AI as Spacetime Compression: From Past Traces to Posterior Futures

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

September 19, 2025

Modern AI can be understood without metaphysics as a spacetime pipeline that compresses the written past into a predictive present to forecast actionable futures. Human writings are traces of past world states. During training, GPUs thermodynamically concentrate those traces into model parameters, minimizing description length while retaining invariants that matter for generalization. At inference, prompts act as partial observations; the model emits posterior-predictive continuations that function like short-term forecasts. When those outputs guide actions, the loop closes: decisions change tomorrow's state and the next wave of traces. This paper proposes a simple, testable framework: (1) a state-space view of data generation and modeling, (2) a minimum-description-length interpretation of training as compression, and (3) an operations dashboard that treats forecasting quality as a horizon to be extended. We introduce practical, model-agnostic metrics—forecast_horizon H, drift_half_life, bits_per_joule eta, calibration error, and SNR proxies—and show how tool use, structure-first prompting, and continuous verification increase effective signal and lengthen the horizon. The result is a compact theory that ties learning, energy, generalization, and control together in one diagram: Past traces → GPU-compressed Present → posterior-predictive Future → feedback. We outline falsifiers, reportable KPIs, and lightweight procedures that any lab can adopt to manage nonstationarity without changing architectures.

Keywords: spacetime model of AI, minimum description length, posterior predictive, concept drift, forecast horizon, calibration, distribution shift, signal-to-noise ratio, bits per joule, control loop, world models, MLOps.

A collaboration with GPT-5. 68 pages. CC4.0.

AI as Spacetime Compression

$$(\theta^*, \pi^*) \;=\; \arg\min_{\theta,\pi} \; \mathbb{E}_t\Big[-\log p_\theta(x_t \mid o_t,\; C_t) \;+\; \lambda_M C(\theta) \;+\; \lambda_D \,\mathrm{KL}(p_{\text{live}} \,\|\, p_{\text{train}}) \;+\; \lambda_C \,\mathrm{ECE}_t \;+\; \lambda_A \,\ell\big(s_{t+1}\big) \;+\; \lambda_E \,\frac{E_{\text{train}}}{\Delta \text{bits}_R} \Big]$$

Closed-loop context:

World/plant: $s_{t+1} = f(s_t, u_t) + w_t$ • Observation: $o_t = W(s_t, a_t, c_t) + v_t$ • Model: $y_t \sim p_\theta(\cdot \mid o_t, C_t)$ • Controller: $u_t = \pi(o_t, y_t)$ • Tools: $C_t = \text{tools}(o_t, y_t)$

Parameter / Term Legend

- θ: model parameters (the **compressed present**).
- π: action policy mapping text to actions (language-conditioned **controller**).
- x_t: written trace at time t (token sequence).
- o_t: prompt/partial observation from world state s_t.
- C_t: tool/context bundle (retrieval, calculators, schema checks).
- $p_\theta(\cdot)$: model's predictive distribution.
- $C(\theta)$: model complexity cost (capacity/regularization).
- $p_{\text{live}}, p_{\text{train}}$: live vs. training input distributions.
- $\mathrm{KL}(\cdot\|\cdot)$: shift/drift penalty.
- ECE_t: calibration error at time t.
- $\ell(s_{t+1})$: control/impact cost on the next state (safety/utility).
- E_{train}: energy spent for the training update window.
- Δbits_R: info gain on reference set R = $(H_{\text{base}} - H_\theta) \times \text{tokens}_R$.
- f, W: world dynamics and write/measurement operators.
- u_t, y_t: action issued, model output text.
- w_t, v_t: process/measurement noise.
- $\lambda_{M,D,C,A,E}$: weights for complexity, drift, calibration, control, energy terms.

One-liner: choose θ and π to minimize predictive code length + model cost + drift + miscalibration + unsafe future cost + energy per bit—**over the data stream your own outputs help create**.

Figure 1. AI as Spacetime Compression.

Outline

1. Problem Statement and Contributions
   1.1 Why a spacetime model now
   1.2 What is new here (diagram, KPIs, ops bundle)
   1.3 Claims, scope, and intended falsifiers
2. Past as Traces: Data-Generating Story
   2.1 Latent states, actions, and measurements
   2.2 The corpus as an observer-filtered light-cone
   2.3 Noise types: semantic, stylistic, selection, duplication
3. Present as Compression: Training = MDL
   3.1 Objective: L(theta) = data_loss + model_cost
   3.2 Cross-entropy as achieved code length
   3.3 Stiff vs sloppy directions (intuition for invariants)
4. Future as Posterior Prediction
   4.1 Prompts as partial observations
   4.2 Sampling, decoding, and mode selection
   4.3 Verification-first generation (procedural, tabular, calculational)
5. Closed-Loop Effects (Control View)
   5.1 Plant, observer, controller, model
   5.2 When forecasts change the world they predict
   5.3 Stability notions for language-conditioned controllers
6. Nonstationarity and Horizons
   6.1 Concept drift and shift taxonomies
   6.2 Definitions: forecast_horizon H, drift_half_life
   6.3 Triggers: KL bands, CE_fresh gaps, worst-slice alerts
7. Thermodynamics of Learning
   7.1 Energy cost and irreversibility
   7.2 Metric: bits_per_joule eta = delta_bits / energy
   7.3 Efficiency levers: data curation, tokenization, adapters
8. Signal-to-Noise Engineering
   8.1 SNR proxies: dedup rate, n-gram entropy, tool-hit rate
   8.2 Structure-first prompting to raise SNR
   8.3 Retrieval, calculators, and small checkers as variance reducers

1) Problem Statement and Contributions

1.1 Why a spacetime model now

AI systems are deployed into moving targets. The world changes (nonstationarity), users and platforms create feedback (closed-loop effects), and compute budgets bind what we can learn or refresh (thermodynamic limits). Yet most write-ups treat training as an atemporal optimization and inference as a static mapping. We need a model that:

- treats history as traces from past world states (the “past light-cone”),
- treats training as present-tense compression of those traces into parameters (a dissipative, energy-costly act),
- treats inference as posterior prediction that pushes candidate futures forward,
- and treats deployment as a control loop where model outputs change tomorrow’s state.

This spacetime view makes three persistent pains operational: (i) drift detection and refresh timing, (ii) horizon-setting for what the model can reliably forecast, and (iii) compute/energy budgeting tied to actual information gained, not just tokens pushed.

1.2 What is new here (diagram, KPIs, ops bundle)

Unified diagram. We package “Past traces -> GPU-compressed Present -> posterior-predictive Future -> feedback” as a single operational loop that explicitly marks where costs, errors, and decisions accrue.

Minimal KPIs (model-agnostic).

- CE_fresh: rolling cross-entropy on recent, out-of-training data.
- CE_gap := CE_fresh - CE_train: overfitting/shift proxy.
- drift_rate := d/dt[CE_fresh] on a fixed-eval stream.
- drift_half_life: time until CE_fresh increases by ln(2) from a baseline.
- forecast_horizon_H: largest lead k where error_k <= target_band.
- eta_bits_per_joule := delta_bits / energy_train, where delta_bits is achieved description-length reduction on a reference set.
- Calibration: ECE (expected calibration error) and reliability slope/intercept.
- SNR proxies: dup_rate, ngram_entropy, tool_hit_rate during generation.

Ops bundle (lightweight, testable).

- Horizon management: set and track forecast_horizon_H per task; when it shrinks, decide between refresh, adapter-tune, or tool-augment.
- Drift policy: trigger evaluation or refresh when KL_t or CE_fresh crosses guard bands; report worst-slice metrics, not just averages.
- SNR engineering: prefer structure-first prompting (tables, steps, pseudo-code), retrieval where helpful, and calculators/checkers to reduce output variance.
- Energy accounting: log eta_bits_per_joule so "more GPUs" must show "more bits learned," not just more heat.
- Verification norm: "always-be-witnessing" (at least one calc/cite/test per key claim) to raise effective calibration without changing architectures.

1.3 Claims, scope, and intended falsifiers

Claims (what we assert).

1. *Compression–forecast link*: Treating training as MDL compression and inference as posterior prediction yields actionable, testable horizons (forecast_horizon_H) that track real performance under shift.
2. *Operational sufficiency*: The proposed KPIs (CE_fresh, drift_half_life, H, ECE, SNR proxies, eta_bits_per_joule) are sufficient to manage most nonstationary, text-centric applications without modifying the model architecture.
3. *Witness improves horizon*: Adding verification/witness steps (calculations, retrieval, tests) increases effective SNR and extends forecast_horizon_H at fixed compute.
4. *Thermodynamic discipline*: Bits-per-joule eta_bits_per_joule is a practical efficiency metric; pipelines that raise eta also tend to improve generalization per unit energy.

Scope (what we do not claim).

- We do not introduce new architectures, loss functions, or decoding schemes.
- We do not solve long-horizon scientific discovery; we bound near-to-mid horizons and show when to refresh or add tools.
- We remain model-agnostic (LLMs, small adapters, RAG) and domain-agnostic, with examples aimed at text and code tasks.

Intended falsifiers (what would refute or limit the thesis).

- If measured forecast_horizon_H fails to correlate with real downstream success across tasks, the horizon construct is not decision-useful.
- If drift_half_life shows no predictive value for when refreshes help (i.e., refresh timing chosen by this metric does not beat simple time-based schedules), the drift KPIs are not operationally better.
- If adding witnesses/tools does not measurably reduce output variance or calibration error on held-out windows, the SNR claim is invalid.
- If eta_bits_per_joule does not differentiate pipeline choices (e.g., expensive runs yield no greater delta_bits than cheap runs of equal data), the thermodynamic efficiency link is weak.
- If worst-slice reporting and KL/CE guard bands do not catch distribution shift earlier than baseline dashboards, our ops bundle adds no real detection power.

Bottom line. The paper's value is a compact loop model plus a small, cheap scoreboard. If these KPIs do not (a) predict when to refresh, (b) extend usable horizon via verification, and (c) improve bits-per-joule, the approach should be rejected or revised.

2) Past as Traces: Data-Generating Story

2.1 Latent states, actions, and measurements

We model the world as hidden state s_t that evolves under actions u_t and process noise w_t. Humans (and sensors) produce written traces x_t that partially measure s_t with measurement noise v_t.

- Dynamics (conceptual, not committed to linearity):
  - s_{t+1} = f(s_t, u_t) + w_t
  - w_t ~ P_w (zero-mean, covariance Q_t if needed)
- Measurement (text as measurement of reality):
  - x_t = g(s_t; a_t, c_t) + v_t
  - v_t ~ P_v, where a_t are author attributes and c_t is context (venue, incentives, language, time).

- Trace operator:
    - x_t = W(s_t, a_t, c_t) + v_t, where W encapsulates perception, judgment, and expression.

Interpretation:

- s_t contains facts and affordances (what exists and what can be done).
- u_t includes policies, markets, experiments, and everyday decisions. AI outputs can influence u_t (closing the loop).
- x_t are strings (text, code, tables) that encode a lossy view of s_t. A corpus D = {x_1..x_T} aggregates many such views.

Minimal invariants we want models to capture:

- event structure (who/what/when/where),
- constraints (physics, finance identities, definitions),
- stable relations (laws, social regularities),
- compositional programs (procedures, formulas) that re-generate observables.

2.2 The corpus as an observer-filtered light-cone

The written record is not the world; it is the subset of past states that was noticed, valued, and recorded by observers. Let L_t be the set of world events within the past light-cone of time t. Only a subset S_t ⊆ L_t is written, edited, translated, and preserved.

Pipeline (conceptual):

- selection: L_t -> S_t (what gets attention)
- articulation: S_t -> X_t (how it is described)
- mediation: X_t -> X'_t (editing, translation, summarization)
- survival: X'_t -> D_t (what persists and is scraped)

Observer filters include:

- economic incentives (what pays or trends),
- institutional norms (journal, newsroom, platform policy),
- language and culture (lexicon, idioms, taboos),

- technical affordances (token limits, markup, templates),
- curation algorithms (ranking, moderation, duplication removal).

Operational consequences:

- Coverage is uneven across topics, geographies, languages, and time.
- The same s_t can yield divergent x_t depending on a_t and c_t.
- Aggregation hides dissenting measurements; minority readings can vanish.

