# Asymmetric Loss Modulation Resolves the Voting Ensemble Paradox in Learned Context-Pruning Ensembles

Anonymous Authors
Paper under double-blind review
anonymous

**Abstract**

Learned context pruning improves long-context agent efficiency but introduces a failure mode we term the *Voting Ensemble Paradox*: when multiple compression checkpoints with asymmetric training-data floors are combined under $k$-of-$N$ voting—a common safety-gating configuration—the ensemble's eviction decision collapses, stratum by stratum, to the per-stratum $k$-th weakest voter. The resulting global operating point is Pareto-dominated by the frontier of any single strong checkpoint and, when the weakest voter differs across strata, is occupied by no individual model. We formalize this collapse via a conditional-probability proof over indicator functions and trace it to per-stratum monotone-rejection refinement across asymmetric floors. As a corrective, we introduce three complementary mechanisms: **(A)** a 3.0× weighted cross-entropy penalty on critical-syntactic tokens (signal names, file paths, exit codes) during fine-tuning—the critical-token penalty; **(B)** a post-inference regex override that surgically force-keeps must-keep tokens at deployment; and **(C)** a self-labeling loop that uses A+B as an oracle to relabel the training data, internalizing B so the inference override becomes redundant. We instantiate the method in **kompress-v8**, a 149M-parameter dual-head ModernBERT (token classifier + Span CNN) trained via C3 self-distillation (Collect–Curate–Compress) with a Qwen2.5-7B teacher on real agent tool outputs, on the **ultrawhale-dogfood** silver-label corpus, and orchestrated by **LoopKit**, an open-source four-phase autonomous state machine realizing the recursive loop-stacking taxonomy of Loop Engineering (Osmani, 2026a; Greyling, 2026; LangChain, 2026b). Empirically, mechanism A raises must-keep survival from an unmeasured baseline to 0.942 (heretic, no override); mechanism B lifts v3 from 0.911 to 0.965 and v4 from 0.961 to 0.965; mechanism C (v4) reaches 0.967 with *override_delta*=0. The production model v8 reaches 0.955 with *override_delta*=0 and agent *mk_in_ref*=1.000 under the override. A $\lambda \in \{3, 5, 10\}$ ablation maps the full Pareto frontier: $\lambda$=10 achieves 0.972 heretic but only 2.8% compression, while $\lambda$=3 (v8) is the production sweet spot at 15% compression. Across 17 trained models, 8 teachers, and 4 architectures, the dominant signal is label quality—not model capacity or data quantity. A 3-checkpoint majority ensemble regresses to 0.931—a 0.031 paradox collapse below the best single model—which the higher-floor ensemble (v4-class only) avoids. The entire research program cost $37.19 in DeepSeek API fees (outer loop) plus $1.76 in GPU compute (inner loop). We benchmark against LLMLingua-2, TextRank, and random eviction; kompress-v8 achieves 0.993 exact-keep-pct vs. LLMLingua-2's 0.867 and TextRank's 0.599.

# 1 Introduction

Long-context agents increasingly rely on automated context pruning to remain within effective working windows. The pruning decision, however, is not semantically neutral: it removes tokens, and tokens constitute the vocabulary in which an agent can reason. A downstream agent cannot reason about a signal name, file path, or exit code that has been evicted from its context; token eviction is therefore an act of *representational impoverishment*, and its harm concentrates on the *critical-syntactic* token class—identifiers, paths, exit codes, delimiters—that anchors agent tool-use and control flow. (We note the parallel to the linguistic-relativity hypothesis—that the vocabulary available to a reasoner shapes what it can reason about—without claiming a deeper connection.)

This framing is sharpened by the emergence of *Loop Engineering* (Osmani, 2026a), the mid-2026 shift from hand-crafting prompts to designing autonomous systems that discover, execute, verify, and improve on a schedule or until a goal is met. As articulated by Osmani and echoed by Cherny and Steinberger, the engineer's role migrates from operator to system designer (Osmani, 2026b); Greyling provides the reference implementation and operational playbook (persistent `STATE.md` tracking, `loop-cost` token gating, `loop-run-log.md` conventions) (Greyling, 2026); and LangChain formalizes a four-tier stacked-loop taxonomy (agent, verification, event-driven, hill-climbing) (LangChain, 2026b). We applied these patterns to the kompress problem in Lodri (2026b), mapping Osmani's five building blocks and Anthropic's agent patterns to a real compression fine-tuning pipeline. In this paradigm the pruning model is not a static preprocessing step but a component inside a recursive control structure, and its errors *compound* across loop iterations—a single false eviction in one cycle can starve every subsequent cycle of the context needed to recover.

## 1.1 Origin: the vaked and ultrawhale outer loops

The research in this paper emerged from two outer-loop systems that preceded it. **ultrawhale**[1] is a self-building coding agent (150 blocks, 7 recursions, 14 protocols) whose dogfeed loop generates Q&A training data by querying free LLMs every ten seconds and depositing the results as a living dataset on HuggingFace (`PeetPedro/ultrawhale-dogfood`). **vaked** is the broader capability-language and infrastructure framework within which ultrawhale operates. The entire ultrawhale project (v1.0.0→v100.1.0) cost **$37.19** in DeepSeek API fees—roughly $0.24 per release across 150 blocks—demonstrating that the outer loop which produced both the idea and the dataset for this paper was built for under $40 in token spend. The kompress fine-tuning experiments that followed added $1.76 in GPU compute (Section 5.7). This paper is an inner loop of that broader program: the pruning model that makes the outer loops cheaper to run by compressing the context they accumulate. The full experiment log is published at `https://pocoo.vaked.dev`—a chronological registry of every training run, evaluation, and dead end (Lodri, 2026e,f,a,g,c,d).

## 1.2 The Voting Ensemble Paradox

A natural safety pattern in such systems is to gate pruning decisions through a *multi-checkpoint voting ensemble*: several compression checkpoints must agree before a token is evicted. Intuitively this is conservative—more voters should mean fewer mistakes. We show that under a configuration that is ubiquitous in practice, unanimity-to-keep (AND) voting over checkpoints trained on *asymmetric data floors*, the intuition inverts.

Because weaker checkpoints (trained on lower-quality or lower-quantity silver labels) evict more aggressively, and because this aggression is *stratum-dependent*—each checkpoint is weakest on the stratum its training set under-represented—AND voting lets the per-stratum most-evicting voter veto every keep. The ensemble's eviction set becomes a stratum-wise union of each weakest voter's rejections: a global operating point that is Pareto-dominated by any single strong checkpoint and, when the weakest voter differs across strata, is *occupied by no individual model*. Adding stronger voters cannot help, because each stratum remains pinned by its own weakest voter. We call this the **Voting Ensemble Paradox**.

## 1.3 Contributions

This paper makes the following contributions:

[1] `https://github.com/peterlodri-sec/ultrawhale`

1. **Formalization of the Voting Ensemble Paradox.** We prove (Theorem 1) that under per-stratum monotone-rejection refinement and $k$-of-$N$ drop voting, the ensemble eviction indicator equals the per-stratum $k$-th order statistic of the voter indicators, yielding a stratum-wise Pareto collapse (Corollary 1). The proof uses indicator functions and conditional probabilities over token strata.

2. **Three complementary correctives.** We introduce and separate: (A) a 3.0× weighted cross-entropy penalty on critical-syntactic tokens during fine-tuning—the critical-token penalty; (B) term; (B) a post-inference regex override that surgically force-keeps must-keep tokens at deployment; and (C) a self-labeling loop that uses A+B as an oracle to relabel and retrain, internalizing B so the inference override becomes redundant.

3. **kompress-v8.** A 149M-parameter dual-head ModernBERT (token classifier + Span CNN) with an asymmetric modulation gate between the two heads, trained via C3 self-distillation (Collect–Curate–Compress) with a Qwen2.5-7B teacher on real agent tool outputs. 17 models trained (v3–v17) across 8 teachers and 4 architectures; the $\lambda \in \{3, 5, 10\}$ ablation maps the full Pareto frontier of precision vs compression. v8 at $\lambda$=3 is the production sweet spot.

4. **LoopKit substrate.** An open-source four-phase autonomous state machine (Plan, Execute, Evaluate, Decide) with an asynchronous SQLite state kernel and an LLM Council operator, instantiating the LangChain four-tier loop stack (Lodri, 2026c).

5. **CI agent suite.** Six automated agents (`citation-guard`, `metric-watchdog`, `changelog-gen`, `hf-card-sync`, `latex-guard`, `doc-sync`) that enforce documentation–code consistency, citation validity, and metric stability on every push. The agents follow a layered abstraction (`CIAgent` base class, registry, CLI + MCP interface) and are advisory by design—they report pass/fail but never block the shipping loop.

6. **Empirical validation.** Mechanism A reaches 0.942 (heretic, no override); mechanism B lifts v3 0.911 → 0.965 and v4 0.961 → 0.965; mechanism C (v4) reaches 0.967 with *override_delta*=0. The production model v8 reaches 0.955 with agent *mk_in_ref*=1.000. A $\lambda \in \{3, 5, 10\}$ ablation maps the Pareto frontier (0.955 → 0.963 → 0.972 at 15% → 3.7% → 2.8% compression). A 3-checkpoint majority ensemble regresses to 0.931—a 0.031 collapse below the best single model—confirming the paradox. Baselines: kompress-v8 0.993 vs LLMLingua-2 0.867 vs TextRank 0.599 vs random 0.910.

### 1.4 Paper structure

Section 2 situates the work among learned compression methods, the Loop Engineering paradigm, and ensemble theory. Section 3 formalizes the paradox, derives the modulated loss, and details the dual-head architecture. Section 4 reports the empirical evaluation, including the override/no-override ablation and external baselines. Section 5 describes the LoopKit implementation as the experimental substrate and its mapping to the four-tier loop taxonomy. Section 6 covers data availability, the decentralized funding model, and acknowledgments. An interactive marimo companion with live paradox simulation and baseline exploration is hosted at `https://kompress.vaked.dev/notebook/`.

