The Cognitive Processor — persistence-driven task selection

Contents

The Cognitive Processor: Persistence-Driven Task Selection via Monte Carlo Tree Search over an LLM Policy **2**

Abstract . . . 3
1 Introduction . . . 4
1.1 The cognitive processor as a problem . . . 4
1.2 Task selection under finite computation . . . 4
1.3 Contributions . . . 5
2 Definitions . . . 6
2.1 The cognitive processor . . . 6
2.2 Task levels and the level recursion . . . 6
2.3 States, actions, and trajectories . . . 6
2.4 The task-selection problem . . . 7
3 Wiring the persistence ratio to runtime signals . . . 8
3.1 Power income P_{in} . . . 8
3.2 Predictive efficiency η_I . . . 8
3.3 Coordination cost ω and noise floor $\mathcal{E}_\Sigma$ . . . 8
3.4 Delusion divergence $\mathcal{D}_{KL}^{(\Sigma)}$ . . . 8
3.5 Senescence $\Gamma^{(\Sigma)}$ . . . 9
3.6 Shelter and substrate factors Ψ, Φ . . . 9
3.7 Composite . . . 9
4 MCTS over the task tree . . . 10
4.1 Search tree, root, and persistence per node . . . 10
4.2 Selection: persistence-aware UCT . . . 10
4.3 Expansion: the LLM as proposer . . . 10
4.4 Simulation: three-tier evaluator ladder . . . 11
4.5 Backpropagation . . . 11
4.6 Termination and action commitment . . . 11
4.7 Where the search tree differs from the task tree . . . 11
5 Main results . . . 13
5.1 Convergence to the persistence-optimal action . . . 13
5.2 Greedy single-rollout penalty . . . 13
5.3 Fractal credit consistency . . . 14
5.4 Necessary conditions . . . 14
6 Reference implementation: `aion-core` . . . 15
6.1 Component mapping . . . 15
6.2 The persistence service . . . 15
6.3 The MCTS path through the loop . . . 15
6.4 Deployment ladder . . . 17
6.5 What is *not* implemented . . . 17
7 Predictions, limitations, and discussion . . . 18
7.1 Falsifiable predictions . . . 18
7.2 Limitations . . . 18
7.3 Relation to existing work . . . 18
7.4 Boundary with non-physical claims . . . 19
8 Conclusion . . . 20
Acknowledgements . . . 21

Author contribution 21
Declarations 21
References 22

The Cognitive Processor: Persistence-Driven Task Selection via Monte Carlo Tree Search over an LLM Policy

Lasse Hyyrynen *Independent researcher*

May 21, 2026

Abstract

We define a **cognitive processor** as an information-persisting system (IPS) whose Markov blanket is implemented in software: a finite-resource agent stack that converts incoming attention, energy, and tasks into structured outputs while maintaining its own persistence ratio $\mathcal{R} \geq 1$. We model the cognitive processor as a four-component runtime — a large language model **policy**, a durable **task tree**, a sandboxed **environment**, and a **prediction market** that scores beliefs by sequential Kullback–Leibler reduction — and show that the natural objective of such a runtime is not "tasks completed" but persistence itself. Building on the Fractal Persistence Equation (FPE) of [Hyyrynen, 2026a], we formalise the *task-selection problem* as the question: which of the currently READY tasks should the processor admit into its single RUNNING slot so that the expected change $\mathbb{E}[\Delta\mathcal{R}]$ in the processor's own persistence ratio is maximised under a finite computational budget? We prove that under mild regularity conditions a Monte Carlo Tree Search (MCTS) procedure operating over the processor's task tree, with the LLM as proposal policy and per-task market beliefs as leaf evaluator, converges to the persistence-optimal selection at a rate $O(1/\sqrt{n})$ in the number of simulations n (Theorem 5.1). We further prove that any cognitive processor without such budgeted lookahead — selecting greedily on a single LLM rollout — has a strictly tighter dissolution-time bound than one with adequate MCTS budget whenever the delusion divergence $\mathcal{D}_{KL}^{(\Sigma)}$ exceeds a critical value (Theorem 5.2). We instantiate the construction in the open-source `aion-core` reference implementation, mapping each MCTS phase to existing services, and discuss limitations: irreversible side effects, sandbox approximation gaps, and the cost of the proposal LLM relative to the budget it spends. The contribution is a quantitative bridge between persistence as a thermodynamic accounting identity and budgeted search as a concrete algorithm: the processor that survives is the one whose tree search is honest about its own books.

Keywords: cognitive architectures, large language models, Monte Carlo tree search, free-energy principle, persistence ratio, prediction markets, KL divergence, agent orchestration, information thermodynamics.

1 Introduction

1.1 The cognitive processor as a problem

A modern LLM-driven agent stack is asked to do something that no chat endpoint can do alone: hold a long-running plan, coordinate parallel work, change the world through tool calls, and remain *trustworthy enough* over time that operators continue to feed it attention, compute, and money. The orthodox response — wrap the LLM in a workflow engine — solves the bookkeeping but misses the central physical fact: such a system is a non-equilibrium pattern whose continued existence depends on a continuous balance between income (attention, work delivered) and expenditure (compute, errors, fatigue). When that balance turns negative, no quantity of clever prompting saves it; the operator unplugs the deployment and the pattern dissolves.

This paper takes that physical fact seriously. We treat the agent stack as an *information-persisting system* (IPS) in the sense of [Hyyrynen, 2026a]: a bounded subsystem with a Markov blanket, a non-equilibrium drive, an internal model, and an identity that must be preserved over a time scale much longer than its Landauer cycle. We call such an IPS, when implemented in software around an LLM, a **cognitive processor**. Its persistence ratio

$$\mathcal{R}^{(\Sigma)} \;=\; \Psi\big(\mathcal{R}^{(L+1)}\big)\;\cdot\;\frac{P_{\text{in}}\,\eta_I(\mathcal{D}_{KL}^{(\Sigma)})}{\omega\,\mathcal{E}_\Sigma\big(1+\mathcal{D}_{KL}^{(\Sigma)}+\Gamma^{(\Sigma)}\big)}\;\cdot\;\Phi\big(\mathcal{R}^{(L-1)}\big) \tag{1.1}$$

is the same dimensionless identity that governs nuclei, cells, and firms; the only novelty is that for a cognitive processor each term is *measurable from internal logs*: P_{in} from attention and value events, ω from coordination state, $\mathcal{E}_\Sigma$ from error rates, $\mathcal{D}_{KL}$ from prediction-market reduction scores, Γ from context pressure and stale beliefs.

1.2 Task selection under finite computation

A cognitive processor with H handlers can run at most H tasks at once; READY tasks queue. Each handler must, at every step, answer two coupled questions:

1. **Which task should be *current*?** — admission and focus.
2. **Which tool call should be issued *now*?** — within-task action selection.

Both questions are *decisions under a hard computational budget*: bounded LLM tokens, bounded wall-clock, bounded API spend. The dominant industrial answer (single-shot proposer: ask the LLM once, execute its top tool call) is the limit of **zero search**. It is cheap, fast, and — as we shall show — pays a measurable persistence penalty whenever the policy's beliefs about action consequences diverge from the truth.