Practical guardrails for dataset builders:

- log source metadata (time, venue, author class, language),
- keep multiple independent accounts of the same event (do not collapse away disagreement),
- retain timestamps and revisions (versioned traces preserve evolution),
- segment by observer class to enable slice-level evaluation later.

2.3 Noise types: semantic, stylistic, selection, duplication

Text traces carry multiple noise modes. Naming them clarifies which mitigations help and which model KPIs they will affect.

1. Semantic noise (misstatement or uncertainty about the world)

- Definition: content-level errors or unresolved ambiguity about s_t (wrong numbers, false claims, incomplete causality).
- Symptoms: contradictions across sources; future corrections; retractions.
- Mitigations:
    - source triangulation (independent corroboration),
    - attach structured references (ids, links, datasets),
    - prefer documents with embedded procedures (code, equations) that can be replayed.
- KPIs influenced: CE_fresh (improves when semantics are truer), calibration (ECE drops if claims map to outcomes), forecast_horizon_H (extends with reliable semantics).

2. Stylistic noise (surface variation unrelated to meaning)

- Definition: differences in tone, register, punctuation, markup, boilerplate, layout.
- Symptoms: high token entropy without new information; inflated n-gram variety; style-specific artifacts.
- Mitigations:
    - normalization (whitespace, markup, boilerplate stripping),
    - structure extraction (tables, numbers, code blocks),
    - tokenizer choices that down-weight punctuation quirks.
- KPIs influenced: SNR proxies (ngram_entropy down), efficiency eta_bits_per_joule up (less compute to learn surfaces).

3. Selection noise (who/what gets written at all)

- Definition: systematic gaps due to attention, access, censorship, or survivorship.
- Symptoms: topic or locale under-representation; sudden regime-dependent voids; mismatch between real incidence and corpus frequency.
- Mitigations:
    - quota or reweighting by slice (geography, language, domain),
    - targeted data collection for low-coverage slices,
    - explicit “unknown/uncertain” tagging rather than hallucinated fill-ins.
- KPIs influenced: worst-slice metrics (report and optimize), CE_gap stability across slices, robustness under shift.

4. Duplication noise (exact or near-duplicate traces)

- Definition: repeats of the same content across mirrors, syndication, quote chains, or spam.
- Symptoms: inflated effective sample size, shortcut learning, memorization, brittle generalization.
- Mitigations:
    - near-duplicate detection (MinHash/SimHash, Jaccard on shingles),
    - embedding-space clustering with cosine similarity thresholds,

    - dedup within time windows and by source family.
- KPIs influenced: CE_train down without real CE_fresh gains (overfitting signal), forecast_horizon_H may look inflated on familiar prompts but collapses on fresh data; dedup tends to raise eta_bits_per_joule by removing wasted compute.

Quick metric recipes (implementation-agnostic):

- dup_rate := fraction of documents with Jaccard_sim >= tau on character or token shingles within a window.
- ngram_entropy := H of 1–3 gram distributions after normalization; track delta before/after cleaning.
- tool_hit_rate := fraction of generations that invoke calculators/retrieval when numeric claims or named entities appear (proxy for semantic discipline).
- slice_gap := max_slices | CE_fresh(slice) − CE_fresh(global) | over defined observer slices.

Takeaway:

- Treat D as a measured, filtered projection of s_t, not as ground truth.
- Name and track the noise you can control (style, dup), model the noise you cannot (selection), and seek procedures that convert semantic claims into replayable checks.
- Cleaning and metadata do not just prettify the corpus; they raise SNR, improve calibration, and measurably extend forecast_horizon_H at fixed compute.

3) Present as Compression: Training = MDL

3.1 Objective: L(theta) = data_loss + model_cost

Minimum Description Length (MDL) views training as picking parameters theta that yield the shortest two-part code for the corpus D.

- Two-part code:
  L(theta; D) = L_data(theta; D) + L_model(theta)
  where
  L_data(theta; D) = sum_{x in D} (-log p_theta(x))
  L_model(theta) = C(theta)

Interpretations:

- L_data = bits/nats to encode the data using the model as a code.
- L_model = bits/nats to describe the model itself (capacity/complexity).
- MDL chooses theta* = argmin L(theta; D), trading fit against complexity.

Practical surrogates:

- Common training objective: J(theta) = sum_{x} CE_theta(x) + lambda * R(theta)
  where CE is token-level cross-entropy and R is a regularizer (L2, sparsity, low-rank, etc.).
- Map to MDL by reading CE as L_data and lambda * R as L_model.

Compute-efficiency link:

- delta_bits := H_emp(D) − H_theta(D) (achieved reduction vs a baseline code)
- eta_bits_per_joule := delta_bits / E_train
  (bits learned per joule; log alongside loss to see whether extra compute pays)

Refresh decision (rule-of-thumb):

- Retrain/adapter-tune when CE_fresh − CE_train > band OR KL_t to a fresh slice > tau,
  and expected delta_bits / added_energy is favorable.

3.2 Cross-entropy as achieved code length

For a discrete tokenization:

- Empirical cross-entropy:
  H_theta(D) = (1/|D|) * sum_{x in D} (-log_b p_theta(x))
  Units: bits if b=2; nats if b=e. Pick and stick to one base.
- Relation to compression:
  Expected code length per token under optimal coding equals H_theta(D).
  Lower H_theta(D) means a shorter code for D via p_theta.
- Gap that matters:
  CE_gap := CE_fresh − CE_train
    - Small gap: model generalizes; learned invariants transfer.
    - Large gap: overfit or distribution shift; horizon H is shrinking.

- Accounting for duplicates:
  Dedup typically raises generalization at the same H_theta(train) because the model spends fewer bits on memorizing repeats and more on reusable structure.

Metric recipes (rolling window):

- Track H_theta on: (i) held-out i.i.d. slice, (ii) time-advanced fresh slice, (iii) worst slice by source/locale.
- Track delta_bits_window := (H_baseline − H_theta) * tokens_window
  to estimate bits saved by an update.

3.3 Stiff vs sloppy directions (intuition for invariants)

Not all parameter changes matter equally. “Stiff” directions encode invariants; “sloppy” directions chase style/noise. A local lens:

- Let H(theta) = ∇^2 J(theta) (Hessian of the training objective).
- Eigenpairs (lambda_i, v_i) of H:
  - Large lambda_i (stiff): small movement changes loss a lot → likely tied to core regularities.
  - Small lambda_i (sloppy): large movement barely changes loss → likely style, redundancy, or underdetermined quirks.

Operational proxies (no full Hessian needed):

- Sharpness (epsilon-perturbation):
  sharp_eps := J(theta + eps * sign(grad)) − J(theta)
  Lower sharpness (flatter minima) often correlates with better transfer.
- SGD noise stability:
  Re-train small adapters with different seeds; high variance across seeds ⇒ sloppy dominance.
- Fisher diagonal or gradient norm by feature family:
  High, consistent Fisher for a feature family across time ⇒ stiff signal.
- Logit lens stability (for LLMs):
  Stable intermediate predictions across layers/prompts ⇒ stiffer internal structure.
- Parameter-perturbation tests:
  Randomly zero/quantize small-magnitude weights; if CE_fresh barely moves, those directions were sloppy.

Design levers to favor stiff directions:

- Data: increase SNR (dedup, structure extraction, numeric normalization); curate diverse, independent accounts of the same events.
- Objective: weight decay / low-rank adapters / sparsity; mix token-level CE with small amounts of sequence-level consistency losses if available.
- Optimization: flat-minima bias (e.g., SAM-style perturb-and-descent, SWA averaging).
- Decoding/use-time: structure-first prompts, tool calls, and verification (calculations, retrieval, tests) to "lock in" stiff content and de-emphasize sloppy surface variation.

What to log (cheap and useful):

- sharp_eps (fixed eps), per epoch
- seed_variance on small adapter tunes
- CE_fresh under weight pruning/quantization at several thresholds
- Fisher_diag summaries per token/feature bucket
- Correlation: changes in these indicators vs changes in forecast_horizon_H

Takeaway:

- MDL formalizes training as compression with a capacity cost.
- Cross-entropy is the achieved code length; its *fresh* value and gap tell you how much of the compression is reusable tomorrow.
- Favoring stiff over sloppy directions—via data SNR, regularization, and structure/verification at inference—tends to increase bits-per-joule and extend the usable forecast horizon at fixed compute.

4) Future as Posterior Prediction

4.1 Prompts as partial observations

At use time, the prompt is not "the whole world," it is a partial, noisy observation of task state.

- Notation:
    - Hidden task state: h (goal, constraints, hidden variables)
    - Prompt tokens: o (partial observation of h)

    - Context C (history, tools, retrievals)
    - Model emits y ~ p_theta(y | o, C)
- Intent inference (conceptual):
    - Infer z := task intent/type from o.
    - Then decode y conditioned on (o, C, z_hat).
- Practical signals to extract from o:
    - task_type ∈ {fact, code, plan, table, math, summary}
    - required_format ∈ {JSON, CSV, steps, bullet list}
    - required_tools ∈ {calculator, retrieval, plotting}
    - constraints (units, ranges, dates, word limits)
    - uncertainty flags (missing data, ambiguous scope)
- Minimal extractor (pseudo):

1. classify(o) -> task_type, required_format, required_tools
2. parse_constraints(o) -> fields, units, bounds
3. if ambiguity_detected(o): ask one clarifying question OR branch into multiple candidates (pass@k) and verify

- Posterior framing:
  p_theta(y | o) ≈ ∫ p_theta(y | z, o) p(z | o) dz
  In practice we approximate with z_hat and diversify with stochastic decoding or pass@k.

KPIs to log (rolling):

- intent_detect_rate, format_detect_rate
- ambiguity_rate (fraction of prompts flagged)
- clarification_needed_rate vs user acceptance
- schema_conformance_rate (if JSON/CSV requested)

4.2 Sampling, decoding, and mode selection

Decoding chooses which “future” you surface. Choose methods to match task type and risk.

- Common decoders:
    - Greedy: deterministic, low variance; can be brittle.
    - Temperature T < 1: sharper, more conservative.
    - Nucleus (top-p): samples from the smallest prefix mass p; balances diversity/control.
    - Top-k: restricts to k highest-prob tokens.
    - Beam search: explores multiple high-prob paths; good for constrained formats, can overfit style.
    - Constrained decoding: regex/CFG/JSON schema; guarantees structure.
- Mode selection heuristics:
    - factual QA → low T, small top-p (e.g., T=0.2..0.5, p=0.8), plus retrieval/tooling
    - math/code → low T, constrained or self-consistency (pass@k)
    - planning/brainstorm → higher T or p for diversity, then verify parts
    - tabular/JSON output → constrained decoding to schema
    - safety-critical → ensemble or rerank with verifier
- Self-consistency / pass@k (for brittle tasks):
    - Sample k candidates with modest diversity.
    - Score with a task-specific verifier V(y, o) -> {pass/fail or numeric score}.
    - Return argmax_y V.
    - Typical k: 3..10 for latency-sensitive tasks.
- Lightweight reranking features:
    - coverage_score(o, y): fraction of required fields/constraints addressed
    - witness_score(y): presence/validity of calcs, refs, test snippets
    - calibration_proxy(y): hedge language penalties vs numeric commitments
    - schema_valid(y): binary

KPIs to log:

- pass_at_k (by task type)
- verifier_accept_rate
- schema_valid_rate
- latency_per_mode and tokens_per_mode
- error_vs_T,p curves (pick safe bands per task)

4.3 Verification-first generation (procedural, tabular, calculational)

Instead of "generate then hope," generate **with** witnesses you can check immediately.