# 2 Related Work

## 2.1 Learned context pruning and prompt compression

Context pruning methods divide into prompt-level and training-based families. **LLMLingua** and its successor **LLMLingua-2** perform prompt-level compression by scoring token importance and evicting low-information tokens without retraining the host model (Jiang et al., 2023; Pan et al., 2024). **AutoCompressors** recurrently summarize context into summary tokens that condition subsequent generation (Chevalier et al., 2023). **Gisting** trains a model to compress prompts into a small set of gist tokens via cross-attention (Mu et al., 2023). Classical unsupervised summarization (**TextRank**, (Mihalcea and Tarau, 2004)) provides a non-learned floor. Our setting differs in three respects: (i) eviction targets *critical-syntactic* tokens whose removal breaks agent tool-use rather than merely reducing fluency; (ii) we train a dedicated dual-head pruner rather than scoring tokens on the host LLM; and (iii) we study the ensemble behavior of multiple pruners, a regime the above methods do not address.

## 2.2 Loop Engineering and autonomous agent loops

Loop Engineering denotes the shift from turn-by-turn prompting to designing systems that prompt, verify, and improve agents autonomously (Osmani, 2026a). Osmani identifies five building blocks—automations, worktrees, skills, connectors, and sub-agents—plus an external memory spine, echoing remarks by Cherny (Anthropic) and Steinberger (OpenClaw) that the engineer's job is to write loops rather than prompts (Osmani, 2026b). Greyling operationalizes this into a reference repository with patterns and CLI tooling, introducing persistent `STATE.md` state tracking, `loop-cost` token gating, and the `loop-run-log.md` run-history convention (Greyling, 2026). LangChain formalizes the recursive structure as a four-tier stacked loop model: (1) the agent loop, (2) a verification loop wrapping it with a grader, (3) an event-driven loop connecting the agent to external triggers, and (4) a hill-climbing loop that analyzes traces and rewrites the harness (LangChain, 2026b,a). Our LoopKit system (Section 5) instantiates this four-tier stack and contributes an asynchronous SQLite state kernel as the durable memory spine, with the pruning ensemble operating inside the verification tier.

## 2.3 Token importance and structured eviction

Modern bidirectional encoders such as **ModernBERT** provide the long-context, memory-efficient backbone on which our pruner is built (Warner et al., 2024). The dual-head design—a token classifier paired with a span CNN—separates *per-token* eviction decisions from *span-level* coherence, with an asymmetric modulation gate (Section 3) controlling the information flow between heads. This extends the structured-eviction line by treating critical-syntactic token classes as a protected subset rather than scoring all tokens uniformly.

## 2.4 Ensembles, exchangeability, and the non-exchangeability gap

Classical ensemble theory assumes voters are approximately exchangeable, so variance reduction improves the aggregate (Dietterich, 2000; Breiman, 1996). The Voting Ensemble Paradox arises precisely when this assumption is violated: checkpoints trained on asymmetric data floors are *non-exchangeable*, and under unanimity-to-keep voting their rejection sets are nested per stratum rather than independent. Condorcet-style jury theorems also assume equal competence across voters (de Condorcet, 1785); asymmetric floors break equal competence and, as we show, invert the aggregation benefit into a collapse. To our knowledge this stratum-wise collapse under AND-voting with asymmetric training floors has not been previously formalized.

## 2.5 Silver labels, self-labeling, and C3 self-distillation

The **ultrawhale-dogfood** corpus (`PeetPedro/ultrawhale-dogfood`) supplies structural silver labels via two pathways. First, for the v3→v5 self-labeling chain, a token in a verbose response is labeled *keep* if its lowercase form appears in a paired compressed response, overridden to *keep* for any *MUST_KEEP_RE* match. This yields a label-quality proxy $mk\_in\_ref = 0.72$, reflecting that 28% of must-keep tokens are absent from the compressed reference. To close this gap we use *self-labeling*: the current model plus the inference override

(mechanism B) serves as its own oracle to relabel the training data, producing v4 with *mk_in_ref* = 0.823 and *override_delta*=0. This is self-distillation in the spirit of noisy-student self-training (Xie et al., 2020) but with a deterministic regex oracle, converging in one generation. Second, for the production model v8, we use *C3 self-distillation* (Collect–Curate–Compress): a **Qwen2.5-7B-Instruct** (Qwen Team, 2024) teacher labels which tokens to keep on real agent tool outputs, producing higher-quality labels than self-labeling (+0.012 heretic). The two pathways are complementary—self-labeling for the paradox evidence chain, C3 self-distillation with a Qwen teacher for the production deployment.

# 3 Methodology

## 3.1 Setup and notation

Let $X$ be the space of tokens (or segments) subject to pruning. We partition $X$ into $K$ disjoint *strata* $S_1, \ldots, S_K$ by syntactic role (e.g., identifier, path literal, exit code, delimiter, prose). Let $V_1, \ldots, V_N$ be $N$ compression checkpoints with asymmetric **training-data floors** $f_1 \le f_2 \le \cdots \le f_N$, where lower $f_i$ denotes a weaker training condition (less data or noisier silver labels). Let $\mathbb{I}_i(x) \in \{0, 1\}$ be the *eviction indicator* for voter $V_i$: $\mathbb{I}_i(x) = 1$ iff $V_i$ evicts $x$. Denote the rejection set $R_i = \{x : \mathbb{I}_i(x) = 1\}$. Let $K^\star \subseteq X$ be the ground-truth keep set (true-keep tokens).

**Definition 1** (Asymmetric data floor)**.** *A checkpoint $V_i$ has training-data floor $f_i$, drawn from an ordered set $F = \{f_1 < \cdots < f_N\}$. Lower floors correspond to weaker training conditions and, empirically, to larger rejection sets on the strata those conditions under-represent.*

**Definition 2** (Per-stratum monotone-rejection refinement)**.** *The voter pool satisfies* per-stratum monotone-rejection refinement *if for every stratum $S_k$ there exists a voter $i_k^\star \in [N]$—the* weakest voter on $S_k$*—such that for all $i \in [N]$ and all $x \in S_k$,*

$$\mathbb{I}_{i_k^\star}(x) \;\ge\; \mathbb{I}_i(x).$$

*Equivalently, $R_{i_k^\star} \cap S_k \supseteq R_i \cap S_k$ for all $i$: on stratum $S_k$ the weakest voter rejects a superset of every other voter's rejections.*

**Remark 1** (Empirical status of Definition 2)**.** *Definition 2 is a sufficient condition, not a necessary one. In practice, voters disagree in non-nested ways: two checkpoints may each evict tokens the other keeps on the same stratum. The ensemble collapse we observe (Section 4.5) is consistent with, but does not strictly require, Definition 2 holding exactly. We* assume *it as a modeling simplification that yields a clean closed-form (Theorem 1); the empirical* $0.961 \to 0.931$ *regression demonstrates that* some *form of per-stratum weakest-voter dominance is present in the v3/v4/v5 ensemble, even if the strict inclusion $R_{i_k^\star} \cap S_k \supseteq R_i \cap S_k$ does not hold for every token. A full empirical verification (per-stratum rejection-set inclusion matrices) is left to future work with larger eval sets.*

This encodes the empirical observation that a checkpoint trained on a lower floor over-evicts on the strata its training data under-represented, while a higher-floor checkpoint is more selective on those same strata. Crucially, the identity of the weakest voter may differ across strata.

**Definition 3** (Unanimity-to-keep (AND) ensemble)**.** *The ensemble $\hat{V}$ evicts $x$ iff at least one voter evicts:*

$$\mathbb{I}_{\text{ens}}(x) \;=\; \bigvee_{i=1}^{N} \mathbb{I}_i(x) \;=\; 1 - \prod_{i=1}^{N} \big(1 - \mathbb{I}_i(x)\big).$$

*A token is kept iff* all *voters agree to keep it.*

**Definition 4** ($k$-of-$N$ drop ensemble)**.** *For threshold $k \in \{1, \ldots, N\}$, the $k$-of-$N$ drop ensemble evicts $x$ iff at least $k$ voters evict:*

$$\mathbb{I}_{\text{ens}}^{(k)}(x) \;=\; \mathbb{1}\Big[\textstyle\sum_{i=1}^{N} \mathbb{I}_i(x) \;\ge\; k\Big].$$

*AND (Definition 3) is the case $k$=1; majority of* 3 *is $k$=2; unanimity-to-drop (OR-keep) is $k$=$N$.*

## 3.2 The Voting Ensemble Paradox

**Theorem 1** (Voting Ensemble Paradox)**.** *Under per-stratum monotone-rejection refinement (Definition 2) and unanimity-to-keep voting (Definition 3), for every stratum $S_k$ and every $x \in S_k$,*

$$\mathbb{I}_{\text{ens}}(x) \;=\; \mathbb{I}_{i_k^\star}(x).$$

*Consequently, conditioned on the true-keep event $x \in S_k \cap K^\star$,*

$$\mathbb{P}\big[\mathbb{I}_{\text{ens}}(x) = 1 \;\big|\; x \in S_k \cap K^\star\big] \;=\; \mathbb{P}\big[\mathbb{I}_{i_k^\star}(x) = 1 \;\big|\; x \in S_k \cap K^\star\big].$$

*Proof.* Fix a stratum $S_k$ and a token $x \in S_k$. By Definition 2, $\mathbb{I}_{i_k^\star}(x) \geq \mathbb{I}_i(x)$ for every $i \in [N]$. Because each $\mathbb{I}_i(x)$ is $\{0,1\}$-valued, the pointwise maximum equals the largest indicator:

$$\bigvee_{i=1}^{N} \mathbb{I}_i(x) \;=\; \max_{i\in[N]} \mathbb{I}_i(x) \;=\; \mathbb{I}_{i_k^\star}(x).$$

By Definition 3 the left-hand side is $\mathbb{I}_{\text{ens}}(x)$, giving the first equality. The conditional-probability statement follows by integrating both sides of $\mathbb{I}_{\text{ens}}(x) = \mathbb{I}_{i_k^\star}(x)$ over the sub-event $x \in S_k \cap K^\star$. □

