments hit.

Boring structural gates over clever semantic ones. The evo-gate is a conjunction of about five predicates written in bash. It is rejected by everyone who encounters it on aesthetic grounds. It is also the only reason the learn-loop has not destroyed the prompt filesystem in six weeks. Semantic quality gates using LLM judges are an active research area [5]; in our experience they are too noisy to run in a blocking position. We keep them in observer mode and correlate with downstream outcomes before promoting.

The three design commitments together form a single thesis: the minimum viable architecture for a self-evolving agent is much smaller than the current agent-protocol literature suggests. The convergence with Hermes-Agent [4] and the Claude Code analysis [3] supports this. Autogenesis [1] is in our reading a formalisation of patterns that have already emerged in deployed systems.

13 Threats to Validity

13.1 Internal validity

Log integrity. All production metrics are extracted from append-only JSONL logs under version control. The logs are written by the same task runner whose behaviour they describe. A compromise of `run-task.sh` would compromise the logs; we have not audited for this possibility beyond reviewing diffs. A reader can reproduce the numbers in Section 8 by cloning the repository and running `paper/metrics.py`, which reads logs that are committed alongside this paper.

Observer effect. The operator is the author of this paper. During the observation window, normal curation activity continued: commits to `brain/`, occasional manual edits to prompts, and `/ingest` of external content. We did not freeze operator behaviour for the benchmark. The figures reflect normal operation, not a clean-room experiment.

Short window. Eighteen days is short for a system that amortises learning across months. Some of the patterns observed (e.g. the learn-loop's 76.5% gate acceptance rate) may regress under longer observation as the gate catches further failure modes the operator does not anticipate.

13.2 External validity

Single operator. Section 9′s client cases are all from one consultancy. The patterns distilled into the playbook (`sitemap-crawlbudget-focus`, `positionering-indexatie`, etc.) may not generalise beyond small-site SEO contexts. A larger deployment would stress the scheduling assumptions in ways we have not observed.

Single LLM. All task executions invoke Claude (variously Opus 4.6 and Opus 4.7 over the window). The `opus-4-7-prompting-contract.md` playbook entry documents behaviour changes observed between these two versions; it is plausible that non-Anthropic models would require different harness details.

Specific deployment surface. The enforcement-layer examples are scraping-blocked platforms and a global `CLAUDE.md` protection hook. A different domain (e.g. medical question-answering) would require a different predicate set; we make no claim that the current predicates transfer.

13.3 Construct validity

SEO outcomes. Case study figures in Section 9 are before–after contrasts without A/B controls. Google Search Console traffic is a non-stationary process with weekly seasonality; some of the reported lifts overlap plausibly with background drift. Case 4 (honest negative) explicitly illustrates the risk of reading early signals as confirmation.

Evo-gate pass rate as a metric. A high pass rate is not necessarily good: the gate could be lenient. A low pass rate is not necessarily good either: the learn-loop could be destructive. The rate is a diagnostic, not a score. We report the rate to show the gate is actively rejecting, not to argue for a target value.

14 Limitations

Single-tenant assumption. The design assumes one operator who reviews, curates, and occasionally corrects. Multi-tenant scenarios with conflicting edits are outside scope. The enforcement layer in particular would need to resolve conflicting predicates across operators, which we have not studied.

Scale of corpus. Our corpus is 111 canonical files. Retrieval behaviour at the thousand-file scale may differ; saturation thresholds may need tuning.

Evaluation window. Eighteen days is short for a system that amortises learning across months. The cases in Section 9 cover five months but rely on external analytics (GSC, Vercel) rather than Myelin's own logs.

LLM-dependence. Every task invokes Claude Code. Behavioural differences between Opus 4.6 and Opus 4.7 have already required prompt adjustments (see *opus-4-7-prompting-contract.md* in the playbook). Model migration is an open operational cost.

No statistical significance tests on SEO cases. Section 9 reports before–after contrasts without A/B controls. Google Search Console traffic is a non-stationary process with weekly seasonality, and some of the reported lifts overlap plausibly with background drift. Case 4 (the honest negative) illustrates exactly the risk of reading early signals as confirmation.