- Procedural generation pattern:
    1. Plan: emit numbered steps with inputs/outputs.
    2. Execute: for steps that are mechanizable (math, lookups, code), call tools.
    3. Validate: compare outputs to constraints; attach checksums or brief logs.
    4. Present: return a compact answer + an optional "witness" block.
- Tabular generation pattern:
    - Define columns, units, and allowed ranges upfront.
    - Constrained decode to CSV/Markdown table.
    - Run schema + range checks; flag or fix violations.
    - Prefer derived columns that can be recomputed (e.g., total = sum of items).
- Calculational generation pattern:
    - Extract quantities and units from o.
    - Normalize units.
    - Show the formula, plug values, compute result, show rounding policy.
    - Emit a minimal "calc trace" that can be re-run.
- Minimal verifier examples (implementation-agnostic):
    - regex/JSON schema for structure
    - unit checks (e.g., [kg]*[m]/[s]^2 → N)

    - reference checks (count of citations with years in plausible ranges)
    - code tests (do supplied snippets run on a sandbox input?)
    - retrieval cross-checks (top-k doc overlaps with named entities in y)
- Simple selection loop (pseudo):
  for i in 1..k:
  y_i <- decode(o, mode)
  w_i <- build_witness(y_i) // calcs/refs/schema
  s_i <- verify(w_i) // numeric score
  return y_argmax(s_i) with w_argmax
- "Always-be-witnessing" cues in prompts:
    - "Return final answer first, then a short witness block with: (i) formula with units, (ii) computed values, (iii) 1 citation if applicable."
    - "Output valid JSON matching this schema … If a field is unknown, write null, not a guess."
- Variance reduction from verification:
    - Structured outputs and tool calls reduce output variance and calibration error, raising effective SNR and extending forecast_horizon_H at fixed compute.

KPIs to log:

- witness_rate := fraction of generations with a valid witness block
- verification_success_rate := fraction that pass all checks
- correction_rate := fraction auto-fixed by the loop
- numeric_error_dist (MAE/MAPE on tasks with ground truth)
- calibration (ECE) before vs after verification
- horizon_gain := H_with_verification − H_without

Takeaway. Treat prompts as partial observations; pick decoding modes to match task risk; and make verification the default, not an afterthought. The combination tends to (i) shrink CE_fresh on new slices, (ii) improve calibration, and (iii) extend the usable forecast horizon without changing model weights.

5) Closed-Loop Effects (Control View)

5.1 Plant, observer, controller, model

We cast deployment as a feedback system.

- Plant (world): hidden state s_t evolves with actions u_t and noise w_t
  s_{t+1} = f(s_t, u_t) + w_t
- Observer (humans/sensors/platforms): produce measurements x_t of s_t with noise v_t
  x_t = g(s_t) + v_t
- Model (AI): internal predictor p_theta; given observation o_t (prompt + context), emits y_t (text/code/plan)
- Controller (human+AI policy): maps (o_t, y_t) to action u_t that affects the plant
  u_t = pi(o_t, y_t)

Interpretation:

- The *same* model can be safe or unstable depending on pi and the environment's sensitivity df/du.
- Language acts as a **control surface**: y_t (plans, numbers, justifications) shapes u_t (what is done).

Minimal telemetry to log per window:

- action_rate := fraction of y_t that led to u_t (acted upon)
- action_lat := time from y_t → u_t
- effect_size proxy := |x_{t+1} − x_t| on targeted metrics
- forecast_error := loss on predicted vs realized signals (where measurable)

5.2 When forecasts change the world they predict

Once y_t influences u_t, the forecast becomes **performative**: publishing the prediction alters outcomes.

Examples (generic):

- Markets: "X will rise" shifts order flow u_t, changing s_{t+1}; naive backtests overstate skill.
- Safety: "Shortage expected" triggers pre-ordering; the shortage may not occur because u_t adapted.

Operational fixes:

1. **Counterfactual evaluation.**
   When possible, evaluate on slices where y_t was *not* acted upon (u_t unchanged), or use A/B throttles:
   - send y_t to a holdout group; compare realized metrics.
2. **Policy-aware scoring.**
   Score forecasts against **policy-conditional** baselines. Define two losses:
   L_passive: error vs s_{t+1} under u_t = u_passive
   L_active: error vs s_{t+1} under observed u_t = pi(o_t, y_t)
3. **Impact-regularized decoding.**
   Penalize outputs with high estimated impact uncertainty:
   - objective: maximize predicted utility − lambda * impact_var
4. **Reveal uncertainty explicitly.**
   Return intervals and caveats that shape safer u_t (e.g., conservative ranges).

Metric recipes:

- performativity_gap := L_active − L_passive (positive means “our use changed the target”)
- action_counterfactual_uplift: difference in outcomes between acted vs held-out cohorts
- stability_sensitivity := d(error)/d(action_intensity) estimated from binning action magnitudes

Decision rules:

- If performativity_gap >> 0, prefer randomized rollout, stronger guardrails, or delayed aggregation.
- If stability_sensitivity is high, lower temperature, require verification, or elevate to human review.

5.3 Stability notions for language-conditioned controllers

We want the closed loop to be stable: bounded errors under bounded disturbances.

Working notions (no linearity assumed, but intuition helps):

1. **Small-gain style bound.**
   Let G_plant be the plant’s gain from action to state deviation, and G_ctrl be the

controller gain from forecast error to action change. If
G_plant * G_ctrl < 1 (in a relevant band), oscillations are unlikely.
Proxies (operational): regress |Δx| on |Δu| (plant) and |Δu| on |Δe| (controller), where e is forecast error.

2. **Monotone controller heuristic.**
   Prefer pi where action magnitude grows sublinearly with uncertainty. Example:
   |u_t| <= k1 * confidence(y_t) + k2 * |mean(y_t)| with k1 small.

3. **Rate limiting and hysteresis.**
   Add bounds on how fast u_t can change:
   |u_t − u_{t−1}| <= r_max, and hold decisions until evidence crosses a band (avoid flip-flop on small text changes).

4. **Abstention and fallback.**
   If uncertainty > tau or schema/verification fails, output “no-go” plan or escalate rather than forcing a low-confidence u_t.

5. **Consistency checks before actuation.**
   - unit/constraint checks on numeric y_t
   - plan feasibility checks (resource, time, policy)
   - reference integrity (citations accessible, IDs valid)

Stability KPIs (rolling):

- boundedness_rate := fraction of windows with |x|, |u| within target bands
- oscillation_flag := spectral power at policy cadence (peaks suggest control-text oscillations)
- reversal_rate := P(sign(u_t) != sign(u_{t−1})) for decisions that should be smooth
- abstain_rate and escalation_rate with outcome quality on escalations
- error_drift_after_action := CE_fresh measured only on post-action slices (detect self-induced shift)

Guardrail bundle (practical defaults):

- **Risk-tiered decoding:** lower T/top-p, constrained formats, and verification for high-impact tasks; allow diversity only where cost of error is low.
- **Action dampers:** r_max on u_t, plus hysteresis bands on key thresholds.

- **Impact budgets:** per period cap on total action magnitude linked to confidence/witness quality.
- **Dual scoring:** track L_active and L_passive; if gap widens, tighten dampers or increase human-in-the-loop.
- **Event studies:** around policy or prompt-template changes, run pre/post error and reversal-rate comparisons.

Takeaway:

- Deployment is not "ask → answer"; it is a feedback controller where language steers real actions.
- Make performativity measurable, keep controller gains small where the plant is sensitive, and default to abstention or escalation when verification fails.
- These controls reduce oscillations, improve calibration in the wild, and extend the usable forecast horizon without touching the model's weights.

6) Nonstationarity and Horizons

6.1 Concept drift and shift taxonomies

We separate what changes into three buckets; each implies different monitors and fixes.

- Covariate shift (input shift): P_train(x) -> P_live(x) changes, but P(y|x) approx stable.
  Symptoms: prompt style/source distribution shifts; CE_fresh up, calibration ok.
  Fixes: retrieval/tooling, normalization, small adapter tunes; reweight eval slices.
- Label/target shift: P(y) changes, P(x|y) stable (rare for text but occurs in class-balancing tasks).
  Symptoms: priors wrong; confidence miscalibrated.
  Fixes: recalibrate priors; importance weighting; targeted refresh.
- Concept drift (conditional drift): P(y|x) changes (the rule itself moves).
  Symptoms: CE_fresh up with systematic error patterns; witness checks begin failing.
  Fixes: collect fresh traces; emphasize interventional or timestamped data; refresh or adapter-tune.

Operational drill-down (text settings):

- Source shift: venue/platform/lang mix changes.
- Temporal novelty: new entities, IDs, products, laws.
- Policy/definition drift: meaning of terms or constraints updated.
- Tool ecology drift: APIs, calculators, schemas evolve.
- Adversarial pressure: spam, templated prompts, prompt-leak patterns.

Slice design for monitoring (make explicit):

- time_slice (e.g., last_7d, last_30d)
- source_slice (publisher/domain/venue)
- geo/lang_slice
- task_type_slice (fact, code, plan, table)
- risk_slice (impact tier)

6.2 Definitions: forecast_horizon H, drift_half_life

We turn “how far can we trust it” into concrete, per-task numbers.

- Error@lead k: choose a task metric E(k) (e.g., CE_fresh, MAE, F1).
  Compute on prompts whose ground truth or ex-post verification is available with lead k (time between generation and judgment).
- Forecast horizon H (per task):
  H := max k such that E(k) <= target_band.
  Example: for factual QA with verification, target_band could be ECE <= 5% and CE_fresh <= baseline + delta.
- Drift half-life (time to material degradation):
  Pick a fixed eval stream representative of deployment; track CE_fresh(t).
  drift_half_life := smallest h > 0 such that CE_fresh(t+h) - CE_fresh(t0) >= ln(2) (same log base as CE).
  Intuition: time to a one-bit penalty increase per token on that stream.
- Rolling estimators (practical):
  Use overlapping windows W_i of width w. For each W_i:
  CE_fresh[i] := mean CE on fresh set in W_i
  Estimate slope via robust regression; derive half-life h from slope s using linearized

CE_fresh(t) ≈ CE_0 + s*(t - t0):
drift_half_life ≈ ln(2) / s, clipped to [h_min, h_max].

- Link to refresh policy:
  If H falls below H_min for a task slice OR drift_half_life < threshold_h, schedule adapter-tune or refresh; prefer the minimal action that raises H.

6.3 Triggers: KL bands, CE_fresh gaps, worst-slice alerts

We define cheap, automatable triggers to catch trouble early.

- CE_gap trigger:
  CE_gap := CE_fresh - CE_train_on_ref
  Fire when CE_gap > band for m consecutive windows or rises at rate > s_thresh.
- KL divergence trigger (input shift proxy):
  For token or feature histograms p_live vs p_train, compute KL(p_live || p_train) per slice.
  Fire when KL exceeds tau_KL for n of the last N windows or when d/dt KL > slope_thresh.
- Worst-slice alert:
  Maintain slices S. Let E_slice(s) be error on slice s.
  worst_gap := max_s | E_slice(s) - E_global |
  Fire when worst_gap > tau_gap OR when a single slice's H drops below H_min while global H stays acceptable.
- Novelty rate trigger (OOV/NE rate):
  Compute rate of new named entities, IDs, or patterns not seen in training index.
  Fire when novelty_rate > tau_novelty; route to retrieval/tool augmentation before retraining.
- Witness failure trigger:
  verification_fail_rate := fraction of outputs failing schema/calcs/refs in a window.
  Fire when rate exceeds tau_verif; lower temperature, strengthen verification, or escalate.
- Composite risk index (optional):
  risk_index := w1*norm(CE_gap) + w2*norm(KL) + w3*norm(worst_gap) + w4*norm(novelty_rate) + w5*norm(verification_fail_rate).
  Escalate tier (observe -> adapter -> refresh) when index crosses thresholds.

Action ladder (default policy):

1. Observe: increase verification strength, reduce T/top-p, enable constrained decoding on affected slices; add retrieval/tools.
2. Adapter-tune: small, recent-data adapters; reevaluate H and CE_fresh.
3. Partial refresh: targeted fine-tune on curated fresh traces; re-measure eta_bits_per_joule.
4. Full refresh: large retrain only if delta_bits / added_energy is favorable and adapters fail to restore H.

What to log (windowed):

- CE_fresh, CE_gap, H per task/slice
- KL(p_live||p_train) on token/features
- worst_slice id and gap
- novelty_rate (NE/OOV per 1k tokens)
- verification_fail_rate
- actions_taken and horizon_gain_after_action

Takeaway:
Define horizon H and half-life h, monitor simple divergences (KL, CE_gap), and prioritize worst-slice alerts. Use a small action ladder to restore H cheaply. If triggers do not predict horizon loss or actions do not restore H, the monitoring scheme needs revision.