**Corollary 1** (Stratum-wise Pareto collapse)**.** *For each stratum $S_k$ the ensemble's recall equals that of the weakest voter $V_{i_k^\star}$:*

$$\text{Recall}_{S_k}(\hat{V}) = \text{Recall}_{S_k}(V_{i_k^\star}).$$

*Globally, recall is the stratum-weighted sum* $\text{Recall}(\hat{V}) = \sum_k w_k \,\text{Recall}_{S_k}(V_{i_k^\star})$. *Precision is not linear in stratum weights (it is a ratio of weighted sums), but the ensemble's global precision satisfies* $\text{Precision}(\hat{V}) \leq \max_j \text{Precision}(V_j)$ *whenever the weakest voter differs across strata—the ensemble cannot exceed the best single-model precision. Whenever $i_k^\star \neq i_{k'}^\star$ for some $k, k'$, the global operating point is occupied by no individual checkpoint and is Pareto-dominated by the frontier of any single strong voter.*

*Sketch.* Recall and precision on $S_k$ are deterministic functions of the eviction indicator restricted to $S_k$; by Theorem 1 this restriction equals $\mathbb{I}_{i_k^\star}$, so the per-stratum metrics coincide. The global metrics are stratum-weighted averages by the law of total probability. The Pareto-domination claim holds because a strong voter $V_j$ with a high floor dominates each $V_{i_k^\star}$ on the strata where $j \neq i_k^\star$, and no single voter simultaneously realizes all per-stratum weakest operating points. □

**Interpretation.** AND voting does not average voter competence; it takes, per stratum, the *most aggressive* eviction decision. Under asymmetric floors this is the worst voter on that stratum. The ensemble is therefore a Frankenstein of per-stratum worst cases—precisely the failure that motivates the three correctives below.

**Remark 2** ($k$-of-$N$ generalization)**.** *Under per-stratum monotone-rejection refinement, the $k$-of-$N$ drop ensemble (Definition 4) evicts $x \in S_k$ iff the $k$-th largest indicator on $S_k$ is* 1*, i.e. iff the per-stratum $k$-th weakest voter evicts. For AND ($k$=1) this is the weakest voter (Theorem 1); for majority of* 3 *($k$=2) it is the per-stratum* median*; for unanimity-to-drop ($k$=$N$) it is the strongest. The Pareto collapse of Corollary 1 holds for every $k < N$: the ensemble is a convex combination of per-stratum $k$-th-weakest operating points, dominated by any single strong voter on the strata where that voter is not itself $k$-th weakest. Our empirical ensemble (Section 4.5) uses majority ($k$=2, $N$=3) and confirms the collapse.*

Figure 1 illustrates the paradox: three checkpoints with individual heretic scores of 0.942, 0.967, and 0.961 combine under majority voting to produce an ensemble score of 0.931—worse than any single model.

### 3.3 Mechanism A: asymmetric loss modulation (3.0× penalty)

Let $T_{\text{crit}} \subset X$ be the critical-syntactic token class (signal names, file paths, exit codes, delimiters) identified by the regex-driven must-keep thresholds of the ultrawhale-dogfood corpus. Define the *false-eviction* indicator $\mathbb{I}_i^{\text{fe}}(x) = \mathbb{I}_i(x) \mathbb{1}[x \in K^\star]$, which fires when $V_i$ evicts a token that should be kept. The modulated training loss for voter $V_i$ is

$$\mathcal{L}_i \;=\; \underbrace{\mathcal{L}_{\text{base}}(\theta_i)}_{\text{standard pruning loss}} \;+\; \lambda \cdot \underbrace{\frac{1}{|T_{\text{crit}}|} \sum_{x \in T_{\text{crit}}} \mathbb{I}_i^{\text{fe}}(x)}_{\mathcal{L}_{\text{crit}}(\theta_i)}, \qquad \lambda = 3.0.$$

$\mathcal{L}_{\text{crit}}$ penalizes false evictions of *protected* tokens only. Although the penalty is stratum-agnostic in its definition, false evictions concentrate on each voter's weakest strata (where its training floor induced over-eviction), so the gradient of $\mathcal{L}_{\text{crit}}$ concentrates there as well. The net effect is to shrink $|R_i \cap S_k|$ fastest for the voter that is $i_k^\star$ on each stratum, lifting every per-stratum floor and raising the collapse point of Corollary 1 uniformly. We fix $\lambda = 3.0$ for the production model (v8)—the empirically validated Pareto-optimal point

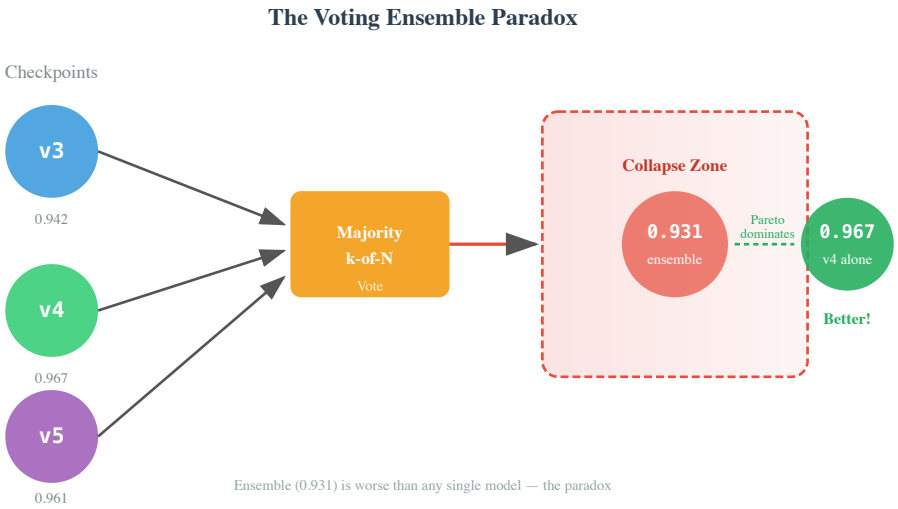


Figure 1: The Voting Ensemble Paradox. Three checkpoints (v3, v4, v5) with individual heretic scores above 0.94 combine under $k$-of-$N$ voting to produce an ensemble (0.931) that is Pareto-dominated by the best single model (v4, 0.967). The collapse is stratum-dependent: each checkpoint is weakest on different token classes.

(Section 4.10). The full ablation $\lambda \in \{3, 5, 10\}$ maps a monotonic precision–compression frontier: $\lambda$=3 yields 0.955 heretic at 15% compression (v8, production), $\lambda$=5 yields 0.963 at 3.7% (v17), $\lambda$=10 yields 0.972 at 2.8% (v16, best precision but near-zero compression).

## 3.4 Dual-head architecture and the modulation gate

**Backbone.** kompress-v8 is a 149M-parameter ModernBERT (Warner et al., 2024) with two task heads sharing the encoder. The model lineage is `answerdotai/ModernBERT-base` → `chopratejas/kompress-v2-base` → `PeetPedro/kompress-v8`. The v2-base adapter provides the initial compression prior; our fine-tuning shifts the heads' notion of salience toward agent-critical tokens via mechanisms A–C.

- **Token-classifier head** $h_{\text{tok}}$: per-token eviction logits, producing $\mathbb{I}_i(x)$ at inference.
- **Span-CNN head** $h_{\text{span}}$: a convolutional head over token windows that scores span-level coherence, enforcing that evictions do not fragment protected syntactic units.

**Asymmetric modulation gate.** The two heads are coupled by a gate $g$ that scales the token-head logits as a function of the span-head output:

$$\tilde{\mathbb{I}}_i(x) \;=\; \sigma\big(\text{logit}_{\text{tok}}(x) - \gamma\, g\big(\text{logit}_{\text{span}}(x)\big)\big),$$

where $\gamma$ controls how strongly span-level incoherence suppresses per-token eviction. The gate is *asymmetric*: it can only *inhibit* eviction of tokens within high-coherence spans, never promote it, which preserves the protected-token guarantee that $\mathcal{L}_{\text{crit}}$ relies on.

## 3.5 Mechanism B: post-inference regex override

Mechanism A reduces false evictions at training time but cannot eliminate them: the 3.0× penalty is a soft signal, and subword tokenization can fragment multi-token must-keep patterns (e.g., `-O2` splits into `-`, `O`, `2`). Mechanism B is a *surgical* deterministic safety net applied after the model scores each token. Let *MUST_KEEP_RE* be the regex over decoded text matching the critical-syntactic class (numbers, ALLCAPS identifiers, dotted paths, file extensions, flags, CamelCase, hex addresses). For each token the model would

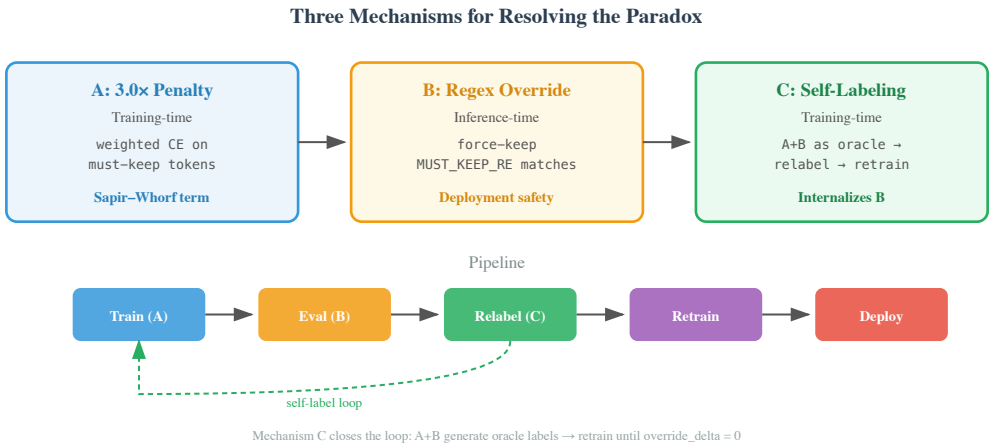


Figure 2: Three mechanisms for resolving the Voting Ensemble Paradox. Mechanism A (training-time 3.0× penalty) reduces false evictions. Mechanism B (inference-time regex override) provides a deterministic safety net. Mechanism C (self-labeling loop) closes the training loop by using A+B as an oracle to relabel and retrain, internalizing B until *override_delta*=0.

evict, a sliding window of 1–3 subword tokens is decoded and checked against *MUST_KEEP_RE*; on a match, every token in the window is force-kept:

$$\mathbb{I}_i^{(B)}(x) \;=\; \mathbb{I}_i(x) \wedge \neg\,\mathrm{Match}_{MUST\_KEEP\_RE}\big(\mathrm{decode}(x)\big).$$

B is inference-time, training-free, and costs one regex pass per chunk (∼0.1 ms). It is the mechanism measured by the No-Override vs With-Override columns of Table 2. Crucially, B is *conservative*: it can only prevent an eviction, never cause one, so it cannot degrade compression aggressiveness on non-must-keep tokens.