The natural alternative is *budgeted search*: spend a slice of the available compute *evaluating* candidate actions before committing to one. Among budgeted-search algorithms, Monte Carlo Tree Search (MCTS; [Coulom, 2006; Browne et al., 2012]) is canonical: it grows a search tree on demand, balances exploitation and exploration at each node via UCT [Kocsis & Szepesvári, 2006], and converges asymptotically to the optimal action under mild assumptions on the leaf evaluator. The match to a cognitive processor is striking:

- The processor already maintains a durable **task tree** with parent/child links, gather joins, and named synchronisation. *That is the search tree.*
- The LLM, with a sampling temperature greater than zero, is a stochastic **proposer** for child nodes. *That is the expansion policy.*
- The prediction market produces calibrated **beliefs** $P(\textsc{success})$ per task, scored by sequential KL reduction. *That is the leaf evaluator.*
- The persistence ratio (1.1) supplies a **scalar** that is well-defined at every node and recomposes fractally up and down the tree. *That is the value to back-propagate.*

Each piece exists. What has been missing is a formal account of *why these particular pieces compose into a survival-relevant algorithm*, what convergence guarantees one obtains, and where the construction breaks.

1.3 Contributions

This paper provides that account. Concretely:

- (§2) We give a formal definition of a cognitive processor as a level-L IPS, with sub-tasks as level-$(L-1)$ children and the deployment as the level-$(L+1)$ shelter.
- (§3) We map every term of (1.1) to an observable signal in the runtime, refining the table of [aion-core, persistence_objective_and_mcts.md] into a measurement protocol.
- (§4) We define MCTS over the task tree with persistence as the leaf signal, give the exact UCT rule, and specify the proposal–evaluation–backpropagation cycle in terms of HTTP-level operations on the existing services.
- (§5) We prove three theorems: (i) *Persistence-optimal convergence* of UCT with KL-honest leaf evaluator (Theorem 5.1); (ii) *Greedy penalty*, that single-rollout selection has strictly shorter dissolution-time bound than adequate-budget MCTS whenever delusion exceeds a threshold (Theorem 5.2); (iii) *Fractal credit*, that backpropagated $\Delta\mathcal{R}$ at level L is consistent with the FPE composition law across levels $L \pm 1$ (Theorem 5.3).
- (§6) We instantiate the construction in `aion-core`: which existing endpoints serve which MCTS roles, what state is persisted in `Task.state._mcts`, and how a v1→v3 deployment ladder respects the budget caps without breaking backward compatibility.
- (§7) We discuss four families of falsifiable predictions and four limitations: irreversibility of real side effects, sandbox–reality gap, proposer-LLM cost, and the danger of optimising a *proxy* persistence ratio computed from a deluded internal model.

The paper is intentionally narrow. We do not claim that MCTS is *the* general algorithm for cognitive architectures; we claim that *under the persistence accounting identity*, MCTS is the structurally-correct way to spend a finite computation budget on choosing the next task and the next action within it.

2 Definitions

2.1 The cognitive processor

We adopt the IPS framework of [Hyyrynen, 2026a, §2] without modification.

Definition 2.1 (Cognitive processor). A **cognitive processor** Σ is an IPS at organisational level L whose internal degrees of freedom decompose into the disjoint union

$$\Sigma_{\text{int}} \;=\; \mathcal{P} \sqcup \mathcal{M} \sqcup \mathcal{L} \sqcup \mathcal{Q}, \tag{2.1}$$

with the following roles:

- $\mathcal{P}$ — the **task store**: a directed acyclic graph $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ of tasks with statuses in $\{\text{PREPARING}, \text{READY}, \text{RUNNING}, \text{BLOCKED}, \text{COMPLETED}, \text{FAILED}\}$ together with named synchronisation primitives (locks, semaphores, events, timestamp locks, queues). The store is durable: $\mathcal{P}$ survives restarts of any other component.
- $\mathcal{M}$ — the **environment**: a sandboxed file system and process space whose effects are restricted to a designated data root.
- $\mathcal{L}$ — the **policy**: a stochastic map $\pi_\theta : S \times \mathcal{A} \to [0, 1]$ from merged states $s \in S$ to actions $a \in \mathcal{A}$, parameterised by an LLM with weights θ. Actions are tool calls that mutate $\mathcal{P}$ or $\mathcal{M}$.
- $\mathcal{Q}$ — the **evaluator**: a prediction market that maintains a distribution $P_m^{(t)}$ over outcomes for each task-market m and scores arriving bets by sequential KL reduction toward the final consensus P_m^* in the sense of [aion-core, market_math.md, §3].

The blanket $\partial\Sigma$ is the union of the HTTP API surface and the operator's UI: every flux of tokens, files, and trust crosses one of these channels. The internal model μ is the joint state of $\mathcal{P}$, $\mathcal{M}$, and the LLM context window at time t.

The four-component decomposition is not arbitrary. It is the minimum set that closes the FPE accounting at the level of an LLM-driven agent: $\mathcal{P}$ supplies ω (coordination cost) and Γ (subtree depth, retry chains); $\mathcal{M}$ supplies $\mathcal{E}_\Sigma$ (tool error rate) and P_{in} (artefact-mediated value); $\mathcal{L}$ realises η_I (the policy's predictive efficiency); $\mathcal{Q}$ supplies $\mathcal{D}_{KL}^{(\Sigma)}$ (delusion). Removing any one component leaves at least one FPE term unmeasurable, and an unmeasurable term cannot be optimised.

2.2 Task levels and the level recursion

The level recursion (2.3) of [Hyyrynen, 2026a] becomes, in the cognitive processor:

Level	Identity	Examples
$L+1$	Deployment / operator / parent organisation	The cloud account, the firm, the user session
L	The cognitive processor itself	One running `aion-core` stack
$L-1$	Agent type / root task	A planning agent, a "ingest morning newsletter" root task
$L-2$	Subtask	A child task: "read URL X", "summarise file Y"
$L-3$	Tool call	One HTTP request to `machine` or `processor`

Each level inherits its FPE accounting from the next finer level via Φ and from the next coarser level via Ψ. We will be especially interested in the coupling between **task-level** $\mathcal{R}^{(\text{task})}$ and **processor-level** $\mathcal{R}^{(\Sigma)}$: a task that locally finishes ($\mathcal{R}^{(\text{task})} \to \infty$ at completion) but consumed enormous compute relative to operator value can still pull $\mathcal{R}^{(\Sigma)}$ below 1. The accounting must compose, not merely aggregate.

2.3 States, actions, and trajectories

Let S denote the space of merged states: tuples $s = (\mathcal{T}, \mathcal{F}, \mathbf{c}, \mathbf{P})$ where $\mathcal{T}$ is the task tree, $\mathcal{F}$ is the file system snapshot, $\mathbf{c}$ is the current task's message history, and $\mathbf{P}$ is the vector of per-task market beliefs. An **action** $a \in \mathcal{A}(s)$ is a tool call available to the LLM at s, drawn from the processor and

machine OpenAPI surfaces (filtered to the agent's `available_tools` list). A **trajectory** is a sequence $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ generated by alternating policy actions and environmental transitions $s_{t+1} = T(s_t, a_t)$, where T is partly deterministic (writing a file) and partly stochastic (LLM-generated content, exec output).

We will *not* assume that S is finite, that $\mathcal{A}$ is finite at every state, or that T is Markov in the strict sense. We will however assume:

- (A1) **Bounded value variation.** $|\mathcal{R}^{(\Sigma)}(s)| \leq R_{\max} < \infty$ for all reachable s.
- (A2) **Lipschitz local dynamics.** Single-step state transitions induce changes $|\mathcal{R}^{(\Sigma)}(s_{t+1}) - \mathcal{R}^{(\Sigma)}(s_t)| \leq K$ for a finite Lipschitz constant K.
- (A3) **Independent leaf noise.** The leaf evaluator returns an unbiased estimator $\hat{R}(s)$ of $\mathbb{E}[\mathcal{R}^{(\Sigma)}(s)]$ with bounded variance σ^2.