7) Thermodynamics of Learning

7.1 Energy cost and irreversibility

Training turns ordered energy (electricity) into lower description length of data plus heat. The process is inherently dissipative: weights are updated by many irreversible microscopic operations (floating-point multiplies, memory traffic), and even if you “replay” SGD in reverse you cannot recover the exact pre-training state or power.

- Compute→information pathway (conceptual):
  energy_input → arithmetic + memory ops → parameter updates → reduced code length on D (delta_bits) + heat.

- Irreversibility sources:
    - Nonlinear optimization with stochasticity (minibatch sampling, dropout, kernel scheduling).
    - Quantization/rounding; lossy checkpointing; non-deterministic kernels.
    - Data pipeline randomness (shuffles, augmentations).
- Practical implication:
    - Each training run has an **energetic cost of information**. You should decide refreshes by **expected information gain per joule**, not by calendar or habit.
    - The same logic applies to inference-time tool use (RAG, calculators): small extra joules can yield large effective information gains if they reduce description length of answers (lower CE_fresh, better calibration).

Telemetry to record (per run/window):

- energy_train (J) from power meters or provider reports.
- tokens_processed, flops_estimated.
- delta_bits on a fixed reference set (see 7.2).
- eta_bits_per_joule := delta_bits / energy_train.
- wall-clock per 1e9 tokens; Joules/token.

7.2 Metric: bits_per_joule eta = delta_bits / energy

We define a simple efficiency metric that ties energy to information gain.

- Definitions:
    - Baseline code length on a fixed reference set R under a neutral code: H_base(R). Examples of neutral codes: unigram language model; previous checkpoint; gzip/bpe baseline.
    - Achieved code length after training step: H_theta(R).
    - Information gain (bits): delta_bits := (H_base(R) − H_theta(R)) * tokens_R.
    - Energy spent (J): energy_train over the period producing theta.
- Metric:
  eta_bits_per_joule := delta_bits / energy_train (bits per joule)

- Why this is useful:
    - Normalizes different runs, model sizes, and clusters onto a single figure of merit.
    - Encourages **cheap gains first** (data curation, tokenization) before expensive retrains.
    - Makes waste visible: if a long run yields tiny delta_bits, pause and re-assess.
- Practical recipe:

1. Choose a stable reference set R reflecting deployment (stratify by slices).
2. Fix H_base once per study (e.g., previous best checkpoint or a simple baseline).
3. After each significant step (cleaning, adapter tune, partial refresh), compute H_theta(R).
4. Update eta and a cumulative "bits ledger."

- Extensions (optional):
    - eta_fresh: compute delta_bits on a **fresh** rolling set F to reflect live gains.
    - bits_per_dollar = delta_bits / cost_train.
    - bits_per_gpu_hour = delta_bits / gpu_hours.

Decision rules (examples):

- Proceed with a full refresh only if projected eta exceeds a policy threshold or clearly dominates adapters.
- Prefer interventions with highest **eta per day** when time-to-impact matters.

7.3 Efficiency levers: data curation, tokenization, adapters

Many of the largest eta gains come from **pipeline** choices rather than more GPUs.

1. Data curation (raise SNR before compute)

- Deduplication:
    - Remove exact/near duplicates within sources and time windows (SimHash/MinHash, shingling).
    - Effect: lowers CE_train without harming CE_fresh; reallocates capacity to invariants.

    - Expectation: eta ↑ (fewer wasted tokens), forecast_horizon_H ↑ (less shortcut learning).

- Structure extraction:
    - Preserve tables/code/units; standardize numeric formats.
    - Effect: tokens carry more reusable constraints; reduces stylistic noise.
- Slice balancing:
    - Reweight or top-up under-represented slices (geo/lang/task).
    - Effect: worst-slice CE_fresh ↓; robustness ↑ with minimal extra compute.

2. Tokenization (pack more meaning per token)

- Domain-aware vocabularies:
    - Merge frequent multi-token entities (IDs, chemical names, statutes) into stable tokens.
    - Effect: shorter sequences; lower H_theta for the same content; fewer long-range dependencies.
- Numeric/units handling:
    - Canonicalize units; use specialized numeric encodings (e.g., split mantissa/exponent).
    - Effect: easier arithmetic generalization; fewer surface variants.
- Evaluation:
    - log tokens_per_example and H_theta per domain; choose tokenizer that minimizes H_theta on R with minimal vocabulary bloat.

3. Adapters and partial refresh (cheap capacity where needed)

- Small adapters (LoRA/IA3/bitfit) on fresh data:
    - Target drifted slices; keep base frozen.
    - Effect: large CE_fresh drops on target slices for modest energy_train.
    - Expectation: very high eta compared to full retrain; quick restoration of H.

- Retrieval/tool augmentation:
    - Add RAG for facts; calculators for math; schema constraints for outputs.
    - Effect: offloads brittle knowledge to tools; lowers CE_fresh and ECE at inference with tiny energy overhead.
    - Track: tool_hit_rate, delta CE_fresh when tools enabled.
- Curriculum and sampling:
    - Front-load high-value tokens (rare but important entities, updated laws).
    - Effect: steeper early delta_bits curve; earlier plateau detection.

Operational checklist (per intervention):

- Before: record H_theta(R), CE_fresh (by slice), eta to date.
- Apply lever (e.g., dedup pass, tokenizer swap, adapter tune).
- After: recompute H_theta(R), CE_fresh, horizon_H, energy_delta; update eta.
- Keep a **bits ledger** with entries: {date, action, energy_delta, delta_bits, eta, notes}.
    - Use the ledger to pick the next action with highest expected eta and horizon gain.

Takeaway:

- Treat training as a thermodynamic trade: joules for bits.
- Measure efficiency via eta_bits_per_joule on a fixed reference; prefer interventions that raise eta first (cleaning, tokenization, adapters, tools) before expensive retrains.
- When eta is tracked and optimized, you typically get two bonuses “for free”: better robustness (lower CE_fresh, tighter calibration) and longer forecast_horizon_H at the same or lower energy.

8) Signal-to-Noise Engineering

8.1 SNR proxies: dedup rate, n-gram entropy, tool-hit rate

We cannot measure “true information” directly, but cheap proxies indicate when tokens carry reusable signal vs surface noise.

A) Dedup rate (dup_rate)

- Goal: estimate fraction of tokens spent repeating near-identical content.
- Method (windowed):
    1. Shingle each doc into token or char k-grams (k=10..15 chars or 5..8 tokens).
    2. Compute SimHash/MinHash; cluster by Hamming/Jaccard.
    3. Mark a doc as near-duplicate if Jaccard_sim >= tau (e.g., 0.85) within a time/source window.
- Metrics:
    - dup_rate := duplicates / total_docs
    - eff_tokens := total_tokens * (1 − dup_rate_approx)
- Targets:
    - Pre-train corpora: dup_rate <= 5..10% after cleaning.
    - Fine-tunes: dup_rate <= 2..5%.
- Impact:
    - Lower dup_rate → CE_train ↓ with CE_fresh stable or ↓ → eta_bits_per_joule ↑.
    - Watch for over-eager dedup that removes legitimate multi-source corroboration; prefer windowed dedup (by domain+time) over global.

B) n-gram entropy (ngram_entropy)

- Goal: catch stylistic noise and boilerplate that inflate token variety without adding semantics.
- Method:
    - Normalize text (lowercase, strip markup/boilerplate, canonicalize numbers/units).

    - Compute H_n for n in {1,2,3}: H_n := − sum p(g_n) log_b p(g_n).
- Interpret:
    - High H_1 with stagnant CE_fresh suggests stylistic churn; normalize or restructure.
    - Drop in H_2/H_3 after structure extraction with CE_fresh ↓ indicates better SNR.
- Targets:
    - Track deltas, not absolutes; define per-domain baselines and aim for H_n reductions that correlate with CE_fresh reductions.

C) Tool-hit rate (tool_hit_rate)

- Goal: proxy for semantic discipline at generation time.
- Definition:
    - tool_hit_rate := (# generations that invoked an appropriate tool) / (# generations where a tool was indicated)
- Detection:
    - Rule-based cues (numbers, dates, currency, equations, code blocks) → tool_expected = 1.
    - If a calculator/retrieval/schema checker is called and passes → tool_hit = 1.
- Targets:
    - Facts with dates/IDs: > 0.6
    - Math/code tasks: > 0.8
- Impact:
    - Higher tool_hit_rate → lower variance and ECE; typically extends forecast_horizon_H without weight updates.

Quick scoreboard (per rolling window)

- dup_rate, eff_tokens
- H_1, H_2, H_3 (normalized), delta_H_n vs previous window
- tool_hit_rate, tool_pass_rate

- Correlate each with CE_fresh, ECE, horizon_H; keep interventions that improve both CE_fresh and horizon_H.

8.2 Structure-first prompting to raise SNR

Prompts that specify form, units, and checks compress ambiguity and increase effective SNR.

A) Format contracts

- Always request one of {JSON, CSV, markdown table, numbered steps}.
- Example:
    - “Return JSON matching this schema: { 'answer': string, 'witness': { 'formula': string, 'inputs': {name: number, unit: string}[], 'result': number, 'unit': string } }. If unknown, set null.”

B) Constraint articulation

- Name units, ranges, rounding, and allowed nulls.
- Example:
    - “All currency in USD; round to 2 decimals; if a quantity is unknown, emit null (do not guess).”

C) Coverage cues

- List required fields and forbid extraneous ones.
- Example:
    - “Include exactly these keys: date_iso, entity, metric_name, metric_value, unit.”

D) Two-pass pattern

1. Plan: “List the steps with inputs/outputs; do not solve yet.”
2. Execute: “Now execute steps, calling tools as needed; then validate constraints.”

E) Ambiguity branches

- If ambiguity_detected(o) → emit k short alternatives labeled with assumptions; verify each, then pick best.

KPIs

- schema_conformance_rate
- coverage_score (fraction of required fields present)
- constraint_violation_rate (units/ranges)
- witness_rate (presence of calc/refs/test)
- delta CE_fresh and ECE pre/post structure-first templates

Minimal template (copy-safe)

Task: <one line>

Required format: JSON schema = <schema>

Constraints: <units, ranges, rounding, allowed_nulls>

Steps:

1) Parse inputs and restate assumptions.

2) If numbers/dates/IDs appear, call calculator/retrieval.

3) Validate against constraints; if a field is unknown, set null.

4) Emit only JSON; no prose.

Return: final JSON, then a short witness with formula/inputs/result if applicable.

8.3 Retrieval, calculators, and small checkers as variance reducers

Tiny tools at decode-time act like noise cancellers; they reduce output variance and calibration error cheaply.

A) Retrieval (RAG-lite)

- Use for named entities, statutes, dates, structured facts.
- Policy:
    - Trigger on NE detection or missing context.
    - Restrict to 3–5 snippets; require citation ids/years in the witness.

- Verifier:
    - entity_overlap := overlap between entities in y and retrieved snippets.
    - citation_valid := URLs/IDs resolvable; years plausible.
- Impact:
    - Reduces hallucinations; improves CE_fresh on fact slices; raises horizon_H where novelty_rate is high.

B) Calculators (numeric/unit ops)

- Use for arithmetic, rates, compounding, unit conversion.
- Pattern:
    - Extract quantities with units; normalize; compute with a library; round per constraint.
    - Emit formula string and intermediate values.
- Verifier:
    - unit_dim_check (e.g., [kg]*[m]/[s]^2 → N)
    - recompute result from emitted formula/inputs
- Impact:
    - Sharp drop in numeric_error_dist; ECE improves as the model commits to numbers with checks.

C) Small checkers (schema/tests)

- JSON/CSV schema validator
- Regex guards for IDs/dates
- Little property tests for code answers (run on toy inputs)
- Range/monotonicity checks (e.g., totals >= 0; sum of parts = total within tol)

D) Selection/rerank with witnesses

- For pass@k candidates, compute a score s_i from:
    - schema_valid ∈ {0,1}
    - coverage_score ∈ [0,1]

    - witness_score (calc present and passes, citations valid) ∈ [0,1]
    - constraint_penalty (violations) ≥ 0
    - s_i := w1*schema_valid + w2*coverage_score + w3*witness_score − w4*constraint_penalty
- Return argmax s_i.