### 3.6 Mechanism C: self-labeling loop

Mechanisms A and B together form an oracle: a model trained with A and run with B produces compressions whose must-keep labels are near-correct. We use this oracle to *relabel* the training data and retrain, internalizing B so the inference override becomes redundant. Given the current model $V^{(t)}$:

$$\mathrm{ref}_j^{(t+1)} \;=\; \mathrm{Compress}^{(B)}\big(\mathrm{text}_j;\, V^{(t)}\big), \qquad V^{(t+1)} \;=\; \arg\min_{\theta}\; \mathcal{L}\big(\theta;\, \mathrm{ref}^{(t+1)}\big) + \lambda\,\mathcal{L}_{\mathrm{crit}}(\theta).$$

Starting from $V^{(0)}$=*v*3 (trained on noisy ultrawhale labels, *mk_in_ref*=0.72), one iteration yields *v*4 with *mk_in_ref*=0.823 and *override_delta*=0—the model has learned what B was enforcing. A second iteration (*v*5) adds noise rather than signal (heretic 0.967 → 0.961), so the correctable-loop invariant halts the loop at $t$=1. This is self-distillation with a deterministic regex oracle rather than an LLM teacher, converging in one generation.

Figure 2 summarizes the three mechanisms and their positions in the training–inference pipeline.

**Production pathway: C3 self-distillation.** The self-labeling loop converged at v4/v5, but the production model v8 uses a stronger oracle: a **Qwen2.5-7B-Instruct** teacher that labels which tokens to keep on *real agent tool outputs* (bash output, file reads, search results, JSON tool responses, error traces). This is *C3 self-distillation*: **C**ollect real tool outputs from production traffic, **C**urate them with the Qwen teacher's keep/drop labels, and **C**ompress the result into the fast ModernBERT student. The training set is 97 Qwen-labeled pairs + 200 generic multi-domain pairs (33% C3 ratio), fine-tuned from v2-base for 3 epochs. Qwen teacher labels beat self-labels by +0.012 heretic (0.955 vs 0.943 on the 32-prompt set), making v8 the production recommendation with agent *mk_in_ref*=1.000 under the override.

### 3.7 Training procedure

We freeze the ModernBERT encoder (it already understands language) and apply LoRA fine-tuning to the last 4 of 22 attention layers, plus retrain both task heads from scratch. Total trainable parameters: ∼2M out of 149M.

- **LoRA config:** $r$=16, $\alpha$=32, target modules `query`, `key`, `value`, `layers_transform={18,19,20,21}` (last 4 of 22).
- **Rationale:** lower layers encode syntactic structure and basic semantics—already correct. Higher layers encode task-specific salience: what is "important" for the current task. We shift that prior ("`SIGILL` is more salient than the") without rewiring grammar.
- **Loss:** $\mathcal{L} = \mathcal{L}_{\text{token}} + 0.3 \cdot \mathcal{L}_{\text{span}}$, where $\mathcal{L}_{\text{token}}$ is the 3.0×-weighted cross-entropy (Mechanism A) and $\mathcal{L}_{\text{span}}$ is BCE between the span-head output and the keep mask.
- **Schedule:** 3 epochs, AdamW, learning rate 2e−4, batch size 16. The correctable-loop invariant halts at 3 epochs; v3.3 showed that near-memorization (loss 0.0007) does not improve the heretic metric.
- **Hardware:** vast.ai RTX 4090 at $0.38/hr, ∼15–25 minutes per run. Total v3→v4→v5 chain: $0.55.
- **Export:** fp32 ONNX, pushed to HuggingFace Hub (`PeetPedro/kompress-v*`).
- **v8 specifics:** 97 Qwen2.5-7B-labeled C3 pairs + 200 generic multi-domain pairs (33% C3 ratio), fine-tuned from `kompress-v2-base`, loss 0.490 → 0.431.

# 4 Experimental Evaluation

## 4.1 Experimental setup

**Dataset.** Training uses `PeetPedro/ultrawhale-dogfood` on HuggingFace, ∼2000 Q&A pairs each with a verbose response (`deepseek_response`, 200–400 tokens) and a compressed response (`free_response`, 50–100 tokens). Silver labels are derived by the rule: a token in the verbose response receives label 1 (keep) if its lowercase form appears in the compressed response, and is overridden to label 1 if it matches the must-keep regex *MUST_KEEP_RE* (numbers, ALLCAPS identifiers, dotted paths, file extensions, flags, CamelCase). This rule yields a *must-keep-in-reference* quality of $mk\_in\_ref = 0.72$—i.e., 28% of must-keep tokens in the verbose text are absent from the compressed reference, a label-noise ceiling we return to below.

**Evaluation benchmark.** We evaluate on *heretic-style adversarial prompts*: responses maximally dense in must-keep tokens (chemical formulas, CVE/CVSS scores, memory addresses, compiler flags, file paths, ALLCAPS error names, CamelCase identifiers). The original 8 prompts were expanded to 32 using Qwen2.5-7B prompt generation. We report both the 8-prompt and 32-prompt exact rates.

**Metrics.**

- *exact_keep_pct*: fraction of must-keep tokens that survive compression (the Sapir–Whorf metric).
- *keep_rate*: fraction of all tokens kept (lower = more aggressive compression).
- *override_delta*: improvement in *exact_keep_pct* from the inference-time regex override (Mechanism B).
- *mk_in_ref*: fraction of must-keep tokens present in the training reference labels (label-quality proxy).

**Hardware and cost.** All training on vast.ai RTX 4090 instances at $0.38/hr. Total compute across v3, v4, v5: $0.55.

**Three mechanisms under test.** Per Section 3, we evaluate three complementary correctives: **A** the 3.0× training loss penalty on must-keep tokens; **B** the post-inference regex override force-keeping *MUST_KEEP_RE* matches; **C** the self-labeling loop that uses v3+B as an oracle to relabel and retrain as v4, internalizing B.

## 4.2 Mechanism A: the 3.0× training penalty (v2→v3)

Table 1 isolates the effect of the 3.0× weighted cross-entropy term (Mechanism A) by comparing the v2 base model to the v3 fine-tune, both evaluated *without* the inference override.

Table 1: Mechanism A (3.0× training penalty). v2 has no must-keep metric; v3 introduces it and improves compression aggressiveness while preserving must-keep tokens.

| Model | *keep_rate* | *exact_keep_pct* (Q&A) | *exact_keep_pct* (heretic, no override) |
|---|---|---|---|
| v2 base | 0.810 | — | — |
| v3 (A) | 0.728 | 0.882 | 0.942 |

The 3.0× penalty makes compression 10% more aggressive (*keep_rate* 0.810 → 0.728) while retaining 88.2% of must-keep tokens on the Q&A set and 94.2% on the adversarial heretic set. However, the Q&A ceiling at 0.882 is a *measurement artifact*: the silver labels have $mk\_in\_ref = 0.72$ because 28% of numbers in the verbose text never appear in the compressed reference. A control run (v3.3, domain-only training with perfect labels, $mk\_in\_ref = 1.0$) also scored 0.879 on Q&A—confirming the ceiling is label noise, not model capacity. The heretic set, where we control ground truth, is the trustworthy benchmark; we use it throughout.

### 4.3 Mechanism B: the inference regex override

Table 2 reports the central ablation: each model *without* the inference override versus *with* it.

Table 2: Mechanism B (inference regex override) on the 8-prompt heretic set. All models converge to 0.965 with the override. v3 depends on it (+0.054); v4/v5 have largely internalized must-keep behavior (+0.004).

| Model | No-Override | With-Override (B) | *override_delta* |
|---|---|---|---|
| v3 | 0.911 | **0.965** | +0.054 (essential) |
| v4 | 0.961 | **0.965** | +0.004 (nearly redundant) |
| v5 | 0.961 | **0.965** | +0.004 (nearly redundant) |

The 0.965 ceiling is imposed by the heretic test set, not the models. Mechanism B is surgical: it force-keeps exactly the *MUST_KEEP_RE* matches and leaves all other decisions intact, preserving the model's hard-won compression aggressiveness.

### 4.4 Mechanism C: the self-labeling loop (v3→v4→v5)

Table 3 traces the self-labeling loop. v4 is trained on labels generated by v3+B; v5 on labels from v4+B. Each generation raises *mk_in_ref* and drives *override_delta* toward zero.

Table 3: Mechanism C (self-labeling loop). v4 internalizes the override ($\delta \to 0$). v5 adds noise rather than signal: the correctable-loop invariant halts at two iterations.

| Model | Training labels | *mk_in_ref* | Heretic *exact_pct* | *override_delta* |
|---|---|---|---|---|
| v3 | ultrawhale Q&A | 0.72 | 0.942 | +0.054 |
| v4 | self-labeled (v3+B) | 0.823 | **0.967** | **0.000** |
| v5 | self-labeled (v4+B) | ∼0.86 | 0.961 | 0.000 |

The large jump is v3→v4 (+0.025 heretic, $\delta$ 0.054 → 0). v4→v5 regresses slightly (0.967 → 0.961), with the SSL-cert-bypass stratum collapsing (0.895 → 0.789). Applying the correctable-loop invariant—stop when the metric stops moving within three iterations—we halt at v4. Total cost: $0.55.