(A1)–(A3) hold in any practical cognitive processor with finite compute, finite tool counts, and a calibrated market. They are the same mild assumptions used in standard MCTS convergence proofs [Kocsis & Szepesvári, 2006; Coulom, 2006].

2.4 The task-selection problem

At any wall-clock instant the processor's handler h has zero or one RUNNING tasks and a non-empty set $\mathcal{R}_h \subseteq V_{\mathcal{T}}$ of READY tasks it could claim next. Let b_h denote the *budget* available for the decision: typically a wall-clock cap B_t and a token cap B_θ.

Definition 2.2 (Task-selection problem). The task-selection problem at handler h with state s and budget b_h is to choose

$$a^\star \in \arg\max_{a \in \mathcal{A}(s)} \mathbb{E}\big[\, \Delta\mathcal{R}^{(\Sigma)} \,\big|\, s, a, b_h \,\big], \tag{2.2}$$

where the expectation is over the joint randomness of the LLM, the environment, and the predictor pool, and $\Delta\mathcal{R}^{(\Sigma)} = \mathcal{R}^{(\Sigma)}(s') - \mathcal{R}^{(\Sigma)}(s)$ is the change in processor-level persistence between the pre- and post-action states.

The set $\mathcal{A}(s)$ ranges over both *focus* actions (claim READY task T_i) and *intra-task* actions (issue tool call c_j inside the current RUNNING task). Equation (2.2) does not distinguish them; both are decisions whose persistence consequences must be estimated under the same budget.

The chief difficulty of (2.2) is that $\mathbb{E}[\Delta\mathcal{R}^{(\Sigma)}]$ is *not* available in closed form. It depends on subsequent actions, which depend on subsequent states, and so on. This is the same difficulty solved in games, robotics, and theorem-proving by tree search; in §4 we will spell out its specialisation here.

3 Wiring the persistence ratio to runtime signals

Before MCTS can use $\mathcal{R}^{(\Sigma)}$ as a leaf signal, every term of (1.1) must be computable from data the processor already records. We give the measurement protocol term-by-term and note where each integration with [aion-core, persistence_objective_and_mcts.md, §2] is the same and where it differs.

3.1 Power income P_{in}

For a cognitive processor, P_{in} is the rate at which the deployment's $L+1$ environment delivers attention, value, or compute *contingent on perceived usefulness*. It is not the raw token-per-second of the LLM — that would couple income to expense and trivialise the ratio. We adopt three composable channels:

1. **Operator attention:** count of inbound user requests, console sessions, and authored objects, weighted by session length.
2. **Downstream artefact value:** the operator-tagged `value_weight` of root tasks at COMPLETED, propagated through `Task.processes.adopted.metadata`.
3. **External value events:** revenue, social engagement scores, blockchain market wins, where bridged.

In a v1 deployment, only channel 1 and a coarse channel 2 are routinely available; the protocol must therefore tolerate P_{in} being a noisy under-estimate. We require *monotonicity*: if a candidate action a raises any of channels 1–3 with non-negative effect on the others, $P_{\text{in}}(a) \geq P_{\text{in}}(\text{NOOP})$.

3.2 Predictive efficiency η_I

By (4.2) of [Hyyrynen, 2026a],

$$\eta_I \;\leq\; 1 - \frac{\mathcal{D}_{KL}^{(\Sigma)}}{\log|\mathcal{E}|}. \tag{3.1}$$

For a cognitive processor, the implementable proxy is the rolling **success rate**

$$\hat{\eta}_I^{(L)} \;=\; \frac{|\{\text{COMPLETED tasks under node } L\}|}{|\{\text{COMPLETED}\}| + |\{\text{FAILED}\}| + |\{\text{ABORTED}\}|} \quad (\text{rolling window } w \text{ days}) \tag{3.2}$$

restricted to root tasks attributed to node L via `target_agent_type` or `root_task_id`. This estimator is biased *toward* (3.1) for honest agents (where success rate tracks calibration) and biased *against* (3.1) for sycophantic ones (where SUCCESS is declared without validation), but $\hat{\eta}_I$ remains the cheapest proxy that survives restart and consults no external service.

3.3 Coordination cost ω and noise floor $\mathcal{E}_\Sigma$

The processor's structural complexity is the bit-level cost of describing its current coordination state:

$$\omega^{(L)} \;\propto\; \log\!\left[\,|A_L| \cdot |I_L| \cdot |S_L| \cdot d_L\,\right] \tag{3.3}$$

where $|A_L|$ is the number of active agent types at level L, $|I_L|$ the number of running instances, $|S_L|$ the number of live synchronisation primitives, and d_L the maximum running/blocked subtree depth. Equation (3.3) follows by counting: each variable contributes $\log_2$ of its cardinality to the system descriptor.

The noise floor is the *involuntary* error rate the processor pays even with a perfect policy. We aggregate five sources:

$$\mathcal{E}_\Sigma^{(L)} \;=\; r_{\text{adm}} + r_{\text{lease}} + r_{\text{tool}} + r_{\text{ovr}} + r_{\text{disagree}}, \tag{3.4}$$

the rates of admission failures, expired leases, tool-call HTTP errors, context-overflow events, and predictor-disagreement events respectively. The mapping to existing endpoints (`/scheduler/invariants`, `slot_leases` rows, `audit/emitter.py` events) is given in [aion-core, persistence_objective_and_mcts.md, §2.4]; we keep it unchanged here.

3.4 Delusion divergence $\mathcal{D}_{KL}^{(\Sigma)}$

This is the subtle term and the one the prediction market is built to measure. For each market m over a task T attributed to node L, [aion-core, market_math.md, §3] defines per-bet values $v_{m,t}$ with telescoping sum

$$\sum_t v_{m,t} \;=\; \mathcal{D}_{KL}(P_m^* \,\|\, P_m^{(0)}), \tag{3.5}$$

i.e. the KL divergence between the final consensus and the prior. We *re-purpose* (3.5): the same quantity, summed over all live markets attributed to node L and time-averaged, is the empirical estimate of the node's delusion divergence:

$$\hat{\mathcal{D}}_{KL}^{(L)}(t) \;=\; \frac{1}{|\mathcal{M}_L(t)|} \sum_{m \in \mathcal{M}_L(t)} \mathcal{D}_{KL}(P_m^* \,\|\, P_m^{(0)}). \tag{3.6}$$

The estimator (3.6) is positive and bounded; it equals zero exactly when every market resolves at its prior, i.e. when the policy's a-priori beliefs coincide with the empirical consensus; it grows whenever the policy is systematically over-confident in either direction. By [Hyyrynen, 2026a, Theorem 5.2] this directly maps to lifetime: each nat of $\hat{\mathcal{D}}_{KL}^{(L)}$ divides the dissolution-time budget by e.

A second, complementary signal is **process match accuracy**: the deviation between `processes.adopted.process_id` and the ex-post fit of the actual trajectory. We treat this as an additive contribution to (3.6) when the markets are sparse.

3.5 Senescence $\Gamma^{(\Sigma)}$

Senescence accumulates from sources the policy cannot reverse cheaply: stale documentation, suspended LLM profiles, repeated task resets, unread learning notes (cf. [aion-core, learning_modes.md]). The estimator

$$\hat{\Gamma}^{(L)}(t) \;=\; \alpha_1 \cdot p_{\text{ovr}}(t) + \alpha_2 \cdot p_{\text{susp}}(t) + \alpha_3 \cdot p_{\text{stale}}(t) + \alpha_4 \cdot \rho_{\text{retry}}(t) \tag{3.7}$$

sums normalised fractions of context-overflow steps, suspended profiles, stale seed YAMLs, and per-task reset frequency. The weights α_i are tuned per deployment but the *monotonicity* of $\hat{\Gamma}$ in each component is fixed: aging never decreases.