15 Operational Notes

This section collects observations from six weeks of running Myelin that are not strictly results but matter to reproducibility.

Git as the audit log. Every append-only log file is under version control; every decision is a commit. Multiple near-failures during development were diagnosed by walking `git log` on `data/eval-log.jsonl` and correlating exit codes with prompt diffs. A traditional logging stack would have separated application state from code changes; conflating them in git has been a consistent win.

Auto-commits as noise. Automated commits (`brain: ...` message prefix) can drown manual commits in the log. Tooling filters these out when computing the "last human touch" signal used by decay (Algorithm 1, step 2). This asymmetry between automated and manual commits is worth stating explicitly because it inverts the usual assumption that all commits are equal.

Hook debugging. `PreToolUse` hooks that fail spuriously (e.g. a Python import error) can soft-lock the agent. The lesson learned the hard way: keep hooks small, keep them Python-stdlib only, and log their decisions to a rotating file so the operator can see what the hook is thinking even when the agent cannot.

Timezone and scheduling. The cron schedule operates in the VPS timezone (UTC), but content refers to Dutch working hours. Translating between the two in the operator's head was a source of several minor bugs. The fix was not to change the schedule but to add a datetime helper that always reports both.

Discord rate limits. Seven agents posting independently occasionally bunch up and hit Discord's webhook rate limit. A 30-second soft queue on the sending side eliminated this without losing messages.

The pre-flight lockfile. The daily 05:50 lockfile that blocks all tasks on test failure has fired four times across the project's lifetime. Each firing prevented a cascading mis-run that would have taken half a day to clean up manually. The lockfile is a load-bearing piece of infrastructure even though it does nothing positive on any given day.

16 Conclusion

Myelin is what a self-evolving agent looks like when the author refuses to write any abstraction that git, cron, and markdown do not already provide. Over eighteen

days of unsupervised production operation the system executed 289 tasks at 94.1% completion, refused 8 of its own self-mutations at a deterministic structural gate, and supported a one-person consultancy managing six client sites. The three open failure modes (stale retrieval bias, evo-gate quality gap, daily-log retrieval pollution) are documented with the same empirical rigour as the successes, because they are the next honest piece of work.

We release the engine as open source at `github.com/AIFAIS/myelin`. The reproduction artefact for this paper, including the 18 days of raw logs and the fault-injection harness, is included in the repository under `paper/` (build instructions in `paper/README.md`).

17 Deployment Timeline

For context, the system's operational history is summarised below. Numbers are commit-grounded; milestone events are tagged in the repository.

Date	Event
Nov 2025	First task runs headlessly on VPS.
Jan 2026	Enforcement hooks added; AIFAIS pivot commit.
Feb 2026	Evo-gate introduced; first learn-02 mutation.
Mar 2026	Discord two-way bot; pre-flight test suite.
Apr 2026	Observation window begins (04-01).
Apr 2026	FRESCO bias measured on own corpus (04-18).

Table 10: Deployment milestones since the first commit in November 2025. The evaluation window of this paper covers the final eighteen days of the timeline.

Total commits to date: 726, of which 527 during the observation window (29.3 per day). Git lifecycle is the paper's evidence trail: every mutation in a reported metric corresponds to at least one commit reachable from `master`.

A Hardware and Operating Costs

Compute. A single Hetzner CX22 VPS (2 vCPU, 4 GB RAM, 40 GB NVMe, €3.79/month as of April 2026). Peak daily compute observed in the window was approximately 43 minutes of Claude Code wall-clock time. The VPS's CPU is used primarily for bash, Python utility scripts, Playwright scraping, and QMD's embedding workloads.

LLM usage. Task executions route to `claude -p` which consumes an Anthropic subscription token allowance. Over the observation window, usage stayed inside the subscription ceiling; no per-call API spend is attributable to Myelin in the reported period.

Retrieval. QMD 2.0.1 with Qwen3-Reranker-0.6B runs on the same VPS (no GPU). Model weights are approximately 1.6 GB on disk; one-time embedding of the 248-file corpus takes approximately 10 minutes on the CX22.