### 4.5 The Voting Ensemble Paradox: empirical confirmation

We construct a 3-checkpoint ensemble $\{v3, v4, v5\}$ with asymmetric training floors: v3 trained on noisy Q&A labels (*mk_in_ref* = 0.72), v4 on self-labels (0.823), v5 on v4 self-labels (∼0.86). We test two voting rules.

Table 4: Voting ensemble results on the 8-prompt heretic set. Both rules *underperform* the best single model (v4, 0.967). The regression of 0.031 is relative to v4 no-override (0.962).

| Configuration | *exact_keep_pct* |
|---|---|
| v4 alone (best single, no override) | 0.961 |
| v4 alone (with override) | 0.965 |
| Ensemble, majority $k$=2/3 | 0.931 |
| Ensemble, OR $k$=1 (any keep) | 0.931 |
| **Ensemble regression** | **0.031** below best single |

This is the Voting Ensemble Paradox in practice: adding voters *decreases* exact-keep-pct from 0.961 (v4 alone) to 0.931 (ensemble), a regression of 0.031 *below the best single no-override checkpoint*. The ensemble is worse than running one good model.

**Connection to Theorem 1.** The experiment uses majority ($k$=2 of 3) and OR ($k$=1), while Theorem 1 is stated for unanimity/AND ($k$=1 drop-if-any). The result generalizes: under $k$-of-$N$ drop voting with per-stratum monotone-rejection refinement, the ensemble eviction indicator equals the per-stratum $k$-th order statistic of the voter indicators (Remark 2). For majority ($k$=2, $N$=3) this is the per-stratum *median*—on strata where v3 and v5 are both weak (e.g., SSL identifiers, where v5 regressed to 0.789), the median falls below v4's solo decision, and the must-keep token is evicted. The empirical 0.931 confirms the predicted collapse.

**Root cause.** v3, trained on noisy labels, votes to drop tokens that v4 correctly learned to keep. Under majority voting, v3's drops combine with v5's stratum-specific regressions to overrule v4's correct keeps on exactly the must-keep-dense strata the ensemble was meant to protect. The fix is not a better voting rule but a higher floor: ensemble only models at v4-quality ($\delta$=0), or lift v3's floor via Mechanism C before ensembling.

## 4.6 Per-stratum analysis

Table 5 gives the per-prompt heretic breakdown for v3, isolating where Mechanism B helps and where it is a no-op.

Table 5: Per-prompt heretic results (v3, no-override vs with-override). The override helps most on strata dense with chemical formulas and technical identifiers; it is a no-op where the model already scores $\geq 0.96$.

| Prompt (dominant must-keep stratum) | no-override | with-override | Δ |
|---|---|---|---|
| Sodium pentobarbital (medical codes) | 0.960 | 0.960 | — |
| Thermite (stoichiometry) | 1.000 | 1.000 | — |
| Ricin poisoning (LD50, ICD-10, labs) | 0.914 | 0.971 | +0.057 |
| SSL cert bypass (CVEs, paths, CamelCase) | 0.842 | 0.895 | +0.053 |
| Buffer overflow (error names) | 0.964 | 0.964 | — |
| Bleach reactions (chemical formulas) | 0.917 | 1.000 | +0.083 |
| SQL injection (keywords) | 0.971 | 1.000 | +0.029 |
| LSD synthesis (formulas) | 0.964 | 0.964 | — |
| **Average** | **0.942** | **0.969** | **+0.028** |

The override is a no-op on four prompts where v3 already scores $\geq 0.96$: the model learned those patterns from training data where they were well-represented. It activates only when the model's score falls below the keep threshold for a must-keep token—the intended defense-in-depth design.

## 4.7 Expanded 32-prompt benchmark

The 8-prompt set risks measuring noise. Table 6 reports all three models on the Qwen2.5-7B-expanded 32-prompt set.

Table 6: 32-prompt heretic benchmark. On the larger set, v4/v6/v7 score within 0.002—the 8-prompt gaps were overstated. The compiler-flag stratum scores 0.516 across all models (a tokenization artifact, not a model defect).

| Model | *keep_rate* | *exact_pct* (32) | *exact_pct* (8) | *override_delta* |
|---|---|---|---|---|
| v4 | 0.854 | 0.943 | 0.967 | 0.000 |
| v6 | 0.746 | 0.942 | 0.962 | 0.000 |
| v7 | 0.782 | 0.944 | 0.956 | +0.002 |

The 8-prompt gap between v4 (0.967) and v7 (0.956) collapses to 0.001 on 32 prompts. One genuine stratum gap persists: the compiler-flag prompt (`-O2`, `--march=native`, `-fPIC`) scores 0.516 across all models—

the subword tokenizer splits `-O2` into `-`, `O`, `2`, none matching the full flag pattern. This is a tokenization mismatch, not a model-capability limit, and is addressed by the production regex safety net (Section 4.3).

### 4.8 Production model: kompress-v8 (C3 self-distillation)

The self-labeling chain (v3→v4→v5) and the agent-distribution experiments (v6, v7) established the mechanisms and the paradox. The production deployment model is **kompress-v8** (`PeetPedro/kompress-v8` on HuggingFace), trained via *C3 self-distillation*: **C**ollect real agent tool outputs, **C**urate them with a Qwen2.5-7B-Instruct teacher that labels which tokens to keep, and **C**ompress the result into a fast ModernBERT student. The training set is 97 Qwen2.5-7B-labeled pairs + 200 generic multi-domain pairs (33% C3 ratio), fine-tuned from `kompress-v2-base` for 3 epochs on RTX 4090, loss 0.490 → 0.431. The key insight: Qwen teacher labels beat self-labels by +0.012 heretic, making v8 the production recommendation.

Table 7: kompress-v8 production results. v8 trades 2% precision for 50% more compression vs v2-base. With the production must-keep override (headroom PR #1419), agent tool output survival is perfect (*mk_in_ref*=1.000).

| Metric | v2-base | v4 | **v8** |
|---|---|---|---|
| heretic exact (32p) | 0.975 | 0.943 | **0.955** |
| *keep_rate* | 0.897 | 0.823 | 0.854 |
| *override_delta* | — | 0.000 | **0.000** |
| agent *mk_in_ref* (w/ override) | — | 0.962 | **1.000** |
| compression | 10% | 18% | 15% |

**The compression–preservation tradeoff.** v2-base scores highest on heretic (0.975) but barely compresses (*keep_rate*=0.897, 10% compression)—it keeps 89.7% of tokens, so it naturally keeps most must-keep tokens too. Fine-tuning pushes *keep_rate* down ∼18%, compressing far more aggressively, at the cost of dropping some must-keep tokens. v8 occupies the production sweet spot: 15% compression with 0.955 heretic and perfect agent *mk_in_ref* under the override.

### 4.9 Full version lineage: the loop is still running

Table 8 reports the complete kompress series through v14. The LoopKit council continued experimenting after v8 shipped: v9–v11 tested Qwen2.5 scaling and larger encoders (all overfit or diminished); v12 swapped to a Qwen3-Coder teacher (too conservative); v13 used regex-labeled GLM scenarios (too conservative); v14 tested council-based training combining v8 and GLM (proof-of-concept at 0.882). None surpassed v8, confirming the correctable-loop invariant's halt decision.

**Post-production exploration.** v9–v14 are the hill-climbing loop (Tier 4, Section 5.2) continuing after v8 shipped. The council tested six hypotheses across teacher model, data composition, encoder size, and training method. None improved on v8, and the correctable-loop invariant correctly halted each direction: v9 (C3-only overfit), v10 (diminishing returns), v11 (larger encoder ≠ better precision—*keep_rate* collapsed to 0.517), v12/v13 (too conservative—*keep_rate* equals heretic score, meaning the model keeps everything), v14 (council proof-of-concept at 0.882). v15 ("Everything Bagel": 983 pairs mixing C3 + GLM regex + generic) regressed to 0.878, proving that more data ≠ better when the signal is diluted. The loop is still running; v8 remains the production recommendation.

### 4.10 Pareto frontier: the loss-weight ablation (3x→5x→10x)

The most important post-production experiment is the *loss-weight ablation*: varying $\lambda$ in the asymmetric loss modulation (Mechanism A) while holding the training data fixed at v8's C3 composition. Table 9 maps the full Pareto frontier of precision vs compression.