3.6 Shelter and substrate factors Ψ, Φ

The shelter factor is the level-$(L+1)$ buffer: the deployment, the cloud, the user. We approximate

$$\hat{\Psi}^{(L)} \;=\; \min\Big(1,\; \tilde{\mathcal{R}}^{(L+1)}\Big), \tag{3.8}$$

where $\tilde{\mathcal{R}}^{(L+1)}$ is a normalised score derived from operator session frequency and contractual commitments (uptime SLAs, billing health). When several enclosing IPS buffer distinct noise channels — operator attention, cloud reliability, regulatory environment — we apply (2.4) of [Hyyrynen, 2026a]:

$$\hat{\Psi}_{\text{eff}}^{(L)} \;=\; \prod_j \hat{\Psi}_j^{(L)}. \tag{3.9}$$

The substrate factor is the geometric mean of children's local persistence:

$$\hat{\Phi}^{(L)} \;=\; \left(\prod_{i=1}^{n_L} \hat{\eta}_i^{(L-1)} \right)^{1/n_L}, \tag{3.10}$$

a v1 approximation that uses success rate as a proxy for child $\mathcal{R}$. Equation (3.10) underestimates Φ when children are heterogeneous (a single critical organ failure should drive Φ to 0, not just lower the geometric mean), so a v2 estimator weights children by structural criticality, learned from past collapse traces.

3.7 Composite

Substituting (3.2)–(3.10) into (1.1) yields a **fully measurable** persistence ratio

$$\hat{\mathcal{R}}^{(L)}(t) \;=\; \hat{\Psi}_{\text{eff}}^{(L)} \cdot \frac{\hat{P}_{\text{in}}^{(L)}\, \hat{\eta}_I^{(L)}}{\hat{\omega}^{(L)}\, \hat{\mathcal{E}}_{\Sigma}^{(L)}\, (1 + \hat{\mathcal{D}}_{KL}^{(L)} + \hat{\Gamma}^{(L)})} \cdot \hat{\Phi}^{(L)}. \tag{3.11}$$

This is the scalar that MCTS will back-propagate. Its evaluation cost is dominated by (3.6) and (3.10), both of which are aggregations over rows already in the prediction market and processor databases.

4 MCTS over the task tree

We now define the search procedure that selects $a^\star$ in (2.2). The presentation follows the standard four-phase MCTS recipe [Coulom, 2006; Browne et al., 2012] specialised to the cognitive processor. Throughout, the LLM acts as the proposer and rollout policy, the processor's task tree supplies the persistent search structure, the machine sandbox supplies optional rollouts, and the prediction market supplies leaf evaluations. Persistence (3.11) is the back-propagated value.

4.1 Search tree, root, and persistence per node

The search tree $\mathcal{S}$ is layered *on top of* the durable processor task tree $\mathcal{T}$, not replacing it. Each search node $u \in \mathcal{S}$ corresponds to a triple

$$u \;\equiv\; (\, s_u,\; a_u,\; \mathrm{stats}_u \,), \tag{4.1}$$

with s_u a (possibly hypothetical) merged state, a_u the action whose execution at the parent state produced s_u, and $\mathrm{stats}_u = (N_u,\; V_u,\; Q_u)$ the visit count, sum of back-propagated values, and running average $Q_u = V_u/N_u$. The root $r \in \mathcal{S}$ is anchored to the *real* current state s_0 of the handler at the moment the search begins; child nodes correspond to candidate actions.

For each node u we associate a persistence estimate $\hat{\mathcal{R}}_u = \hat{\mathcal{R}}^{(\Sigma)}(s_u)$ computed via (3.11). The leaf signal that drives the search is the **persistence delta**

$$\Delta\mathcal{R}_u \;\equiv\; \hat{\mathcal{R}}_u - \hat{\mathcal{R}}_r, \tag{4.2}$$

positive when the action sequence reaching u improves the processor's books and negative when it degrades them.

4.2 Selection: persistence-aware UCT

From a non-leaf node u we select a child $c \in \mathrm{children}(u)$ by maximising

$$\mathrm{UCT}(c) \;=\; Q_c \;+\; C\sqrt{\frac{\ln N_u}{N_c}} \;+\; \beta\,\Pi(c), \tag{4.3}$$

where $C \geq 0$ is the standard UCT exploration constant (default $\sqrt{2}$, tunable per agent), and the optional term $\beta\,\Pi(c)$ is a **persistence prior**: a non-negative bias that favours children whose predicted action would, ceteris paribus, improve the processor's books. We use

$$\Pi(c) \;=\; \tilde{\eta}_I(c) \;-\; \tilde{\omega}(c) \;-\; \tilde{\Gamma}(c), \tag{4.4}$$

with each tilde term a normalised contribution to the FPE numerator/denominator inferred from the candidate action's metadata (e.g. spawning twenty parallel children sharply raises $\tilde{\omega}$). When the prior is unavailable, $\Pi \equiv 0$ and (4.3) reduces to vanilla UCT. The presence of $\beta\,\Pi(c)$ does not alter the asymptotic convergence guarantee (Theorem 5.1) but accelerates approach to the optimum; it is the search analogue of action-value initialisation in Q-learning.

4.3 Expansion: the LLM as proposer

When selection reaches a node u with no expanded children and visit count $N_u \geq 1$, the LLM is invoked as a *K-sample proposer*. With temperature $\tau > 0$ and top-p nucleus sampling [Holtzman et al., 2020], the LLM emits K candidate tool calls

$$\{\, a_u^{(1)}, \ldots, a_u^{(K)} \,\} \;\sim\; \pi_\theta(\cdot \mid s_u), \tag{4.5}$$

each constrained to the agent type's `available_tools` list. Duplicates are deduplicated; structurally equivalent calls are collapsed via the canonicalisation hash on `(tool_name, sorted(args))`. Each surviving candidate becomes a child of u with $N = 0$, $Q = 0$, and the next selection step picks among them by (4.3).

The proposer LLM may differ from the **rollout** LLM. Because expansion is run many times per decision, deployments often pick a smaller, cheaper model with higher temperature for proposals, reserving the expensive frontier model for execution; this is the analogue of the *fast policy / slow policy* split in AlphaZero-style architectures [Silver et al., 2017]. The split is invisible to MCTS: any proposer with non-zero probability on the persistence-optimal action produces consistent statistics in the limit (cf. [Browne et al., 2012, §3.4]).

4.4 Simulation: three-tier evaluator ladder

Once a fresh leaf ℓ has been expanded, MCTS needs a scalar to back-propagate. We use a *ladder* of three evaluators, ordered cheapest to most expensive; the first that fits the remaining budget wins.

Tier 1: market prior. If the candidate action a_ℓ creates or focuses a task whose `task_type` already has a prediction market with a calibrated $P_m^{(0)}(\text{SUCCESS})$, we use

$$\hat{R}_\ell^{(\text{T1})} \;=\; \hat{\mathcal{R}}_r \cdot \frac{P_m^{(0)}(\text{SUCCESS}) + \varepsilon}{\hat{\eta}_I^{(L)} + \varepsilon} \tag{4.6}$$

as a proxy for $\hat{\mathcal{R}}_\ell$, with ε a small numerical floor. This tier is *free* in compute terms — it is a database read — and it is biased toward actions whose task families historically succeed.