Discord. Seven webhook URLs plus one bot token for the two-way `#nikola` channel. No ongoing cost beyond the standard Discord server.

Total monthly cost: €3.79 (VPS) plus an existing Anthropic subscription.

B Reproduction

Clone. `git clone https://github.com/AIFAIS/myelin`

Install. Requires `bash >= 5.0`, `git`, `python >= 3.10`, `bun` (for Playwright scouts), and the `claude` CLI. See `server/README.md` for the full list.

Configure. Copy `server/config.sh.example` to `server/config.sh` and fill in Hetzner IP, Discord webhooks, and repository list. The file is the single source of truth for deployment targets.

Install cron. `bash server/install-crontab.sh` reads `server/crontab.txt` and installs the task schedule.

Verify. `bash server/tests/run-all-tests.sh` runs eighteen structural tests (prompt lint, config drift, cron integrity, data dependencies). A green run means the deployment is structurally valid.

Reproduce this paper's metrics. `python paper/metrics.py --logs brain/data/` extracts the numbers cited in Section 8 directly from the logs committed alongside this paper.

C Frontmatter Schema

Every knowledge item serialises Definition 1 as YAML frontmatter:

```
---
confidence: 0.85       # 0 .. 1, decayed weekly
evidence: 3            # count of confirmations
domain: automation     # for saturation scoring
last_used: '2026-04-17'
last_validated: '2026-04-13'
stale: false           # true if > 90 days
archive_candidate: false  # true if > 180 days
---
```

Required fields: `confidence`, `evidence`, `domain`, `last_validated`. Optional: `last_used`, `stale`, `archive_candidate`. The `source` field is added when an item is derived from external content (a URL, paper, or repository).

D Session-Start Hook

The failure-ranking injection is implemented by a shell hook invoked by Claude Code's `SessionStart` event. Abbreviated schema:

```
#!/usr/bin/env bash
set -euo pipefail
python3 "$BRAIN/scripts/rank-failures.py" \
  --log "$BRAIN/data/failure-categories.json" \
  --k 5 \
  --half-life-days 14 \
  > "$BRAIN/BRIEFING.md.tmp"
cat "$BRAIN/BRIEFING.md.tmp" "$BRAIN/BRIEFING.md" \
  > "$CLAUDE_CONTEXT_FILE"
```

The hook composes a briefing file that is read by Claude Code on every new session. Dynamic ranking of failures is combined with static operator memory (`MEMORY.md`, `mark-denkpatronen.md`) and the previous day's Discord digest to form the session context.

Acknowledgements

The system owes its core design commitments to several operator traditions encoded in its `mental-models/` folder: Peterffy's *automate everything*, Cursor's *raise the ceiling*, Munger's inversion checklist, and Sankar's *forward deployed* stance on problem ownership. The convergence with Hermes-Agent and the Claude Code analysis suggests that these traditions are not coincidental but load-bearing.

References

[1] Anonymous. *Autogenesis: A Self-Evolving Agent Protocol.* arXiv:2604.15034, April 2026.
[2] Anonymous. *FRESCO: Benchmarking Re-rankers for Evolving Semantic Conflict in RAG.* arXiv:2604.14227, April 2026.
[3] Anonymous. *Dive into Claude Code: The Design Space of Today's and Future AI Agent Systems.* arXiv:2604.14228, April 2026.
[4] NousResearch. *Hermes-Agent.* `github.com/NousResearch/hermes-agent`, 2026.
[5] Myelin contributors. *Observer-Mode LLM-Judge.* Playbook entry `observer-mode-llm-judge.md`, AIFAIS/myelin, 2026.
[6] Husain, H. *Evaluating LLMs as Judges.* `hamel.dev`, 2025.
[7] Liu, J. *Context Engineering.* `jxnl.co`, 2025.
[8] Rumelt, R. *Good Strategy, Bad Strategy.* Crown Business, 2011.
[9] Anthropic. *Agent Harness Experiments.* Internal report, March 2026.