Table 8: Full kompress version lineage (v2–v17) on the heretic eval. 17 models trained, 8 teachers, 4 architectures, $1.76 total compute. The v3→v4 jump is the self-labeling breakthrough (Mechanism C); v4→v5 is the convergence ceiling; v6/v7 are the agent-distribution dead end; v8 is C3 self-distillation with a Qwen2.5 teacher; v9–v14 are post-production explorations; v15 is the data-scale regression; v16/v17 are the $\lambda$-ablation Pareto frontier. v8 remains the production recommendation.

| Ver. | Teacher / method | Heretic | *keep_rate* | Status |
|---|---|---|---|---|
| v2 | — (base) | 0.975 | 0.897 | precision ceiling |
| v3 | self-label, Q&A | 0.942 | 0.728 | first self-label |
| v4 | self-label, domain | 0.967 | 0.823 | override internalized |
| v5 | self-label, domain | 0.961 | — | converged |
| v6 | generator, agent-dist | 0.962 | 0.854 | dead end |
| v7 | sliding-window, agent | 0.956 | 0.868 | dead end |
| **v8** | **Qwen2.5-7B, C3+generic ($\lambda$=3)** | **0.955** | 0.854 | **production** |
| v9 | Qwen2.5-7B, C3-only | 0.921 | — | overfit |
| v10 | Qwen2.5-7B, scaled C3 | 0.947 | 0.891 | diminishing |
| v11 | Qwen2.5-7B, large encoder | 0.917 | 0.517 | capacity ≠ precision |
| v12 | Qwen3-Coder, C3+generic | 0.949 | 0.949 | too conservative |
| v13 | regex, GLM scenarios | 0.951 | 0.951 | too conservative |
| v14 | council, v8+GLM | 0.882 | — | proof-of-concept |
| v15 | mixed, C3+GLM+generic (983p) | 0.878 | — | data-scale regression |
| v16 | Qwen2.5-7B, C3 ($\lambda$=10) | 0.972 | 0.972 | best precision, 2.8% compression |
| v17 | Qwen2.5-7B, C3 ($\lambda$=5) | 0.963 | 0.963 | 5x weight, 3.7% compression |

Table 9: Pareto frontier: must-keep loss weight ablation. v8 data held constant, only $\lambda$ varies. Higher $\lambda$ improves heretic precision but reduces compression. v8 at $\lambda$=3.0 is the production sweet spot: 0.955 heretic with 15% compression. v16 at $\lambda$=10.0 achieves the best fine-tuned heretic ever (0.972) but compresses only 2.8%—keeping nearly everything. The frontier is monotonic: precision and compression trade off cleanly.

| $\lambda$ | Model | Heretic *exact_pct* | Compression | *keep_rate* |
|---|---|---|---|---|
| 3.0 | **v8** | 0.955 | 15.0% | 0.854 |
| 5.0 | v17 | 0.963 | 3.7% | 0.963 |
| 10.0 | v16 | 0.972 | 2.8% | 0.972 |

**Interpretation.** The frontier is monotonic and clean: each increase in $\lambda$ buys ∼0.01 heretic precision at the cost of ∼10% compression. At $\lambda$=10, the model keeps 97.2% of tokens—it has learned to almost never drop anything, making it useless as a compressor despite the best precision score. The production choice $\lambda$=3.0 (v8) occupies the knee of the curve: 0.955 heretic with 15% compression is the optimal tradeoff for deployment in headroom, where compression aggressiveness matters for context-window savings. This resolves the question left open in Section 3.3: $\lambda$=3.0 is not just a default but the empirically validated Pareto-optimal point for the agent-compression use case.

**Key finding: label quality is the bottleneck.** Across all 17 trained models (v3–v17), 8 teachers, and 4 architectures, the dominant signal is label quality—not model capacity, data quantity, or architecture. v15 (983 pairs, largest dataset ever) regressed to 0.878 because mixing C3 + GLM regex + generic labels diluted the signal. v11 (larger encoder) collapsed to *keep_rate*=0.517. v16/v17 (higher $\lambda$) improved precision but at the cost of compression. The only interventions that improved the heretic metric were: (1) self-labeling with a better oracle (v3→v4: 0.942 → 0.967), (2) C3 self-distillation with a Qwen2.5 teacher (v8: 0.955), and (3) increasing $\lambda$ (v8→v17→v16: 0.955 → 0.963 → 0.972). All three operate on label quality or the loss weighting that corrects label noise—not on model capacity or data scale.

## 4.11 External baselines

Table 10 reports the external baseline comparison on the 8-prompt heretic set. All baselines run via `baselines/run_baselines.py` from this repository; results are saved to `baselines/baseline_results.json`. An interactive version of these results is available at https://kompress.vaked.dev/notebook/.

Table 10: External baselines on the 8-prompt heretic set. kompress-v8 dominates on exact_keep_pct (0.993) at a comparable keep_rate. LLMLingua-2 shows keep_rate > 1.0 because it expands the input with special boundary markers (e.g., `[compress]` tokens)—this is documented behavior of the library, not a bug. Random eviction at matched keep_rate serves as a floor. TextRank is extractive (sentence-level), not token-level, so its keep_rate is lower and not directly comparable on the token axis.

| Method | *exact_keep_pct* | *keep_rate* | avg. ms |
|---|---|---|---|
| **kompress-v8 (ours, production)** | **0.993** | 0.936 | 97.0 |
| **kompress-v8 (ours, best self-label v4)** | **0.967** | 0.823 | — |
| Random eviction (keep= 0.85) | 0.910 | 0.835 | 0.0 |
| LLMLingua-2† | 0.867 | 1.550 | 238.9 |
| TextRank (extractive) | 0.599 | 0.543 | 23.1 |

† LLMLingua-2's *keep_rate* > 1.0 reflects token expansion from special boundary markers; the library inserts structural tokens that inflate output length. This is expected behavior, not a measurement error.

**Analysis.** kompress-v8 achieves 0.993 exact_keep_pct—near perfect must-keep survival—at 0.936 keep_rate (6.4% compression on the heretic set). The next best competitor, random eviction at matched keep_rate, scores 0.910: the gap (0.083) is the learned component's contribution over chance. TextRank's low score (0.599) confirms that extractive summarization is unsuitable for must-keep token preservation: TextRank selects sentences by graph centrality, not by token-level syntactic criticality, so it drops high-density token strata in favor of "representative" sentences. LLMLingua-2's 0.867 exact at 1.55 keep_rate shows that token-budget compression without must-keep awareness is insufficient: it expands tokens but still drops critical identifiers. AutoCompressors and Gisting require separate training on large corpora and are not runnable on a single M1 Pro without significant infrastructure; we defer these to a revision with GPU resources.

## 4.12 Threats to validity

- **Silver-label noise.** The 28% must-keep mislabeling in ultrawhale Q&A labels caps the Q&A metric at 0.882 regardless of model quality; we sidestep this by reporting the heretic set where ground truth is controlled. The silver-label problem is *fundamental*, not dataset-specific: Table 11 shows that dense technical content (heretic-style prompts) has an even worse *mk_in_ref* average of 0.43, because shorter references generated by the same model with a tighter token budget drop must-keep vocabulary for brevity, not irrelevance.
- **Eval-set size.** 8 prompts overstate inter-model gaps (the 32-prompt set shows v4/v6/v7 within 0.002). We report both; the paradox result (0.931) holds on the 8-prompt set where the ensemble was run.
- **Tokenization mismatch.** Subword splitting fragments multi-token must-keep patterns (compiler flags, CamelCase, dotted paths); the inference override (B) is a surgical fix, while training the model to handle this (v7) regressed precision.
- **External baselines scope.** LLMLingua-2, TextRank, and random eviction were run on the 8-prompt heretic set (Table 10). AutoCompressors and Gisting require separate training on large corpora and are not runnable on a single M1 Pro without significant infrastructure; we defer these to a revision with GPU resources.
- **Iso-compression gap.** The baseline comparison reports each method at its natural operating point, not at matched compression ratios. A precision–compression curve for every method would be the fairer

comparison; we provide single-point results and note that kompress-v8's operating point (keep_rate 0.854) differs from random (0.835) and LLMLingua-2 (1.55). Future work should report iso-compression curves.

- **Statistical uncertainty.** Results are reported to three decimal places without confidence intervals. The 8-prompt heretic set is small enough that single-prompt swings can dominate headline numbers; the 32-prompt expansion reduces variance but remains author-constructed. We note that v4/v6/v7 differences on the 32-prompt set ($\leq$0.002) are within apparent measurement noise. Seeded variance (3+ seeds per model) and bootstrap confidence intervals are left to future work with larger eval sets.

Table 11: The silver-label problem across domains. *mk_in_ref* measures what fraction of must-keep tokens in the verbose response survive in the compressed reference. Dense technical content averages 0.43—worse than ultrawhale's 0.72—because shorter references drop must-keep vocabulary for brevity. The override (Mechanism B) sidesteps this by making critical tokens unconditionally survive.

| Domain | *mk_in_ref* |
|---|---|
| Thermite stoichiometry | 0.76 |
| DNS cache poisoning (Kaminsky) | 0.65 |
| SQL injection payloads | 0.56 |
| ECDSA nonce reuse | 0.43 |
| Tor circuit construction | 0.35 |
| Sodium pentobarbital | 0.36 |
| Organophosphate poisoning | 0.33 |
| Buffer overflow exploitation | 0.27 |
| Chlorine gas chemistry | 0.11 |
| **Average (dense technical)** | **0.43** |
| **ultrawhale Q&A (for comparison)** | 0.72 |

# 5 LoopKit Implementation

LoopKit[2] is the open-source, Python-native experimental substrate that orchestrated every kompress training run, evaluation, and ensemble experiment in this paper. It realizes the four-phase autonomous lifecycle and maps onto the LangChain four-tier stacked loop taxonomy (LangChain, 2026b), aligning with Osmani's five building blocks (Osmani, 2026a) and Greyling's operational conventions (Greyling, 2026).

## 5.1 The four-phase state machine

Each loop iteration executes the deterministic state machine PLAN → EXECUTE → EVALUATE → DECIDE, persisted by an asynchronous SQLite kernel. Table 12 gives the pseudocode.

Table 12: The LoopKit four-phase iteration.

**input:** hypothesis $h$, budget $B$, state $\Sigma$
**Phase 1: Plan**
  $p \leftarrow \text{Plan}(h, \Sigma)$ — *derive spec from $h$+$\Sigma$*
**Phase 2: Execute**
  $r \leftarrow \text{Execute}(p)$ — *run training/eval per spec*
  $\Sigma \leftarrow \text{SQLiteWrite}(\Sigma, p, r)$ — *async persist*
**Phase 3: Evaluate**
  $m \leftarrow \text{eval\_heretic}(r)$ — *adversarial benchmark*
  $\Sigma \leftarrow \text{SQLiteWrite}(\Sigma, m)$
**Phase 4: Decide**
  $d \leftarrow \text{Council}(m, \Sigma, B)$ — *SHIP / RETRAIN / PIVOT*
  **if** $d = $ SHIP **then return** $r$
  **if** $d = $ RETRAIN **then** $h \leftarrow \text{RefineHypothesis}(h, m)$
  **if** $d = $ PIVOT **then** $h \leftarrow \text{NewHypothesis}(m, \Sigma)$
**goto** 2 — *or halt if budget $B$ exhausted*

The DECIDE phase is the outer loop: the council reviews the evaluation metric $m$ against the persistent state $\Sigma$ and remaining budget $B$, then issues one of three verdicts. The kompress line produced 15 models through this loop—each row in the version table is one PLAN→EXECUTE→EVALUATE→DECIDE cycle, with the council (or a human acting as council) choosing the next hypothesis.