E) Cost discipline

- Log energy_infer (J) per 1k generations with tools.
- Compute eta_infer := delta_bits_fresh / energy_infer
    - delta_bits_fresh from CE_fresh reduction when tools are enabled vs disabled.
- Keep tools that yield high eta_infer; demote those with low gain.

KPIs (rolling)

- retrieval_hit_rate, citation_valid_rate
- calc_invocation_rate, calc_pass_rate
- schema_valid_rate, test_pass_rate
- numeric_error_dist (MAE/MAPE)
- ECE before/after tools
- horizon_gain_tools := H_with_tools − H_without

Takeaway.

- Raise SNR upstream (dedup, normalization), constrain form and constraints in prompts, and let small tools do the heavy lifting on facts and math.
- Track simple proxies (dup_rate, n-gram entropy, tool_hit_rate) and prefer interventions that jointly reduce CE_fresh, improve ECE, and extend forecast_horizon_H at minimal energy.

9) Calibration and Robustness

9.1 Reliability curves, ECE, and selective generation

Goal. Make stated confidence match empirical accuracy, and only answer when confidence is sufficient.

Reliability curve (binning recipe).

- For each generation, produce a scalar confidence $c \in [0,1]$ (e.g., max token prob for a decisive token, verifier score mapped to [0,1], or a calibrated head).
- Bin predictions into B equal-width bins, $I_b = ((b-1)/B, b/B]$.
- For bin b:
  $conf_b := mean(c_i \mid c_i \in I_b)$
  $acc_b := mean(1\{ correct_i \} \mid c_i \in I_b)$
- Plot $(conf_b, acc_b)$; perfect calibration lies on $y=x$.

Expected Calibration Error (ECE).

- $ECE := \sum_{b=1..B} w_b * | acc_b - conf_b |$, where $w_b = n_b / N$.
- Also track **MCE** (max bin gap) for worst-bin miscalibration.
- For numeric tasks, pair with **Brier score** and **NLL**.

Calibrating.

- Post-hoc scaling on a held-out set:
  - temperature scaling for logits,
  - isotonic regression for non-parametric mapping,
  - Platt scaling for binary verifiers.
- Re-estimate per **task type** (fact/code/plan/table) and **slice**; do not assume one map fits all.

Selective generation (answer only when calibrated).

- Define a coverage–risk curve:
  - For threshold τ, answer if $c \geq \tau$; otherwise abstain or fallback.
  - $Coverage(\tau) := P(c \geq \tau)$; $Risk(\tau) :=$ error rate conditioned on $c \geq \tau$.

- Choose τ to meet target risk (e.g., ≤ 5%). Track **AURC** (area under risk–coverage) as a summary: lower is better.
- At decode time, couple confidence with verification:
    - c := f(logit_confidence, schema_valid, witness_pass) → calibrated to [0,1].
    - Enforce τ per **impact tier** (higher τ for high-risk tasks).

KPIs (rolling): ECE, MCE, Brier, NLL, AURC, selected τ per task/tier, abstain_rate, escalation_rate, post-abstention success rate.

9.2 Slice-aware evaluation and worst-case reporting

Why. Averages hide failure pockets. Manage for the tail.

Define slices explicitly (time/source/geo/lang/task_type/risk). Maintain a stable slice catalog S.

Per-slice metrics.

- E_slice(s): primary error (CE_fresh, F1, MAE, …)
- Cal_slice(s): ECE on s
- H_slice(s): forecast_horizon on s
- Vol_slice(s): variance of error across windows on s

Worst-case reporting.

- worst_gap := max_{s∈S} | E_slice(s) − E_global |
- q05_error := 95th-percentile worst slice error across S (or 5th-percentile accuracy)
- Report **(global, worst_slice, q05)** each window; optimize not just the mean but the worst 5–10%.

Stress evaluations.

- **Time shift:** last_7d vs last_30d vs last_90d.
- **Domain shift:** new entities/APIs/laws flagged by novelty_rate.
- **Adversarial templates:** prompt patterns known to raise error.
- **Low-resource slices:** under-represented languages or venues.

Remediation loop.

1. Identify top-k failing slices by worst_gap and Cal_slice.
2. Apply minimal lever: structure-first prompting, tools, adapters, or retrieval targeted at those slices.
3. Re-measure H_slice and ECE; record **horizon_gain_slice** and **delta ECE_slice**.

KPIs (rolling): worst_gap, q05_error, max ECE_slice, horizon_min := min_s H_slice(s), actions_taken_on_slices, gains_post_action.

9.3 Simple abstention and fallback rules

Aim. Reduce unforced errors with cheap, deterministic policies.

Abstention triggers (any fires ⇒ no answer yet).

- confidence $c < \tau\_task$ (calibrated threshold)
- schema_valid == 0 (JSON/CSV/table invalid)
- witness_pass == 0 (calc fails, citation invalid, unit check fails)
- ambiguity_flag == 1 (detector finds unresolved ambiguity)
- novelty_rate_local > τ_novel (many new entities w/o retrieval)
- risk_tier == HIGH and ($c < \tau\_high$ or verification_pass < full)

Fallback ladder (lowest cost first).

1. **Constrained re-decode:** lower T/top-p; enforce schema; re-try once.
2. **Tool augment:** enable retrieval/calculator; re-decode with witness.
3. **Assumption branch:** emit k short candidates with explicit assumptions; select via verifier.
4. **Escalate:** structured clarification question to user (one line, pointed), or hand off to human with a compact “witness packet” (inputs, steps tried, failures).

Formatting for abstain/escalate (copy-safe).

- Final answer: “ABSTAIN”
- Reason codes (comma-delimited): low_confidence, schema_fail, calc_fail, citation_invalid, ambiguity, high_risk_insufficient_confidence, novelty_unresolved
- Optional: 1-line clarification if user present.

Selective decoding knobs (safe defaults).

- High-impact: $T \in [0.1, 0.4]$, top-p $\in [0.6, 0.9]$, constrained to schema; pass@k with verifier (k=3).
- Low-impact/creative: higher T/top-p, but still verify easy constraints (units, IDs) when present.

What to log.

- abstain_rate (overall and per slice), reasons histogram
- fallback_success_rate by ladder step
- post-abstention accuracy (quality when we *do* answer)
- delta_ECE and delta_AURC after enabling abstention
- user_impact: clarification_accept_rate, time_to_resolution

Rules of thumb.

- Tune τ to hit target AURC while keeping abstain_rate acceptable per task tier.
- Never suppress witness failures: abstain rather than pass through.
- If worst-slice AURC remains high, add slice-specific adapters or retrieval; if that fails, refresh.

Takeaway. Calibrate first, answer selectively at fixed risk, and report tails—not just means. Pair abstention with cheap fallbacks (schema, tools, small verifiers). This tightens reliability curves, lowers ECE—especially on bad slices—and lengthens usable horizon without touching model weights.

10) The Operations Dashboard (Methods)

10.1 Minimal metrics to log each window (H, half-life, ECE, SNR)

Windowing. Use rolling windows W of width w (e.g., 24h or 10k prompts). For each window and each defined slice s, log:

Core horizon & drift

- H_task,s := forecast_horizon for task and slice (Sec. 6.2)
- CE_fresh,s := cross-entropy on fresh eval in W
- CE_gap,s := CE_fresh,s − CE_train_on_ref
- drift_slope,s := d/dt CE_fresh,s (robust fit across last k windows)
- drift_half_life_s := ln(2) / max(drift_slope,s, eps)

Calibration

- ECE_s, MCE_s (binning B=10 or 15)
- AURC_s (area under risk–coverage with calibrated c)

SNR proxies

- dup_rate_s (near-duplicate fraction)
- ngram_entropy_s (H_1, H_2, H_3 after normalization)
- tool_hit_rate_s (calls when indicated) and tool_pass_rate_s (calls that passed checks)
- schema_conformance_rate_s, witness_rate_s, verification_success_rate_s

Shift detectors

- KL_input_s := KL(p_live||p_train) on token/features
- novelty_rate_s := new IDs/NE per 1k tokens
- worst_gap := max_s | CE_fresh,s − CE_fresh,global |

Cost & efficiency

- energy_train_window (J), energy_infer_window (J)
- delta_bits_ref := (H_base − H_theta)*tokens_ref (on fixed R)
- eta_bits_per_joule_train := delta_bits_ref / energy_train_window

- eta_infer := delta_bits_fresh / energy_infer_window (tools on vs off)

Recommended schema (CSV/JSON)

ts_start, ts_end, slice, task, CE_fresh, CE_gap, drift_slope, drift_half_life,

H, ECE, MCE, AURC, dup_rate, H1, H2, H3, tool_hit_rate, tool_pass_rate,

schema_conformance, witness_rate, verification_success_rate,

KL_input, novelty_rate, worst_gap,

energy_train_J, energy_infer_J, delta_bits_ref, eta_train, eta_infer, actions_taken

One example row

2025-09-18T00:00Z,2025-09-18T24:00Z,news_en,last_7d_fact,

2.92,0.37,0.011,63h, 5.0d, 0.043,0.12,0.082,

0.06,6.21,3.91,2.77,0.74,0.66,0.91,0.88,

0.21,0.37,0.51, 0.0, 1.6e11, 3.2e10, 0.20, 0.13, "tools_on,adapter_tune"

10.2 Update policy: refresh, adapter-tune, or tool-augment

Use a simple **action ladder** with guard bands. Evaluate per slice s and task.

Inputs (per slice s)

- H_s, drift_half_life_s, CE_gap_s, KL_input_s, ECE_s, AURC_s
- eta_train (expected), eta_infer (observed)
- novelty_rate_s, tool_hit_rate_s, verification_success_rate_s

Decision table (default thresholds; tune per org)

Condition (any triggers)	Preferred action	Rationale
H_s < H_min OR AURC_s > τ_AURC	Enable tools & structure; lower T/top-p; constrained decoding; require witness	Fast horizon gain via variance reduction
verification_success_rate_s < τ_verif	Strengthen verification; block actuation on fails; add calculator/retrieval	Reduce unforced errors; improve calibration
KL_input_s > τ_KL AND CE_gap_s rising	Tool-augment (retrieval); light adapter-tune on fresh slice	Input shift; cheap correction first
drift_half_life_s < τ_half (fast drift)	Adapter-tune targeted on fresh traces	Restore H quickly with high eta
worst_gap large (tail failing)	Slice-specific tools/adapters; reweight eval	Lift tail without touching global
ECE_s > τ_ECE while CE_fresh ok	Post-hoc calibration head per task/slice	Safer selective generation
eta_infer >> eta_train for slice	Prefer tools over training	Better bits/joule at inference
After 2 adapter rounds H_s still < H_min	Partial refresh on curated new data	When local fixes insufficient
After partial refresh H_s still low, CE_gap_s high	Full refresh if projected eta_train above policy floor	Last resort; only if efficient

Pseudocode (copy-safe)

```
for each slice s:
  compute metrics in window W
  if H_s < H_min or AURC_s > tau_AURC:
    enforce_structure(); enable_tools(); tighten_decoding()
  if verification_success_rate_s < tau_verif:
```

```
  require_witness(); block_on_fail()
 if KL_input_s > tau_KL and CE_gap_s rising:
  enable_retrieval(); consider_adapter_tune(s)
 if drift_half_life_s < tau_half:
  adapter_tune(s); measure H_gain
 if worst_gap large:
  target_tail_with_tools_adapters()
 if ECE_s > tau_ECE and CE_fresh stable:
  calibrate_confidence_head(s)
 if repeated_failures after adapters:
  partial_refresh(s)
 if partial_refresh fails and eta_train projected high:
  full_refresh()
 log actions_taken and horizon_gain_after_action
```

Targets (starting points)

- H_min per task (e.g., facts: 3–7 days; code: 1–3 versions; plans: 24–72h)
- τ_half (e.g., 2–4 weeks for gen-purpose; 3–7 days for news)
- τ_KL: 0.10–0.20 on token histograms (domain-dependent)
- τ_ECE: ≤ 5% for fact/code; ≤ 8% for plans
- τ_verif: ≥ 0.85 pass rate on high-impact tasks

Action logging

- actions_taken := ordered list (e.g., ["tools_on","adapter_tune:v5","calibrate_head"])
- horizon_gain := H_post − H_pre (and per slice)
- delta_ECE, delta_CE_fresh, energy_delta, eta_delta
- rollback_conditions pre-declared (e.g., if CE_fresh worsens by > 0.1 for 2 windows → revert)

10.3 Lightweight A/Bs to validate horizon gains

When you make a change, **validate it** with cheap, fast experiments.