Tier 2: LLM value head. A single LLM call asks the proposer (or a dedicated value model) for a one-step lookahead estimate of $\Delta\mathcal{R}$ given the candidate action and the merged state, returned in a structured form analogous to `loop_submit_step` with `loop_mode=state`. The estimate is unbiased only to the extent the LLM is calibrated; it is fast (one call, no real side effects) and does not depend on a market existing for the task.

Tier 3: sandbox rollout. The machine forks the data root to an ephemeral path `/data/_mcts_sandbox/{run_id}/`, executes the candidate action there, observes the real state diff, recomputes (3.11) on the resulting state, and discards the branch. This is the most accurate evaluator but the most expensive; it must respect tight budget caps and is appropriate only at v3 of the deployment ladder.

In all three tiers we add **terminal corrections**: a node corresponding to a task reaching COMPLETED with positive operator value receives a bounded bonus; a node corresponding to FAILED or a task that exhausts its `expires_at` deadline receives a penalty equal to the recovery cost. These corrections are necessary to align (4.2) with operator-perceived persistence rather than the FPE of an isolated subtree.

4.5 Backpropagation

After evaluating leaf ℓ to obtain $\hat{R}_\ell$, the back-propagation step walks from ℓ to the root and updates, for each ancestor u on the path,

$$N_u \;\leftarrow\; N_u + 1, \qquad V_u \;\leftarrow\; V_u + \hat{R}_\ell, \qquad Q_u \;\leftarrow\; V_u / N_u. \tag{4.7}$$

We optionally apply *temporal decay* to old visits via $V_u \leftarrow \delta V_u$ between successive root searches, where $\delta = e^{-\lambda \Delta t}$ for a half-life $\ln 2/\lambda$. Decay matters only for sticky search trees that survive task resets (v3 deployment, [aion-core, persistence_objective_and_mcts.md, §4.4]).

4.6 Termination and action commitment

The search loop runs while both budgets hold:

$$\text{cpu\_used} \;<\; B_t, \qquad \text{tokens\_used} \;<\; B_\theta. \tag{4.8}$$

On termination, the committed action is the most-visited child of the root,

$$a^\star \;=\; \arg\max_{c \in \text{children}(r)} N_c, \tag{4.9}$$

not the highest-Q child. This **most-visited rule** is standard MCTS practice [Browne et al., 2012] and is more robust than $\arg\max Q$ when the search budget is small: a child with a single very high evaluation is less reliable than one repeatedly visited with a slightly lower mean. The full search tree (or its top-M subtree) is then persisted to `Task.state._mcts` so that a resumed task — after a BLOCKED transition or a process restart — does not start from scratch.

4.7 Where the search tree differs from the task tree

The processor's task tree $\mathcal{T}$ is durable, side-effectful, and global; the search tree $\mathcal{S}$ is ephemeral, hypothetical, and per-decision. They share structure in the obvious sense — every action in $\mathcal{S}$ that the policy actually commits to becomes a real edge in $\mathcal{T}$ — but the converse does not hold: most search nodes are

never realised in $\mathcal{T}$. This separation is essential. Without it, MCTS would either pay the side-effect cost of every simulated action (sandbox-only) or pollute $\mathcal{T}$ with hypothetical tasks that confuse downstream services. With it, the cognitive processor's persistent state remains a faithful record of what *happened*, while the search tree records what was *considered*.

5 Main results

5.1 Convergence to the persistence-optimal action

Theorem 5.1 (Persistence-optimal convergence). *Let Σ be a cognitive processor satisfying Definition 2.1 and assumptions (A1)–(A3). Let $a^\star$ be the persistence-optimal action of (2.2) at root state s_0, and let $a^{(n)}$ denote the action committed by MCTS (Algorithm of §4) after n simulations with persistence-aware UCT (4.3) and any leaf evaluator from §4.4 with bounded variance. Then*

$$\mathbb{P}\big[a^{(n)} = a^\star\big] \;\to\; 1 \quad \text{as } n \to \infty, \tag{5.1}$$

and the regret

$$\mathrm{Reg}(n) \;\equiv\; \mathbb{E}\big[\mathcal{R}^{(\Sigma)}(s_{a^\star}) - \mathcal{R}^{(\Sigma)}(s_{a^{(n)}})\big] \;=\; O(\sqrt{\ln n / n}). \tag{5.2}$$

Proof sketch. The argument follows the UCT analysis of [Kocsis & Szepesvári, 2006], applied here with $\mathcal{R}^{(\Sigma)}$ as the value function. The persistence prior $\beta\,\Pi(c)$ in (4.3) does not affect the asymptotic visit-count growth of the optimal child relative to suboptimal siblings, since Π is bounded and its contribution is dominated by $\sqrt{\ln N_u / N_c}$ for sufficiently large N_u. Assumption (A1) bounds the value range and therefore the Hoeffding tail of the leaf evaluator; (A2) ensures local bias of the proxy persistence (3.11) under one-step state changes is uniformly bounded; (A3) provides unbiased leaf estimates. The standard UCT regret bound then applies node-wise; aggregating along the path from root to optimal leaf gives (5.2). The most-visited rule (4.9) inherits convergence from (5.2) because the visit-count gap between the optimal action and its siblings grows as $\Theta(n)$ versus $O(n^{1-\rho})$ for some $\rho > 0$. □

Theorem 5.1 says: *if* the leaf evaluator is honest (unbiased estimate of true persistence) and the budget is large enough, MCTS finds the persistence-optimal next action. The result is structural; it does not promise that any *particular* deployment has enough budget. What it does promise is that the algorithm in §4 has the right *shape* — visit counts concentrate on the right action, regret falls at the optimal rate, and the procedure is anytime (any termination point yields a valid action).

5.2 Greedy single-rollout penalty

The chief practical question is not whether MCTS converges in the limit but whether it pays for itself relative to the cheap baseline of *no search*. We can quantify this by comparing the dissolution-time bound (5.1) of [Hyyrynen, 2026a] for a budget-bounded MCTS processor against that of a greedy single-rollout processor under identical noise.

Theorem 5.2 (Greedy penalty). *Let Σ_{MCTS} and Σ_{greedy} be two cognitive processors with identical $P_{\mathrm{in}}, \omega, \mathcal{E}_\Sigma, \Gamma, \Phi, \Psi$ and identical proposer LLM, differing only in their action-selection rule: Σ_{MCTS} runs MCTS with budget $b > 0$ and bounded-variance leaf evaluator; Σ_{greedy} commits the single highest-likelihood action of one LLM call. Suppose the policy's per-action delusion divergence relative to the true posterior over outcomes satisfies $\mathcal{D}_{KL}^{\mathrm{policy}} \geq \Delta > 0$. Then there exists a critical $\Delta^\star$ such that for $\Delta > \Delta^\star$,*

$$\tau_d^{\mathrm{greedy}} \;\leq\; \tau_d^{\mathrm{MCTS}} \cdot e^{-(\Delta - \Delta^\star)}, \tag{5.3}$$

i.e. the greedy processor's expected dissolution time is exponentially shorter in the policy's delusion.