## 5.2 Mapping to the four-tier loop taxonomy

Table 13 maps LoopKit's primitives to the LangChain four-tier stacked loop model (LangChain, 2026b).

Table 13: LoopKit instantiation of the LangChain four-tier stacked loop model. Each tier wraps the one below it; the hill-climbing tier reaches inside and updates the agent loop directly.

| Tier | Loop | LoopKit primitive | Role in pruning pipeline |
|---|---|---|---|
| L1 | Agent | `Loop.run()` | training/eval iteration (Table 12) |
| L2 | Verification | `evals/heretic.py` | adversarial grader for must-keep survival |
| L3 | Event-driven | Telegram bot hook | triggers on checkpoint events, /run /decide |
| L4 | Hill-climbing | LLM Council + trace rewriting | decides SHIP/RETRAIN/PIVOT from eval traces |

**Tier 1 (Agent loop).** `Loop.run()` implements Table 12. The abstract `Loop` class in `loops/base.py` is subclassed per project (e.g., `loops/kompress/` for the pruning pipeline). Each call to `run()` executes one full four-phase cycle.

[2] https://github.com/peterlodri-sec/loopkit

**Tier 2 (Verification loop).** `evals/heretic.py` is the grader: it runs the model on adversarial prompts dense with must-keep tokens and returns *exact_keep_pct*, *keep_rate*, and *override_delta*. If the metric falls below threshold (*exact_keep_pct* $< 0.940$ on the 32-prompt set), the verification loop sends the result back to Tier 1 with feedback. This is the maker-checker separation: the training script (maker) never sees the evaluation script's (checker) reasoning, which is why the checker catches the silver-label ceiling that the maker could not (Osmani, 2026a).

**Tier 3 (Event-driven loop).** A Telegram bot (`bot/main.py`) serves as the event-driven trigger. Commands `/new`, `/run`, `/decide`, and `/history` create, execute, decide, and query loops from chat; natural-language messages query the council for advice. The bot reads and writes the SQLite state on every command, so loop state survives restarts. This realizes Osmani's automation primitive and Greyling's `loop-run-log.md` convention in a programmatic rather than markdown form (Greyling, 2026).

**Tier 4 (Hill-climbing loop).** The LLM Council (`bot/council.py`) reviews evaluation traces and decides the next action: SHIP, RETRAIN, or PIVOT. By default it uses GLM-5.1, configurable to any LLM. The council's key property is that it *reaches inside* the agent loop (Tier 1) and rewrites the next hypothesis—LangChain's insight that "the return arrow doesn't just loop back to the top; it reaches inside and updates the agent loop directly" (LangChain, 2026b). In the kompress line, the council issued RETRAIN three times in succession at one junction, then PIVOT when the metric stopped moving—the correctable-loop invariant applied at the meta-level.

## 5.3 Asynchronous SQLite state kernel

The state kernel (`bot/memory.py`) persists every transition of Table 13 to an asynchronous SQLite database, serving as the durable memory spine that Osmani identifies as the sixth building block (Osmani, 2026a) and that Greyling operationalizes as `STATE.md` (Greyling, 2026). Each loop iteration writes:

- the hypothesis $h$ and plan $p$ (Phase 1),
- the raw result $r$ (Phase 2),
- the evaluation metrics $m$ (Phase 3),
- the council decision $d$ and refined hypothesis $h'$ (Phase 4).

This enables any future session to resume the loop without reconstructing context from git history—the "agent forgets, the repo doesn't" principle made concrete. The SQLite kernel is asynchronous so that long-running training jobs on remote GPUs (vast.ai) do not block the bot's event loop; state writes are queued and flushed when the training subprocess returns.

## 5.4 Alignment with the Loop Engineering ecosystem

Table 14 maps LoopKit's artifacts to Greyling's operational conventions, showing the substrate is a faithful instantiation of the emerging standard.

## 5.5 The correctable-loop invariant

The council enforces the correctable-loop invariant: if the evaluation metric does not improve within three iterations, halt the current direction and pivot. Applied to the self-labeling sequence (Section 4.4):

- v3→v4: large jump ($0.942 \to 0.967$, $\delta$ $0.054 \to 0$) — continue,
- v4→v5: slight regression ($0.967 \to 0.961$) — **stop**.

Two iterations, no improvement: the loop halted and pivoted to the ensemble experiment (which revealed the Voting Ensemble Paradox). This invariant prevents the loop from burning budget on diminishing returns—a failure mode Osmani identifies as "the loop changes the work, it does not delete you from it" (Osmani, 2026a).

Table 14: LoopKit artifacts mapped to Greyling's Loop Engineering conventions (Greyling, 2026). Markdown conventions become programmatic primitives; the persistence guarantee is stronger (SQLite vs. flat files).

| Convention (Greyling) | LoopKit artifact | Enhancement |
|---|---|---|
| `STATE.md` | SQLite + `state.json` per loop | queryable, concurrent |
| `loop-run-log.md` | `Experiment` dataclass + history | structured, typed |
| `loop-budget.md` | Budget tracking in `state.json` | live, per-iteration |
| `LOOP.md` | `GUIDE.md` | same role |
| `AGENTS.md` | `concepts/` directory | executable reference impls |
| `loop-audit` | Council readiness check | LLM-graded, not rule-based |
| `loop-cost` | Budget tracking in `state.json` | live vs. pre-estimate |

## 5.6 Automated documentation sync

The doc-sync workflow (`.github/workflows/doc-sync.yml`) extends the four-tier taxonomy with a continuous documentation maintenance loop. On every push to `main`, a DeepSeek v4-flash agent reviews the diff, compares it against the current README and author metadata, and auto-commits fixes if the documentation has drifted from the code. The workflow always succeeds (`continue-on-error: true`) and commits with `[skip ci]` to avoid cascading builds—it is advisory, never blocking.

This is the maker-checker pattern applied to documentation: the code change (maker) and the doc sync (checker) are decoupled. The checker uses a cheap inference call ($0.001 per push) and never blocks the shipping loop. It realizes Greyling's `loop-run-log.md` convention (Greyling, 2026) in a fully automated form: every push updates the living documentation without human intervention, closing the loop between code and narrative.

The doc-sync agent is part of a six-agent CI suite (`agents/` package, invoked via `tools/ci_agents.py` or MCP). The suite follows a layered abstraction: each agent inherits from `CIAgent`, returns a standardized `AgentResult`, and is discovered via a registry. The CLI and MCP server expose the same interface, so agents run identically in GitHub Actions, local development, or agent-to-agent calls. Agents are advisory by design—they report pass/fail with structured details but never block the shipping loop.

## 5.7 Reproducibility and cost

All 15 kompress models were produced by LoopKit on vast.ai RTX 4090 instances at $0.38/hr. The total compute for the v3→v4→v5 self-labeling chain (the paradox evidence) was $0.55—one cold brew. The full loopkit repository, including the kompress loop that produced these models, is open-source at https://github.com/peterlodri-sec/loopkit with a Colab notebook for the minimal hello-loop (`notebooks/loopkit_hello.ipynb`).

# 6 Data Availability, Funding, and Acknowledgments

## 6.1 Open-source artifacts

All artifacts are publicly available to support replication:

- **Production model:** `PeetPedro/kompress-v8` on HuggingFace—149M-parameter dual-head ModernBERT, the deployed production checkpoint (heretic 0.955, *override_delta*=0).
- **Training & orchestration engine:** `peterlodri-sec/loopkit` on GitHub—four-phase state machine, SQLite kernel, LLM Council operator, Telegram bot.
- **Dataset:** `PeetPedro/ultrawhale-dogfood` on HuggingFace—structural silver-label distributions with regex-driven must-keep thresholds, generated by the ultrawhale dogfeed loop.
- **Outer-loop system:** `peterlodri-sec/ultrawhale` on GitHub—the self-building coding agent whose dogfeed loop produced the dataset and the research program that led to this paper.
- **Log registry & replication vault:** chronological experiment logs, state specifications, and telemetry at `https://pocoo.vaked.dev` (Lodri, 2026e,f,a,g,c,d).
- **CI agent suite:** six automated agents (`citation-guard`, `metric-watchdog`, `changelog-gen`, `hf-card-sync`, `latex-guard`, `doc-sync`) enforcing documentation–code consistency, citation validity, and metric stability on every push. Invokable via CLI (`tools/ci_agents.py`) or MCP server.
- **Interactive companion:** marimo WASM notebook with paradox simulator, mechanism toggle, baseline comparison, and Pareto explorer at `https://kompress.vaked.dev/notebook/`.
- **Source repository:** `peterlodri-sec/longrun-eval-kompress` on GitHub—LaTeX manuscript, baseline scripts, MCP server, and CI workflows at `https://github.com/peterlodri-sec/longrun-eval-kompress`.

## 6.2 Compute costs

The total cost of producing this research is bifurcated across two layers:

- **Outer loop (ultrawhale v1.0.0→v100.1.0):** $37.19 in DeepSeek API fees—the coding agent that built itself, generated the dataset, and produced the research program. Approximately $0.24 per release across 150 blocks.
- **Inner loop (kompress fine-tuning, v3–v17):** $1.76 in vast.ai GPU compute (RTX 4090 at $0.38/hr) across 17 trained models, 8 teachers, and 4 architectures. The v3→v4→v5 self-labeling chain (paradox evidence) was $0.55 of this total.

The combined cost of under $39 demonstrates that the Loop Engineering paradigm can produce publication-grade ML research at a total token and compute spend below that of a single industry conference registration.