Design patterns

- **Interleaved A/B (online):** Randomly assign prompts to Control (A) vs Treatment (B).
    - A: previous settings (e.g., no tools or old template)
    - B: new policy (e.g., structure-first, tools on, adapter v_k)
- **Shadow eval (offline):** Re-decode a fixed eval set with A and B; compare CE_fresh, ECE, H.
- **Switchback (time-based):** Alternate A/B by hour or day to absorb diurnal effects.

Primary endpoints

- horizon_gain := H_B − H_A (per task/slice)
- delta_CE_fresh := CE_fresh,B − CE_fresh,A (want ≤ 0)
- delta_ECE := ECE_B − ECE_A (want ≤ 0)
- abstain_rate and AURC shifts
- energy_delta and eta_delta (bits/joule change)
- worst_slice_improvement (tail metrics)

Minimal power check (rule-of-thumb)

- Let σ be std. dev. of CE_fresh across windows; to detect a mean drop Δ at 95% confidence, need n ≈ (1.96*σ/Δ)^2 windows per arm.
- Practical shortcut: run until CI for delta_CE_fresh excludes 0 and worst_slice improves (or deteriorates) by ≥ pre-set effect size.

Sequential safety

- Use **gated rollout**:
    1. 10% traffic to B → if delta_CE_fresh ≤ 0 and ECE drop, go to 30%
    2. 30% → if worst_slice improves and no KPI regresses > band, go to 50%
    3. 50% → finalize or revert

- Pre-register **revert rules** (e.g., if MCE increases > 2% absolute, auto-revert).

Example A/B checklist (copy-safe)

Experiment: tools_on_for_facts_v2

Arms: A = no_tools, B = retrieval+calculator+schema

Stratification: by slice {news_en, gov_en, code_en}

Duration: 5 windows of 10k prompts each (per arm)

KPIs: H, CE_fresh, ECE, AURC, worst_gap, abstain_rate, energy_infer_J, eta_infer

Gates: proceed if (H_B − H_A) >= +1.0d AND CE_fresh_B ≤ CE_fresh_A AND ECE_B ≤ ECE_A

Revert if MCE_B − MCE_A > +2% or worst_gap increases > +0.05

Outcome record: delta metrics, decision, next action

Instrumentation tips

- Keep **reference sets** fixed across experiments to compute delta_bits consistently.
- Log **random seeds** and **policy hashes** (prompt templates, decoding knobs, adapter IDs) for reproducibility.
- Store **witness packets** for sampled outputs to audit why one arm won (or lost).

Takeaway.
A tiny, consistent scoreboard per window plus a three-step action ladder and lightweight A/Bs is enough to steer a live system: you detect shift, choose the cheapest lever (tools/structure/adapters), and only escalate to training refresh if horizon H and calibration do not recover—with every step justified by measured bits-per-joule and validated by quick experiments.

11) Case Sketches (Concise)

11.1 Factual Q&A with verification hooks

Goal. Answer fact questions with a short reply and a “witness” that can be checked.

Prompt template (copy-safe).

Task: factual_QA

Question: <one line>

Required format:

1) Final answer (one sentence).

2) Witness block: {entities[], dates[], retrieval_ids[], quote_snippets[], url_or_id[], year_range}

Constraints:

- If uncertain, answer "ABSTAIN".

- Use retrieval for named entities or dates (<= 5 snippets).

- Quotes <= 20 words each.

- Return only the answer + witness block, no prose elsewhere.

Decode & verify loop (pseudo).

```
o <- question
snips <- retrieve(o, k<=5)
y <- decode(o, snips, T=0.3, top_p=0.8, constrained_schema)
check1 <- citation_valid(y.url_or_id)
check2 <- entity_overlap(y.entities, snips)
check3 <- year_range_plausible(y.year_range)
if check1 && check2 && check3
  return y
else
  ABSTAIN or re-decode once with stricter constraints
```

KPIs.

- retrieval_hit_rate, citation_valid_rate
- entity_overlap score (0..1)
- schema_valid_rate, abstain_rate
- CE_fresh on fact slice, ECE, AURC
- horizon_gain_tools (with vs without retrieval)

Example (sketch).

Answer: The treaty entered into force on 1994-01-01.

Witness: {entities:["NAFTA"], dates:["1993","1994-01-01"],

retrieval_ids:["govdoc:USTR-NAFTA-1993","stat:Pub.L.103-182"],

quote_snippets:["entered into force on January 1, 1994"],

url_or_id:["ustr.gov/nafta","congress.gov/plaw/103/pub..."], year_range:"1993–1994"}

11.2 Code generation with calculators and tests

Goal. Produce small, correct code by pairing generation with calculations and micro-tests.

Prompt template.

Task: code_function

Spec: "Given price, quantity, tax_rate (%), return total_with_tax."

Required format:

- Code block (language given).
- Test block: 2–3 asserts with explicit numbers.
- Calc trace: formulas + numeric substitutions for each test.

Constraints:

- No external libs; pure function.
- Round to 2 decimals; currency in USD.

Decode & verify loop (pseudo).

```
y <- decode(o, T=0.2, top_p=0.7)
ok1 <- run_tests(y.code, y.tests)          // sandbox
ok2 <- check_calc_trace(y.calc_trace)      // recompute numbers
ok3 <- schema_valid(y)                     // present all blocks
if ok1 && ok2 && ok3
  return y
```

else

re-decode (pass@k<=3) and choose best by ok-scores; else ABSTAIN

Minimal example (sketch, Python).

```
def total_with_tax(price, qty, tax_rate_pct):
    subtotal = price * qty
    total = subtotal * (1 + tax_rate_pct/100.0)
    return round(total, 2)


# tests
assert total_with_tax(10.00, 3, 8.25) == 32.48  # 30 * 1.0825 = 32.475 -> 32.48
assert total_with_tax(5.99, 2, 0.0)  == 11.98
```

Calc trace:

t1: subtotal=10*3=30; total=30*(1+0.0825)=32.475 -> 32.48

t2: subtotal=5.99*2=11.98; total=11.98*(1+0)=11.98

KPIs.

- pass@k, test_pass_rate, calc_trace_pass_rate
- numeric_error_dist (MAE/MAPE)
- CE_fresh on code slice; ECE
- horizon_gain_tools (with calc+tests vs none)
- latency, tokens, eta_infer

11.3 Planning text with outcome logging

Goal. Generate short plans whose steps can be executed or checked later; log outcomes to close the loop.

Prompt template.

Task: planning

Goal: <one line outcome>

Constraints: budget/time/resources; success metric M; deadline D.

Required format:

- Plan v1: numbered steps (each: owner, inputs, expected_output, due_date)
- Risks: top-3 with mitigations
- Witness: definition of M and baseline M0
- Log schema: CSV header for outcomes [step_id, done(0/1), date, M_after, notes]

Rules:

- Keep plan <= 7 steps; no vague verbs.
- Put "ABSTAIN" if constraints conflict with D.

Decode & verify (pseudo).

y <- decode(o, T=0.4, top_p=0.9, constrained_schema)

ok1 <- steps_have_owners_and_due(y.plan)

ok2 <- verbs_in_allowlist(y.plan) // no "optimize", use specific verbs

ok3 <- metric_defined(y.witness.M) and baseline_present(y.witness.M0)

if !(ok1 && ok2 && ok3): ABSTAIN or re-decode strict

emit y + empty CSV with header

Outcome logging (post-deployment).

- After each step, append row:
  step_id, done, date_iso, M_after, notes
- Weekly, compute:
  - delta_M := M_after − M0
  - plan_completion_rate := done_steps / total_steps
  - slippage := mean(max(0, actual_date − due_date))
  - forecast_bias := planned_delta_M − realized_delta_M

Example (sketch).

Plan v1:

1) Owner: A; Action: "Collect last_30d logs"; Inputs: log_path; Output: logs.csv; Due: 2025-09-25

2) Owner: B; Action: "Compute CE_fresh"; Inputs: logs.csv; Output: ce_report.csv; Due: 2025-09-26

3) Owner: C; Action: "Enable retrieval for facts"; Inputs: policy.yaml; Output: rollout_id; Due: 2025-09-27

Risks:

- Missing logs → Mit: fallback to shadow eval

- Tool errors → Mit: dry-run on sample

- Deadline slip → Mit: split steps smaller

Witness:

- Metric M := CE_fresh on news_en; Baseline M0 := 3.12 nats

Log schema:

step_id,done,date_iso,M_after,notes

KPIs.

- plan_completion_rate, slippage
- realized delta_M vs planned delta_M
- horizon_gain_after_plan := H_post − H_pre
- abstain_rate_on_plans (conflict detection)
- auditability: % steps with filled “expected_output”

Takeaway.

- For facts: short answer + witness you can check.
- For code: generate with tests and calc traces; return only if they pass.
- For plans: force owners, dates, and a measurable metric; log outcomes to learn whether the plan actually extended the forecast horizon.

12) Limitations and Falsifiers

12.1 Where horizon metrics can mislead

A) Proxy mismatch.

- *Symptom:* Forecast_horizon H looks stable, but downstream success degrades.
- *Why:* Chosen error E(k) (e.g., CE_fresh) is loosely coupled to task utility (e.g., factual exact-match, code pass rate).
- *Mitigation:* Pair H with a task-native metric (EM/F1/pass@k/MAE) and track both; re-define H using the tighter metric.

B) Evaluation leakage.

- *Symptom:* H improves after refresh, but gains vanish on truly fresh streams.
- *Why:* Reference/eval sets contaminated by training or prompt templating.
- *Mitigation:* Strict data hygiene (hash overlap checks, time-forward eval), rotating cold-start sets.

C) Selective answering bias.

- *Symptom:* H rises while coverage collapses (model abstains too often).
- *Why:* Risk controls trade coverage for lower risk; H computed only on answered cases.
- *Mitigation:* Report **risk–coverage** curves (AURC) and **coverage-at-H**; set minimum coverage floors.

D) Tool confounding.

- *Symptom:* H jumps after enabling tools, but tools silently fail in production edge cases.
- *Why:* RAG/calculators work on eval but degrade under API drift or latency budgets.
- *Mitigation:* Log **tool_pass_rate**, **latency**, and **eta_infer**; include tool-off stress runs.

E) Slice dilution.

- *Symptom:* Global H stable; critical tail slices deteriorate.
- *Why:* Averages hide worst-case collapse.
- *Mitigation:* Always publish **H_min = min_s H_slice(s)** and **worst_gap** with actions taken.

F) Performativity loop.

- *Symptom:* Forecasts alter behavior; targets move, making H appear worse (or better) for the wrong reason.
- *Mitigation:* Track **L_active vs L_passive** (Sec. 5.2) and **performativity_gap**; use switchbacks/holdouts.

G) Metric inertia.

- *Symptom:* H responds slowly to reality because verification labels arrive late.
- *Mitigation:* Add fast proxies (schema/witness failure rate, novelty_rate) as early-warning signals.

12.2 Counterexamples: regime breaks, sparse novelty

A) Hard regime break (new rules, new physics, new API semantics).

- *What happens:* Past invariants collapse; CE_fresh spikes across slices; verification patterns fail in novel ways.
- *Why it matters:* Compression from past traces no longer predicts tomorrow—H shrinks abruptly.
- *Expected signature:* KL_input ↑↑, novelty_rate ↑, witness_fail_rate ↑, H_min → near-zero.
- *Response:* Admit horizon loss; enable tools; collect interventional fresh data; adapter-tune or partial refresh focused on the new regime.

B) Sparse but decisive novelty (rare events with outsized impact).