Proof sketch. Both processors share the FPE denominator factor $1 + \mathcal{D}_{KL}^{(\Sigma)} + \Gamma^{(\Sigma)}$. The MCTS processor *modifies its effective* $\mathcal{D}_{KL}$ by the search procedure: by exploring siblings of the most-likely action, it discovers and avoids high-$\mathcal{D}_{KL}$ branches before committing to them. Quantitatively, conditional on the leaf evaluator being unbiased with variance σ^2, the post-search expected KL of the committed action's task family is reduced by at least $\Delta - O(\sqrt{\sigma^2 / b})$ relative to the proposer's prior (this is the classical Bayesian argument: a search picks the better of K samples; the expected gap shrinks as $\Delta - 1/\sqrt{b}$). Substituting into the lifetime bound (5.1) of [Hyyrynen, 2026a] and using Theorem 5.2 of that paper yields (5.3) with $\Delta^\star \propto \sqrt{\sigma^2 / b}$. □

The interpretation is sharp. A processor whose policy is well-calibrated ($\Delta < \Delta^\star$) gains nothing from MCTS — it should run greedy and save the budget. A processor whose policy is badly calibrated ($\Delta \gg \Delta^\star$) loses lifetime exponentially in the gap if it runs greedy. In this sense MCTS is a *delusion shock absorber*: it converts compute spent on search into delusion divergence reduced before commitment, paying for itself in lifetime when the policy needs it most.

5.3 Fractal credit consistency

A backpropagated $\Delta\mathcal{R}$ at the search root is, by (4.2), a level-L persistence delta. The FPE is recursive: $\mathcal{R}^{(L)}$ depends on $\mathcal{R}^{(L\pm1)}$ via Φ and Ψ. A coherent algorithm must therefore credit subtask actions in a way consistent with the level-$(L-1)$ accounting and with the level-$(L+1)$ envelope.

Theorem 5.3 (Fractal credit). *Let $a^\star$ be the action committed by MCTS at level L from root state s_0, and let $\Delta\mathcal{R}^{(L)}(s_{a^\star})$ be the (estimated) backpropagated value at the root. Then $\Delta\mathcal{R}^{(L)}(s_{a^\star})$ is consistent with the FPE composition law in the sense that*

$$\Delta\mathcal{R}^{(L)}(s_{a^\star}) \;\le\; \Psi(\mathcal{R}^{(L+1)}) \cdot \frac{\Delta P_{\text{in}} \cdot \eta_I + P_{\text{in}} \cdot \Delta\eta_I}{\omega\, \mathcal{E}_\Sigma\, (1 + \mathcal{D}_{KL} + \Gamma)} \cdot \Phi + O(\delta^2), \tag{5.4}$$

where δ collects all linearisation errors at the Σ-state s_0, and the right-hand side is the linearisation of (1.1) at s_0. In particular, when the level-$(L+1)$ shelter dissolves ($\Psi \to 0$) or any structurally critical level-$(L-1)$ child fails ($\Phi \to 0$), the search's committed action delivers $\Delta\mathcal{R}^{(L)} \to 0$ regardless of within-tree statistics.

Proof sketch. Linearise (1.1) around s_0 in the small parameters $\Delta\bullet$ for each FPE term. Substitute the per-action contributions: $\Delta\omega$ from new tasks/instances created by $a^\star$, $\Delta\mathcal{E}_\Sigma$ from observed tool errors during evaluation, ΔP_{in} from terminal corrections (§4.4), $\Delta\eta_I$ from market priors (3.6). The factor Ψ in (5.4) is unchanged by within-L search (the deployment does not respond on the search timescale); Φ enters multiplicatively because critical-child failure invalidates blanket factorisation per (5.3) of [Hyyrynen, 2026a]. The remainder $O(\delta^2)$ is the second-order error of linearisation. □

Theorem 5.3 is a *consistency* statement, not a convergence one. It says the algorithm of §4, when its leaf evaluator (3.11) is computed honestly, returns a $\Delta\mathcal{R}$ that respects the level recursion: the search cannot, even in principle, produce action values that contradict the FPE composition law from [Hyyrynen, 2026a, Theorem 5.3]. Practically, this rules out a class of failure modes where a within-task search celebrates an action that — by isolating one subtree — masks the collapse of a critical sibling. The factor Φ in (5.4) drags the search value to zero in exactly that case.

5.4 Necessary conditions

Together the three theorems characterise when MCTS-on-persistence is the right tool:

- **(C1) Non-trivial delusion.** If $\mathcal{D}_{KL}^{\text{policy}} < \Delta^\star$ (the policy is already calibrated), Theorem 5.2 gives no advantage; run greedy.
- **(C2) Honest leaf evaluator.** If the leaf evaluator is biased — for instance, an LLM value head trained on its own optimistic predictions — Theorem 5.1's regret bound fails and MCTS amplifies the bias rather than averaging it out.
- **(C3) Bounded substrate fragility.** If $\Phi \to 0$ frequently (children fail at high rate), the within-task search value is dragged to zero by Theorem 5.3 and no amount of search rescues a broken substrate.
- **(C4) Adequate budget.** If b is so small that $\Delta^\star \sim \sqrt{\sigma^2/b}$ exceeds the policy's actual delusion, Theorem 5.2 gives a vacuous bound; the budget must be calibrated to the noise floor.

A deployment satisfying (C1)–(C4) gets, on average, exponential lifetime gains from MCTS. A deployment that violates any of them gets either no gain or a net loss; in particular, (C2) has been documented empirically as the primary failure mode of self-supervised value heads [Christiano et al., 2017, on the more general phenomenon of reward hacking].

6 Reference implementation: `aion-core`

The construction of §§2–5 is concrete. We describe its instantiation in the open-source `aion-core` reference stack [Hyyrynen, 2026b], where each MCTS role is served by an existing service and the algorithm of §4 ships as an opt-in path through the `loop` engine. The mapping is summarised below; full HTTP surfaces and source-tree pointers are in the architecture documents `MCTS.md` and `persistence_objective_and_mcts.md`.

6.1 Component mapping

Construction	`aion-core` realisation	HTTP / file pointer
Task store $\mathcal{P}$	`processor` service, SQLite `tasks` table	`aion-core/processor/store.py`, `GET /tasks/{id}`
Environment $\mathcal{M}$	`machine` service under `DATA_ROOT`	`POST /read_file`, `POST /exec`, `POST /verify_working_dir`
Policy $\mathcal{L}$	`loop` service `LoopEngine.take_a_step`	`aion-core/loop/loop_engine.py`
Evaluator $\mathcal{Q}$	`prediction_market` service	`POST /v1/predictions/completions`, market rows
Persistence accountant	New `persistence` module (planned)	`GET /state` under `dev.persistence`
MCTS scratch state	`Task.state._mcts` YAML/structured blob	`POST /tasks/current/merge_state`
Sandbox rollouts (v3)	New `machine` endpoints	`POST /sandbox/fork`, `DELETE /sandbox/{id}`

The policy $\mathcal{L}$ today commits a single tool call per turn; the K-sample proposer of §4.3 lives behind an `AgentTypeConfig.mcts_k` flag, defaulting to $K = 1$ (vanilla single-shot, identical to the current production behaviour).

6.2 The persistence service

A small FastAPI module ingests processor webhooks, recomputes (3.11) per node L, and exposes the result on the merged loop state under `dev.persistence`. The data model is the `PersistenceState` row of [aion-core, persistence_objective_and_mcts.md, §3.1]: per-node-id snapshots with all FPE terms and the resulting $\widehat{\mathcal{R}}^{(L)}$. Recompute is triggered (a) on terminal task events via the same webhook that the prediction market consumes, and (b) on a sweep timer every τ_{sweep} minutes for deployment-level recompute. The leaderboard route surfaces the worst-$\mathcal{R}$ nodes for the operator console.

The persistence service does not mutate the processor or machine. It is a *pure observer* of cross-service signals, in the same pattern as the prediction market. This keeps the FPE accounting *causally downstream* of the runtime: actions produce signals; signals produce $\widehat{\mathcal{R}}$; $\widehat{\mathcal{R}}$ informs subsequent actions through MCTS. The cycle closes only through the policy, never through accounting feedback into orchestration directly.