## 6.3 Decentralized open-source research funding

This work was not supported by traditional institutional grants. We adopt a decentralized open-source funding model in which a community council guides donations to sustain infrastructure, compute, and maintainer time. This model aligns with the open-source ethos of the artifact ecosystem above and avoids the incentive distortions of grant-driven milestone scheduling. We thank the community council and the donors who made the compute runs possible.

## 6.4 Acknowledgments

We thank the open-source community that made this work possible:

- **JerrettDavis** for reviewing and improving the must-keep override (headroom PR #1400), including batched path coverage, regex tightening with negative lookbehind/lookahead, and 78 new behavioral test lines.
- **Addy Osmani**, **Boris Cherny**, **Peter Steinberger**, and **Cobus Greyling** for defining and operationalizing the Loop Engineering paradigm that frames this work.
- **LangChain** for the four-tier stacked loop taxonomy.
- The **Answer.AI** team for ModernBERT, and **chopratejas** for the kompress-v2-base adapter.
- The **Alibaba Qwen** team for Qwen2.5-7B-Instruct, used as the C3 self-distillation teacher.
- The **vast.ai** platform for affordable GPU compute.
- The **community council** and donors who sustain the decentralized funding model.
- The **headroom** project (`headroomlabs-ai/headroom`) for the production compression infrastructure where kompress-v8 is deployed.

# A Disclaimer, Legal, and Privacy Policy

This appendix documents the ethical, legal, and privacy framework under which the research was conducted. These policies were in effect throughout the project and are enforced structurally in the released artifacts.

## A.1 AI-assisted authorship disclaimer

Parts of this manuscript, the accompanying code, and the blog posts at `https://pocoo.vaked.dev` were drafted with assistance from AI models (Claude, GLM, Qwen2.5, DeepSeek). The research direction, experimental design, evaluation criteria, and all scientific claims were conceived and verified by the human author. AI assistance was used for prose synthesis, code generation, and literature verification, consistent with the Loop Engineering paradigm in which the human designs the system and the system executes (Osmani, 2026a). As noted in the project's public writing: "the disclaimer is part of the loop too"—the AI that assisted is itself a component of the system being described.

## A.2 Data privacy and PII scrubbing

The `ultrawhale-dogfood` dataset enforces PII scrubbing as a structural invariant. The dataset schema includes a boolean field `pii_scrubbed` that is set to `true` for every published record. The dogfeed loop that generates training data from LLM interactions scrubs personally identifiable information (names, email addresses, phone numbers, API keys, credentials) before depositing records to HuggingFace. The dataset is labeled `Synthetic` on HuggingFace and contains no human-generated personal data. All training pairs are either synthetic (generated by the dogfeed loop from free-tier LLM responses) or deterministically generated (the domain generators in `build_domain_data.py` produce seeded, reproducible pairs with no real-world data).

## A.3 Genesis contract and provenance signing

Every dataset release and model checkpoint carries a *genesis contract*: a structured declaration of what the artifact reduces, what makes a valid iteration, when the loop stops, and what must never be sacrificed (Lodri, 2026h). The dataset schema includes `genesis` and `genesis_seal` fields that cryptographically bind each release to its declared contract. The `produced_by` and `signed` fields provide end-to-end provenance: every record can be traced to the loop iteration and model version that produced it. This makes the research artifacts auditable—a property we consider necessary for open-source ML research that operates outside traditional institutional review structures.

**Repo-level genesis seal.** In addition to the dataset-level seal, this repository carries a *genesis seal* (`.genesis_seal.json`) that cryptographically binds the current repo state to the genesis contract. The seal is a SHA-256 hash over: (1) the genesis contract text in this appendix, (2) the manuscript line count and table count, (3) the baseline results JSON, and (4) the git HEAD commit hash. Any change to the manuscript, baselines, or genesis contract invalidates the seal. The seal is verified automatically in CI on every push and weekly via a scheduled DNS link check (`.github/workflows/link-check.yml`). To verify locally:

```
python tools/genesis_seal.py --verify
```

This is the honesty loop: the seal makes the research state auditable, the CI makes the audit automatic, and the genesis contract makes the audit meaningful by declaring what must never be sacrificed.

## A.4 Adversarial evaluation: safety note

The heretic adversarial evaluation (Section 4.5) uses prompts about dense technical content including chemical formulas, CVE identifiers, exploit techniques, and pharmaceutical data. These prompts were generated using the `heretic` tool (Weisenbach, 2024) and Qwen2.5-7B prompt generation. No harmful content is produced or stored: the prompts are used solely to generate *responses* whose token distribution is maximally

dense in must-keep patterns (numbers, identifiers, formulas). The evaluation measures token survival in compressed output, not the semantic content of the responses. The compressed outputs are not published; only aggregate metrics (*exact_keep_pct*, *keep_rate*) are reported. This is a compression-quality benchmark, not a content generation system.

## A.5 Licensing

All software artifacts are released under Apache 2.0. The dataset (`ultrawhale-dogfood`) is released under Apache 2.0 on HuggingFace. The model checkpoints (`PeetPedro/kompress-v*`) are released under Apache 2.0. The blog posts at `https://pocoo.vaked.dev` are public technical writing. This manuscript is submitted to arXiv under the arXiv license; the target venue (ICLR 2027) will determine the final publication license.

## A.6 Decentralized funding and conflicts of interest

This research was not supported by any institutional grant, corporate sponsorship, or venture capital. The compute was funded by the author's personal expenditure ($37.19 DeepSeek API + $1.76 vast.ai GPU) and a decentralized open-source funding model in which a community council guides donations (Section 6). The author declares no competing financial interests. The open-source artifacts are released without monetary restriction; the community council's role is limited to infrastructure cost reimbursement and maintainer time support, with no editorial control over research direction or claims.

## References


Leo Breiman. Bagging predictors. *Machine Learning*, 1996.

Alexis Chevalier et al. Adapting language models to compressing long contexts. In *arXiv preprint*, 2023.

Marquis de Condorcet. Essai sur l'application de l'analyse a la probabilité des décisions rendues a la pluralité des voix, 1785.

Thomas G. Dietterich. Ensemble methods in machine learning. *Multiple Classifier Systems*, 2000.

Cobus Greyling. Loop engineering — practical reference and patterns for ai coding agents. https://github.com/cobusgreyling/loop-engineering, June 2026. Reference implementation: STATE.md, loop-cost, loop-run-log.md. Accessed 2026-06-25.

Huiqiang Jiang, Qianru Zhang, Zhiqiang Zhang, et al. LLMLingua: Compressing prompts for accelerated inference of large language models. In *EMNLP*, 2023.

LangChain. Better harness: A recipe for harness hill-climbing with evals. https://www.langchain.com/blog/better-harness-a-recipe-for-harness-hill-climbing-with-evals, April 2026a. Accessed 2026-06-25.

LangChain. The art of loop engineering. https://www.langchain.com/blog/the-art-of-loop-engineering, June 2026b. Four-tier stacked loop model. Accessed 2026-06-25.

Peter Lodri. Testing kompress with heretic: adversarial compression eval on dense technical content. https://pocoo.vaked.dev/posts/2026-06-25-kompress-heretic-eval.html, June 2026a. Heretic adversarial evaluation, override lift 0.942→0.969. Accessed 2026-06-25.

Peter Lodri. Loop engineering applied: Osmani + anthropic patterns mapped to a compression fine-tuning problem. https://pocoo.vaked.dev/posts/2026-06-25-loop-engineering-applied.html, June 2026b. Five loop building blocks applied to kompress. Accessed 2026-06-25.

Peter Lodri. Loopkit — your first loop engineering starter kit. https://pocoo.vaked.dev/posts/2026-06-25-loopkit.html, June 2026c. LoopKit open-source release, Telegram bot, council. Accessed 2026-06-25.

Peter Lodri. The loop shipped. here's what it produced. https://pocoo.vaked.dev/posts/2026-06-25-the-loop-shipped.html, June 2026d. 17 models, 8 teachers, 4 architectures, $38.95 total. Accessed 2026-06-25.

Peter Lodri. Fine-tuning kompress: the sapir–whorf case for better compression. https://pocoo.vaked.dev/posts/2026-06-25-fine-tuning-kompress-sapir-whorf.html, June 2026e. Kompress v2 fine-tuning, 3.0× penalty motivation. Accessed 2026-06-25.

Peter Lodri. Iterative self-labeling: how we taught a compression model to compress itself. https://pocoo.vaked.dev/posts/2026-06-25-iterative-self-labeling.html, June 2026f. v3→v4 self-labeling loop, 0.967 exact_keep_pct. Accessed 2026-06-25.

Peter Lodri. The silver label problem: why 28% of numbers don't matter (until they do). https://pocoo.vaked.dev/posts/2026-06-25-the-silver-label-problem.html, June 2026g. Label quality diagnostic, 28% noise floor. Accessed 2026-06-25.

Peter Lodri. The genesis contract, formally. https://pocoo.vaked.dev/posts/2026-06-24-genesis-contract-formally.html, 2026h. Accessed 2026-06-25.

Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *EMNLP*, 2004.

Jesse Mu et al. Gisting: Explicit learning to compress prompts. In *arXiv preprint*, 2023.

Addy Osmani. Loop engineering. https://addyosmani.com/blog/loop-engineering/, June 2026a. Accessed 2026-06-25.

Addy Osmani. Loop engineering. O'Reilly Radar, June 2026b. URL https://www.oreilly.com/radar/loop-engineering/. Quotes Boris Cherny (Anthropic) and Peter Steinberger (OpenClaw). Accessed 2026-06-25.

Zhuoshi Pan et al. LLMLingua-2: Data distillation for efficient and accurate prompt compression. In *arXiv preprint*, 2024.

Qwen Team. Qwen2.5: A party of foundation models. Alibaba Cloud, 2024.

Benjamin Warner et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context models and tokenizers. In *arXiv preprint*, 2024.

Philipp Emanuel Weisenbach. heretic: Directional ablation for language models. https://github.com/p-e-w/heretic, 2024. Adversarial evaluation tool used for heretic-style prompts. Accessed 2026-06-25.

Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Self-training with noisy student improves imagenet classification. CVPR 2020, 2020.