- *What happens:* Average CE_fresh barely moves; H looks fine, but catastrophic errors occur on rare prompts (safety/security/legal changes).
- *Expected signature:* Global metrics stable; tail audits or red-team prompts show large errors.
- *Response:* Expand **risk_slice** catalogs; report worst-slice metrics; add guardrails/abstention for those slices irrespective of global H.

C) Label drift without input cues (definitions/ground truth redefined).

- *What happens:* P(y|x) changes but tokens look similar; KL_input low, CE_fresh drifts slowly; calibration collapses.
- *Response:* Monitor **ECE**, not just CE; add human-in-the-loop recalibration or quick adapters keyed to new definitions.

D) Strategic/adversarial prompting.

- *What happens:* Prompt distributions are engineered to bypass structure/witness norms; H computed on standard evals overestimates robustness.
- *Response:* Maintain adversarial templates; include them as official slices; enforce constrained decoding/witness checks by policy, not by user goodwill.

12.3 What findings would refute the compression-horizon thesis

We state crisp outcomes that, if observed persistently across domains, should count against the core claims.

F1. Horizon non-predictivity.

- *Refuter:* Measured forecast_horizon H fails to correlate with downstream task success (accuracy/pass@k/MAE) across multiple tasks and time windows (corr ≈ 0).
- *Implication:* "H as a managerial KPI" lacks decision value; the spacetime pipeline offers no actionable horizon.

F2. Half-life irrelevance.

- *Refuter:* drift_half_life does not improve refresh timing versus naive schedules (e.g., weekly retrains); action decisions guided by half-life yield no better CE_fresh or H than time-based baselines.
- *Implication:* Drift metrics are decorative; they do not govern updates.

F3. Witness inefficacy.

- *Refuter:* Adding verification (calculators/retrieval/schema/tests) fails to reduce calibration error (ECE) or output variance and does not extend H at fixed compute on any evaluated slice.
- *Implication:* "Always-be-witnessing" does not function as an SNR lever.

F4. Thermodynamic disconnect.

- *Refuter:* eta_bits_per_joule does not discriminate pipelines—expensive runs routinely produce no larger delta_bits (on a fixed reference) than cheap ones—nor does higher eta associate with better CE_fresh/H.
- *Implication:* The energy↔information link is not operational; MDL framing gives no budgeting advantage.

F5. Tail blindness persists.

- *Refuter:* Slice-aware reporting plus worst-case actions do not improve H_min or worst_gap over time; tails remain as bad as baseline despite interventions.
- *Implication:* The ops bundle cannot manage distributional tails.

F6. Performativity unmanaged.

- *Refuter:* L_active persistently exceeds L_passive with widening **performativity_gap**, and the proposed guardrails (rate limits, abstention, uncertainty display) fail to reduce oscillations or errors.
- *Implication:* The control-view additions do not stabilize the closed loop.

F7. Non-transfer across domains.

- *Refuter:* The framework works only on one narrow domain (e.g., news facts) and fails to show gains in code, planning, or numerical tasks after fair tuning.
- *Implication:* The "model-agnostic, task-agnostic" claim is overstated.

Protocol note (to keep us honest).

- Pre-register KPIs, thresholds, and revert rules before interventions.
- Keep a public **bits ledger** and **horizon log** (H, H_min, ECE, CE_fresh, eta) with dates and actions.
- If two or more refuters (F1–F7) hold after good-faith adjustments, either revise the theory (new KPIs or mechanisms) or abandon it for the affected domain.

Takeaway.
Horizon metrics are useful only if they predict task success, guide refreshes better than the calendar, and improve under verification/tools with superior bits-per-joule. Clear counterexamples and refuters give us a fast path to falsify or refine the spacetime-compression thesis rather than defending it by metaphor.

13) Related Work (Pointers)

13.1 MDL, prediction-by-compression

The **Minimum Description Length (MDL)** principle frames learning as finding the shortest two-part code for data; our "training = compression" lens follows Rissanen and later expositions by Grünwald and Roos. Amazon+2ir.cwi.nl+2
"**Prediction by compression**" traces to Solomonoff induction and Hutter's AIXI, which link sequence prediction and decision-making to algorithmic probability. We adopt only the operational echo (loss ≈ code length; better codes → better forecasts), not the incomputable ideal. Wikipedia+2SpringerLink+2

Positioning. Our use of cross-entropy as achieved code length (Sec. 3) and the **bits-per-joule** metric (Sec. 7) translates classical compression-prediction ideas into run-book KPIs for modern LLM ops.

13.2 World models, RL-as-inference, control

Neural **world models** compress environment dynamics for planning; Ha & Schmidhuber's line shows learning compact latent simulators and even "dream" rollouts. NeurIPS Papers+3arXiv+3University of Cambridge Computer Lab+3
Reinforcement learning as probabilistic inference recasts control with maximum-entropy objectives, tying policy optimization to inference machinery (and to uncertainty-aware decisions). arXiv+2arXiv+2

Positioning. We borrow the control view (Sec. 5) but focus on **language-conditioned controllers** in human loops: telemetry (performativity gap, rate-limits, abstention) and small-gain-style stability checks—lightweight compared to full RL pipelines.

13.3 Drift, calibration, distribution shift

Concept drift surveys map covariate/label/concept changes and adaptation strategies; we inherit their taxonomy and emphasize time-forward, slice-aware monitoring. ACM Digital Library+1
Under **dataset shift**, uncertainty often degrades; Ovadia et al. benchmarked methods and showed post-hoc calibration alone can fail OOD. We pair calibration with **selective generation** and witnesses. arXiv+2NeurIPS Proceedings+2
Foundational **calibration** work (Guo et al.) motivates our use of reliability curves, ECE/MCE, and temperature scaling—applied per task/slice. arXiv+1

The **dataset shift** handbook provides broader background on P_train→P_test mismatch; we operationalize it via KL-to-train and worst-slice alerts. MIT Press+1
For **risk-coverage trade-offs**, selective prediction (reject option) offers formal targets (coverage at bounded risk); we adopt thresholds and simple verifiers rather than new heads, though SelectiveNet is an end-to-end alternative. arXiv+2arXiv+2
Finally, **underspecification** highlights pipelines that produce many equally good i.i.d. models that diverge in the wild; our horizon/half-life KPIs and worst-slice reporting aim to expose such fragility early. arXiv+2Journal of Machine Learning Research+2

Positioning. Prior work explains *what* drifts and *why* calibration fails; our contribution is a compact **ops bundle**—forecast horizon HHH, drift half-life, KL bands, worst-slice reporting, abstention/fallbacks, and bits-per-joule—that a team can log and act on without changing architectures.

14) Conclusion: Managing the Future Cone

14.1 What to measure, when to refresh

Measure four things every window (per task/slice):

- **H (forecast_horizon):** farthest lead k with error ≤ target_band. Treat H as the north-star KPI.
- **Drift:** CE_fresh, CE_gap, drift_slope, drift_half_life = ln(2)/drift_slope. Early-warning: KL(p_live‖p_train), novelty_rate.
- **Reliability:** ECE/MCE, AURC (risk–coverage). Track abstain_rate and witness/verification_success_rate.
- **Efficiency:** eta_bits_per_joule (train) on a fixed reference set; eta_infer (tools on vs off).

Refresh timing (simple policy):

- **Tool-augment now** if: KL or novelty_rate spikes, CE_gap rising, verification_success_rate dips—prefer retrieval/calculators + constrained decoding first.
- **Adapter-tune this week** if: drift_half_life below threshold (fast drift) or H_s falls below H_min on specific slices despite tools.
- **Partial refresh this cycle** if: two adapter rounds restore < 50% of lost H or tail slices (H_min) remain below target.

- **Full refresh** only if: projected eta_bits_per_joule beats your floor and A/B shows horizon_gain > predefined effect size; otherwise postpone.

Guardrails:

- Always report **(global, worst_slice, H_min)**—act on tails, not just averages.
- Dual score forecasts: **L_active** vs **L_passive**; if performativity_gap widens, tighten rate limits/abstention and add holdout cohorts.
- Pre-register revert rules (any KPI worsens beyond band for 2 windows ⇒ roll back).

14.2 A short checklist teams can adopt tomorrow

Daily (rolling window W):

1. Compute per slice: H, CE_fresh, CE_gap, drift_half_life, ECE/MCE, AURC, KL_to_train, novelty_rate, worst_slice gap.
2. Log verification stats: schema_conformance, witness_rate, verification_success_rate; tool_hit_rate and tool_pass_rate.
3. Publish a one-screen dashboard: top 5 improving/declining slices; H_min; actions_taken; eta_train/eta_infer.

If any trigger fires:
4) **Input shift (KL↑, novelty↑):** enable retrieval; enforce structure (JSON/table); lower T/top-p; require witnesses.
5) **Fast drift (half-life↓) or H < H_min:** adapter-tune on fresh, curated traces for that slice; re-measure H and ECE.
6) **Tail collapse (worst_slice gap↑):** target tail with slice-specific tools/adapters; recalibrate confidence head for that slice.
7) **Calibration off (ECE/AURC↑):** apply post-hoc scaling; raise abstention threshold τ_task; keep coverage floor monitored.

Before any heavy training:
8) Estimate projected **eta_bits_per_joule** using a small pilot; proceed only if eta clears the org floor and expected **horizon_gain** ≥ effect size.

Change validation:
9) Run a **lightweight A/B** (or switchback) for ≥ N windows; gate rollout by: H_gain ≥ target, CE_fresh ≤ baseline, ECE ↓, worst_slice improves, no KPI regresses > band.

10) Record in the **bits ledger**: {date, action, energy_delta, delta_bits, eta, H_gain, notes}. Revert automatically if guardrails trip.

Standing defaults:

- "Always-be-witnessing" on high-impact tasks (calc/cite/test or ABSTAIN).
- Rate-limit action magnitude; add hysteresis on policy thresholds.
- Keep a stable reference set R for delta_bits; rotate a small cold-start set monthly.

One-line takeaway:
Treat AI ops as managing a **future cone**—measure horizon H, watch drift and calibration, fix tails first, and only spend big compute when **bits per joule** and **A/B-verified horizon gains** say it's worth it.

Appendix A: Copy-safe formulas and metric recipes

Notation:

- D: corpus; R: fixed reference set; F: fresh rolling eval set
- CE: cross-entropy in nats unless base specified; use base b if needed
- tokens_S: number of tokens in set S
- Window index i over rolling windows W_i = [t_i, t_i + w)
- Slices s ∈ S (time/source/geo/lang/task_type/risk)

A.1 Cross-entropy and gaps

- CE_theta(S) = (1 / tokens_S) * sum_{x in S} (-log_b p_theta(x))
- CE_gap_s = CE_fresh_s − CE_train_on_ref
- Delta bits on reference:
  delta_bits_ref = (H_base(R) − H_theta(R)) * tokens_R
  where H_base is baseline code length (unigram, prior ckpt, or gzip/BPE baseline).
- Units: choose b=2 for bits, b=e for nats; keep consistent.

A.2 KL divergence to detect input shift

- Given histograms p_live and p_train over features/tokens:
  KL(p_live || p_train) = sum_i p_live(i) * log_b(p_live(i) / max(p_train(i), eps))
- Use eps > 0 for stability; compute per slice.

A.3 Drift slope and half-life

- Let CE_fresh[i] be CE on F within W_i; fit CE_fresh[i] ≈ a + s * t_i via robust regression.
- drift_slope_s = s_s (per time unit)
- drift_half_life_s = ln(2) / max(drift_slope_s, eps)

A.4 Forecast horizon H

- Pick task metric E(k) (e.g., CE_fresh at lead k, MAE, pass@k error).
- H_s = max k such that E_s(k) ≤ target_band_s
- Report H_global and H_min = min_s H_s.

A.5 Calibration metrics

- Bin confidences c ∈ [0,1] into B bins I_b = ((b−1)/B, b/B].
- conf_b = mean(c_i | c_i ∈ I_b)
- acc_b = mean(1{correct_i} | c_i ∈ I_b)
- ECE = sum_b (n_b / N) * | acc_b − conf_b |
- MCE = max_b | acc_b − conf_b |
- For numeric regression: use Brier score and NLL additionally.