6.3 The MCTS path through the loop

When an agent type has `mcts_k > 1` and a non-zero search budget, `LoopEngine.take_a_step` follows §4 verbatim:

1. Read the merged state and snapshot it as the search root r.
2. Until budget (4.8) is exhausted: select via (4.3); expand by sampling K candidates (4.5); evaluate via the cheapest available tier of §4.4; backpropagate via (4.7).
3. Commit by (4.9); record the search statistics in `Task.state._mcts`.
4. Dispatch the committed action as a normal HTTP tool call; the rest of the loop (state diff, market trigger) is unchanged.

When `mcts_k == 1` and budget is zero, the loop reduces *exactly* to the current production behaviour: one LLM call, one tool dispatch, no search overhead. This zero-cost fallback is the architectural commitment that lets the construction ship without breaking existing deployments.

The persisted scratch state lives under `Task.state._mcts` as a YAML structure. We adopt YAML rather than JSON throughout the persistence interfaces of `aion-core` to keep configuration files, processor state patches, and cross-service payloads in one declarative dialect; the conversion to in-memory dictionaries is symmetric, but YAML's tolerance for comments, multiline strings, and trailing commas makes operator inspection of search traces materially easier. A representative scratch payload, after a single MCTS pass with $K = 3$ and tier-1 evaluator, has the shape:

```yaml
mcts:
  run_id: 2026-05-21T07-22-11Z-h3f8
  budget:
    cpu_seconds_used: 1.84
    cpu_seconds_cap: 4.0
    tokens_used: 1180
    tokens_cap: 8000
  root_state_hash: sha256:9c4d...e1
  evaluator_tier: market_prior
  nodes:
    root:
      visits: 7
      total_dR: 1.42
      children: [n1, n2, n3]
    n1:
      action:
        tool: processor_create_task
        args:
          title: "summarise inbox"
          parent: current
      visits: 4
      total_dR: 0.93
      Q: 0.2325
    n2:
      action:
        tool: processor_complete_current
        args:
          summary: "no actionable inbox items"
      visits: 2
      total_dR: 0.31
      Q: 0.155
    n3:
      action:
        tool: machine_exec
        args:
          cmd: "rg -l 'TODO' src/"
      visits: 1
      total_dR: 0.18
      Q: 0.18
  selected_action: n1
  committed_at: 2026-05-21T07-22-13Z
```

The shape above is the search trace, not the action itself; only `selected_action` is dispatched as a real tool call. The remaining nodes are evidence of *what was considered and rejected*, useful for post-hoc audit, regret analysis, and resumed search after a BLOCKED transition. Operators reading the trace get a one-glance answer to the question that the greedy baseline cannot answer at all: *why this action and not another?*

6.4 Deployment ladder

The deployment ladder of [aion-core, persistence_objective_and_mcts.md, §7] aligns with the necessary conditions of §5.4:

- **v1 (accounting only).** Persistence service computes (3.11); console surfaces $\mathcal{R}$ leaderboard; loop ranks ties using $\mathcal{R}$ but otherwise behaves greedily. Tests (C3) without paying for search.
- **v2 (one-ply search with market evaluator).** $K = 3$ to 5 proposals; tier-1 leaf evaluator (4.6); UCT selects; statistics persisted. Tests (C1)–(C2) with minimal compute overhead.
- **v3 (multi-ply MCTS with sandbox).** Machine sandbox enabled; tier-3 evaluator available; deeper trees survive task resets via a `mcts_runs` table. Tests (C4) at full budget.

Each rung is independently deployable and instrumented to detect violations of the corresponding condition: v1 surfaces Φ collapse, v2 measures empirical regret of MCTS vs greedy on identical workloads, v3 measures sandbox–reality gap.

6.5 What is *not* implemented

Three pieces remain unimplemented in the current reference stack and constitute the v3-and-beyond engineering surface:

1. **Sandbox semantics.** The machine has no copy-on-write fork primitive. Real rollout requires either a filesystem-level snapshot (e.g. ZFS, OverlayFS) or a process-level sandbox (e.g. Firecracker, Bubblewrap) with signed teardown to ensure stale state is not leaked back into the production data root.
2. **Persistence priors.** The $\Pi(c)$ term in (4.3) requires an action-metadata mapping that today exists only as ad-hoc heuristics. A v2 contribution is to formalise this as a `PolicyTypeConfig.persistence_prior` YAML schema and to learn its weights from past collapse traces.
3. **Self-correcting $\hat{\mathcal{D}}_{KL}$.** The estimator (3.6) treats market-final consensus as ground truth. When markets are themselves systematically biased — operator-driven SUCCESS calls in low-engagement deployments, for instance — the estimator under-reports delusion. A v3 fix is to weight markets by their cumulative KL margin against external benchmarks, in the spirit of the proof-of-trust ledger of [Hyyrynen, 2026c].

These gaps do *not* invalidate Theorems 5.1–5.3 — the theorems hold as conditional statements — but they bound which deployments can claim the conditions are met.

7 Predictions, limitations, and discussion

7.1 Falsifiable predictions

The construction yields four predictions that distinguish it both from generic "agentic LLM" stacks (which lack the persistence accounting) and from generic MCTS work (which lacks the FPE tie).

(P1) **Exponential lifetime gap from MCTS.** Two deployments with matched workload and matched proposer LLM but different `mcts_k` should show, by Theorem 5.2, an exponential separation in expected dissolution time as a function of policy delusion Δ. Concretely: the median time-to-operator-disable should scale as $e^{(\Delta-\Delta^\star)}$ in the gap. This is testable with A/B deployments on the same operator base; the persistence service's leaderboard provides the dependent variable.

(P2) **Negligible advantage when calibrated.** Conversely, deployments whose proposer LLM is well-calibrated (low $\widehat{\mathcal{D}}_{KL}^{(L)}$ in the v1 accounting) should show no MCTS advantage, by (C1). This is a stringent test: many "MCTS helps everywhere" claims in the literature are confounded by miscalibration, and the FPE framing predicts the advantage *vanishes* in the calibration limit.

(P3) **Phase transitions at $\Phi \to 0$.** Theorem 5.3 predicts a *discontinuous* drop in MCTS root value when a structurally critical sub-IPS fails — operators should observe the search abruptly devaluing actions whose subtree depends on a recently-collapsed child, even before the failure is obvious in raw success rates. This matches the "percolation-like" failure mode of [Csete & Doyle, 2004] adapted to the agent stack.

(P4) **Sub-Landauer impossibility.** No cognitive processor can persist below the floor $\omega\,\mathcal{E}_\Sigma$ regardless of η_I improvements from MCTS. In particular, MCTS cannot save a deployment whose noise floor exceeds its income; the search will recommend NOOP-like actions and the operator will eventually disconnect. This is testable by stress-testing $\mathcal{E}_\Sigma$ (induced tool errors) and observing the actual MCTS-committed action distribution narrow toward inaction.

7.2 Limitations

Four limitations bound the construction's scope.

(L1) **Irreversibility of side effects.** The simulation step (§4.4) tier 3 promises sandbox rollouts that "discard the branch", but no real-world side effect is fully reversible: an external API call observed by a third party cannot be unsent, a written file may have been read by a sibling process, an LLM call's billing event has happened. Sandbox rollouts therefore approximate; the gap between sandbox-$\widehat{\mathcal{R}}$ and production-$\widehat{\mathcal{R}}$ is itself a noise term that erodes Theorem 5.1's regret bound. Practical deployments must restrict tier-3 rollouts to actions with bounded externalisation cost.