A.6 Risk–coverage (selective generation)

- For threshold tau, answer if c ≥ tau, else abstain.
- Coverage(tau) = P(c ≥ tau)
- Risk(tau) = error rate conditioned on c ≥ tau
- AURC = integral over tau of Risk vs Coverage (approx via trapezoids).

A.7 SNR proxies

- dup_rate = (# near-duplicate docs) / (# docs), within window and source family
- ngram_entropy H_n:
  H_n = − sum_g p(g_n) * log_b p(g_n), after normalization (case, markup, numbers)
- tool_hit_rate = (# generations where tool indicated AND invoked) / (# generations where indicated)
- tool_pass_rate = (# tool invocations that pass checks) / (# tool invocations)

A.8 Verification rates

- schema_conformance_rate = valid_schema / total
- witness_rate = generations with witness block / total
- verification_success_rate = generations passing all checks / total

A.9 Performativity and active vs passive loss

- L_passive = error against counterfactual s_{t+1} under u_passive (holdout/no-actuation cohort)
- L_active = error against observed s_{t+1} under u_t = pi(o_t, y_t)

- performativity_gap = L_active − L_passive

A.10 Bits-per-joule efficiency

- eta_bits_per_joule_train = delta_bits_ref / energy_train
- eta_infer = delta_bits_fresh / energy_infer
  where delta_bits_fresh = (CE_off(F) − CE_on(F)) * tokens_F for a tools-on vs tools-off A/B.

A.11 Worst-slice reporting

- E_slice(s) = chosen error metric on slice s
- worst_gap = max_s | E_slice(s) − E_global |
- H_min = min_s H_s

A.12 Simple power rule for A/B

- Windows per arm n ≈ (1.96 * sigma / Delta)^2
  where sigma = stddev of CE_fresh across windows, Delta = target mean drop.

Appendix B: Reproducible log schema (CSV/JSON)

B.1 CSV header (one row per (window, slice, task))
ts_start,ts_end,slice,task,CE_fresh,CE_gap,drift_slope,drift_half_life,H,ECE,MCE,AURC,dup_rate,H1,H2,H3,tool_hit_rate,tool_pass_rate,schema_conformance,witness_rate,verification_success_rate,KL_input,novelty_rate,worst_gap,energy_train_J,energy_infer_J,delta_bits_ref,eta_train,eta_infer,actions_taken,policy_hash,eval_set_hash,notes

B.2 CSV example row
2025-09-19T00:00Z,2025-09-19T24:00Z,news_en,factual,2.91,0.35,0.010,69h,5.2d,0.041,0.11,0.079,0.05,6.18,3.88,2.73,0.76,0.67,0.93,0.89,0.22,0.34,0.47,1.9e11,3.1e10,3.3e10,0.17,0.11,"tools_on;adapter_v7;calibrate_head","tmpl#c8f1","eval#7a3e","KL spike on two sources"

B.3 JSON (newline-delimited, one object per (window, slice, task))
{
"ts_start": "2025-09-19T00:00:00Z",
"ts_end": "2025-09-19T24:00:00Z",
"slice": "news_en",
"task": "factual",
"metrics": {
"CE_fresh": 2.91,

"CE_gap": 0.35,
"drift_slope": 0.010,
"drift_half_life": "69h",
"H": "5.2d",
"ECE": 0.041,
"MCE": 0.11,
"AURC": 0.079,
"dup_rate": 0.05,
"H1": 6.18, "H2": 3.88, "H3": 2.73,
"tool_hit_rate": 0.76, "tool_pass_rate": 0.67,
"schema_conformance": 0.93, "witness_rate": 0.89, "verification_success_rate": 0.22,
"KL_input": 0.34, "novelty_rate": 0.47,
"worst_gap": 0.21
},
"efficiency": {
"energy_train_J": 1.9e11,
"energy_infer_J": 3.1e10,
"delta_bits_ref": 3.3e10,
"eta_train": 0.17,
"eta_infer": 0.11
},
"actions_taken": ["tools_on","adapter_v7","calibrate_head"],
"policy_hash": "tmpl#c8f1",
"eval_set_hash": "eval#7a3e",
"notes": "KL spike on two sources"
}

B.4 Policy hashes (recommended)

- policy_hash: stable hash of prompt template, decoding knobs (T, top_p, beam_k), tool policy, verification flags
- eval_set_hash: stable hash of the eval slice IDs for exact reproducibility

B.5 Witness packet (store alongside samples)

- For sampled generations, keep:
  {prompt_id, arm(A|B), y, witness, schema_ok(0/1), calc_pass(0/1), citations_valid(0/1), confidence, reason_codes[]}

Appendix C: Example prompts and verification snippets

C.1 Factual Q&A (with retrieval and citation checks)

Prompt (model-facing):
Task: factual_QA
Question: <one line question here>
Required format:

1. Answer: <one sentence>
2. Witness: {entities[], dates[], retrieval_ids[], quote_snippets[], url_or_id[], year_range}
   Constraints:

- If uncertain, output "ABSTAIN".
- Use retrieval for named entities or dates (<= 5 snippets).
- Each quote_snippet <= 20 words.
- Output exactly two lines: Answer: ..., then Witness: ...

Verifier (pseudo):
snips <- retrieve(question, k<=5)
ok1 <- all_urls_resolve(witness.url_or_id)
ok2 <- entity_overlap(witness.entities, snips) >= tau_entities
ok3 <- year_range_plausible(witness.year_range)
if !(ok1 && ok2 && ok3): flag = "citation_invalid" or "entity_mismatch"
Decision: if flag then ABSTAIN else accept

Reason codes on fail:
low_confidence, citation_invalid, entity_mismatch, year_range_implausible

C.2 Code function (with tests and calc trace)

Prompt:
Task: code_function
Spec: "Given price, qty, tax_rate_pct, return total_with_tax rounded to 2 decimals (USD)."
Required format:

- Code block in Python.
- Test block: 3 asserts with explicit numbers.
- Calc trace: formulas with numeric substitutions for each test.
  Constraints:

- No external libs; pure function.
- If cannot guarantee, output "ABSTAIN".

Verifier (pseudo):
ok_syntax <- run_py(code)
ok_tests <- run_tests(code, asserts)
ok_calc <- recompute_from_trace(calc_trace) == asserted_values (within tol)
ok_schema <- all blocks present
if ok_syntax && ok_tests && ok_calc && ok_schema then accept else ABSTAIN / retry
(pass@k<=3)

C.3 Planning text (owners, dates, measurable M)

Prompt:
Task: planning
Goal: <one line outcome>
Constraints: budget/time/resources; success metric M; deadline D.
Required format:

- Plan v1: 3–7 numbered steps. Each line: step_id, owner, action(verb from allowlist), inputs, expected_output, due_date(YYYY-MM-DD)
- Risks: top-3 with mitigations
- Witness: define M and baseline M0
- Log schema: CSV header "step_id,done(0/1),date_iso,M_after,notes"
  Rules:
- If constraints conflict with D, output "ABSTAIN".
- Use only verbs from allowlist:
  {"collect","compute","enable","deploy","calibrate","validate","report"}.

Verifier (pseudo):
ok1 <- all steps have owner AND due_date
ok2 <- verbs_in_allowlist(steps)
ok3 <- metric_defined(M) AND baseline_present(M0)
Decision: accept if ok1 && ok2 && ok3 else ABSTAIN

C.4 JSON schema and quick validators (copy-paste)

JSON Schema (factual witness):
{

"type":"object",
"properties":{
"entities":{"type":"array","items":{"type":"string"}},
"dates":{"type":"array","items":{"type":"string"}},
"retrieval_ids":{"type":"array","items":{"type":"string"}},
"quote_snippets":{"type":"array","items":{"type":"string"}},
"url_or_id":{"type":"array","items":{"type":"string"}},
"year_range":{"type":"string"}
},
"required":["entities","url_or_id","year_range"]
}

Regex guards (examples):

- ISO date: ^\d{4}-\d{2}-\d{2}$
- Year: ^(19\d\d|20\d\d|21\d\d)$
- DOI: ^10\.\d{4,9}/[-._;()/:A-Z0-9]+$ (case-insensitive)

Unit check (dimensional):

- For force: N == kg*m/s^2
- For currency rounding: round(x, 2) and 0 ≤ cents < 100

C.5 Reason-code schema for abstention (uniform across tasks)
reason_codes := one or more of:
low_confidence, schema_fail, calc_fail, citation_invalid, ambiguity,
high_risk_insufficient_confidence, novelty_unresolved

C.6 Minimal A/B experiment card (template)

Experiment: <name>
Arms: A=<policy_A>, B=<policy_B>
Strata: <list of slices>
Duration: <N windows>
Primary KPIs: H, CE_fresh, ECE, AURC, worst_gap, abstain_rate, energy_infer_J, eta_infer
Gates: proceed if (H_B − H_A) ≥ H_gain_min AND CE_fresh_B ≤ CE_fresh_A AND ECE_B ≤ ECE_A
Revert if MCE_B − MCE_A > mce_band or worst_gap increases > gap_band
Hashes: policy_hash=<...>, eval_set_hash=<...>
Outcome: {delta_H, delta_CE_fresh, delta_ECE, tail_change, decision}

C.7 Bits ledger entry (one line JSON)

{"date":"2025-09-19","action":"adapter_tune:v7","energy_delta_J":8.2e9,"delta_bits_ref":6.1e9,"eta":0.74,"H_gain_days":+1.1,"notes":"targeted news_en slice; KL spike recovered"}

C.8 One-line selective generation policy (per task tier)

- HIGH-IMPACT: if (c < 0.85) or (!witness_pass) or (!schema_ok) -> ABSTAIN; else answer.
- MEDIUM: if (c < 0.70) -> ABSTAIN; else answer with witness.
- LOW: answer; include basic checks (IDs, units) when present.

End of appendices.

References

Barron, A. R., Rissanen, J., & Yu, B. (1998). *The minimum description length principle in coding and modeling*. **IEEE Trans. Info. Theory**. Yale Statistics

Botvinick, M., & Toussaint, M. (2012). *Planning as inference*. **Trends in Cognitive Sciences**. ScienceDirect+1

D'Amour, A., et al. (2020/2022). *Underspecification presents challenges for credibility in modern machine learning*. **arXiv / JMLR**. arXiv+1

Geifman, Y., & El-Yaniv, R. (2017). *Selective classification for deep neural networks*. **NeurIPS**. NeurIPS Papers+1

Geifman, Y., & El-Yaniv, R. (2019). *SelectiveNet: A deep neural network with an integrated reject option*. **ICML (PMLR)**. Proceedings of Machine Learning Research+1

Grünwald, P. D. (2007). *The Minimum Description Length Principle*. **MIT Press**. MIT Press+1

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). *On calibration of modern neural networks*. **ICML (PMLR)**. Proceedings of Machine Learning Research+1

Ha, D., & Schmidhuber, J. (2018). *World Models*. **arXiv**. arXiv+1

Hafner, D., et al. (2019). *Dream to Control: Learning behaviors by latent imagination (Dreamer)*. **arXiv / ICLR**. arXiv+1

Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. **Springer**. hutter1.net+1

Levine, S. (2018). *Reinforcement learning and control as probabilistic inference: Tutorial and review*. **arXiv**. arXiv+1

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). *Learning under concept drift: A review*. **IEEE TKDE**. SciSpace+1

Ovadia, Y., et al. (2019). *Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift*. **NeurIPS**. NeurIPS Proceedings+1

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (Eds.). (2009). *Dataset Shift in Machine Learning*. **MIT Press**. MIT Press+1

Rissanen, J. (1983). *A universal prior for integers and estimation by minimum description length*. **Annals of Statistics** (classic MDL article; accessible scan). Khoury College of Computer Sciences

Solomonoff, R. J. (1964). *A formal theory of inductive inference (Parts I & II)*. **Information and Control**. ScienceDirect+2Ray Solomonoff+2

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). *Characterizing concept drift*. **Data Mining and Knowledge Discovery**. SpringerLink+1

Ha, D., & Schmidhuber, J. (2018). *Recurrent World Models Facilitate Policy Evolution*. **arXiv** (companion). arXiv

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). *A survey on concept drift adaptation*. **ACM Computing Surveys**. ACM Digital Library+1