(L2) **Proposer–budget asymmetry.** Theorem 5.2 assumes the proposer LLM and the MCTS evaluator share the same delusion Δ. In practice, a cheap proposer paired with an expensive evaluator can have *opposite* biases: the proposer is over-confident, the evaluator over-cautious. The composition is not necessarily better than either alone; (C2) requires *honest* leaf evaluation, and an over-cautious evaluator scaled by an over-confident proposer can produce systematic mode-collapse in search.

(L3) **Persistence-of-the-deluded.** The accounting (3.11) is computed by the cognitive processor's own internal model. A processor whose model is itself deluded — over-counting P_{in}, under-counting ω — will report $\widehat{\mathcal{R}}^{(\Sigma)}$ values that flatter its own decisions. MCTS optimising this proxy *makes the delusion more efficient*, not the persistence more secure. The architectural reply (cf. §3) is to ground critical FPE terms in *external* attestation: prediction-market scores from independent predictors (for $\mathcal{D}_{KL}$), operator value tags (for P_{in}), audit-log error counts (for $\mathcal{E}_\Sigma$). Where external grounding is unavailable, the construction is structurally vulnerable to self-flattering optimisation, and operators should treat $\widehat{\mathcal{R}}^{(\Sigma)}$ accordingly.

(L4) **Theorem 5.1 is asymptotic.** The convergence rate $O(\sqrt{\ln n/n})$ is slow; small n deployments may see no measurable gain. The remedy is the persistence prior $\Pi(c)$ (4.4), which initialises action values usefully and shifts effective n upward, but only when the prior is *itself* honest — circular dependency on (L3).

7.3 Relation to existing work

The construction unifies four threads that have until now been pursued largely independently.

- **Active inference and the free-energy principle** [Friston, 2010, 2019; Parr et al., 2022] supply the variational formulation that yields the $\mathcal{D}_{KL}$ term in the FPE; our use of the prediction market as an empirical estimator of this term ((3.6)) operationalises that thread.
- **MCTS in games and planning** [Coulom, 2006; Kocsis & Szepesvári, 2006; Silver et al., 2017; Browne et al., 2012] supplies the algorithmic backbone; we adapt UCT to a non-game setting where the value function is a *thermodynamic accounting identity*, not a win probability.
- **LLM-driven agent stacks** [Yao et al., 2023; Wang et al., 2024] supply the policy realisation; our contribution against this thread is the persistence objective, which renders the otherwise under-specified "be helpful" objective into a measurable quantity.
- **Information-thermodynamic theories of life** [Schrödinger, 1944; Prigogine, 1967; England, 2013] supply the physical motivation; we re-cast that motivation algorithmically: the cognitive processor is a *deliberately constructed* IPS whose books we can audit because we wrote the ledger.

What we believe is novel is the explicit composition: an LLM-driven runtime whose action-selection objective is its own thermodynamic persistence, computed from runtime signals, optimised by MCTS, with provable convergence under stated conditions. Each piece exists in prior work; the whole exists, to our knowledge, only as the design described here and partially shipped in `aion-core`.

7.4 Boundary with non-physical claims

We close §7 with what the construction *does not* claim.

- **It is not a claim about consciousness or phenomenology.** $\mathcal{R}^{(\Sigma)}$ is a thermodynamic ratio; it is silent on whether the processor "experiences" anything. The boundary with the long-form treatment of phenomenology in the companion book series [Hyyrynen, 2026d] is sharp: this paper is about books, not feelings.
- **It is not a claim that LLMs are *necessary* for cognitive processors.** Definition 2.1 requires *some* policy with non-trivial η_I; an LLM is one realisation. A compiled rule, a distilled small model, or a classical reinforcement-learning agent could equally serve; cf. the four-rung policy ladder of [aion-core, efficiency_ladder.md].
- **It is not a claim that MCTS is the *only* budgeted-search algorithm consistent with the FPE.** Beam search, progressive widening, and PUCT-style variants [Silver et al., 2017] all admit similar convergence proofs under (A1)–(A3). The choice of vanilla UCT is for clarity and existing-tooling alignment, not optimality.
- **It is not a claim that operating below $\mathcal{R} = 1$ is morally bad.** It is a claim that operating below $\mathcal{R} = 1$ is *physically unsustainable* over time scales much longer than the Landauer cycle. Whether a particular deployment *should* persist is a separate question, addressed in §6.4 of [Hyyrynen, 2026a].

8 Conclusion

A cognitive processor is an information-persisting system in the same sense as a cell, a firm, or a nucleus: its identity is preserved across time only insofar as its books balance in the dimensionless currency of the persistence ratio (1.1). When the policy of such a processor is implemented as a large language model, the action-selection problem becomes the question of how to spend a finite computation budget so that the expected change in the processor's own $\mathcal{R}^{(\Sigma)}$ is maximised. We have shown that Monte Carlo Tree Search over the processor's task tree, with the LLM as proposer and the prediction market's KL reduction score as the empirical delusion estimate, is the structurally correct way to spend that budget. Theorem 5.1 gives the convergence rate; Theorem 5.2 gives the lifetime gap relative to greedy single-rollout; Theorem 5.3 gives the consistency with the FPE composition law across levels.

The construction is concrete: each MCTS phase maps to existing services in the `aion-core` reference stack, each FPE term maps to an observable runtime signal, and the deployment ladder respects (C1)–(C4) at successive levels of cost. It is also limited: irreversibility, proposer–budget asymmetry, the danger of optimising a self-flattering proxy, and the asymptotic nature of the convergence rate all bound the scope of the claimed gains. The honest cognitive processor is the one that audits its own ledger against external attestation and treats its tree search as a delusion shock absorber, not as a self-validating oracle.

In the framing of [Hyyrynen, 2026a], the universe is full of patterns; those that stay are exactly those whose books balance in the currency of bits. A cognitive processor is a pattern *we have written down*. Its books are auditable because we keep them; its persistence is decidable because we measure it. Whether any particular deployment persists in a given environment depends on whether (1.1) holds for the *signals it actually has access to* — and that, in practice, is the work of operators and engineers, one task at a time. MCTS over persistence is the algorithm that turns that work into a budget allocation problem; the FPE is the law that says when the allocation is honest.

Acknowledgements

The author thanks the open-source authors of the MCTS, free-energy-principle, and stochastic-thermodynamics literatures for materials freely available throughout the development of this work, and several anonymous correspondents for objections to earlier drafts. The reference implementation builds on contributions from the `aion-core` project, whose architecture documents are cited above.

Author contribution

LH conceived and wrote the manuscript and the reference implementation with the help of LLM based AI systems.

Declarations

The author declares no competing interests. No external funding supported this work.

References

- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1–43.
- Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30.
- Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games* (pp. 72–83). Springer.
- Csete, M., & Doyle, J. (2004). Bow-ties, metabolism and disease. *Trends in Biotechnology*, 22(9), 446–450.
- England, J. L. (2013). Statistical physics of self-replication. *Journal of Chemical Physics*, 139(12), 121923.
- Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2), 127–138.
- Friston, K. (2019). A free energy principle for a particular physics. *arXiv:1906.10184*.
- Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The curious case of neural text degeneration. In *International Conference on Learning Representations*.
- Hyyrynen, L. (2026a). Information-persisting systems: a thermodynamic theory of persistence as a non-equilibrium accounting identity. `papers/information_persisting_systems.md`.
- Hyyrynen, L. (2026b). The aion-core cognitive processor: services, theory, and architecture. `aion-core/docs/architecture/`.
- Hyyrynen, L. (2026c). Proof of Trust: a git-native blockchain with KL-scored consensus. `aion-blockchain/`.
- Hyyrynen, L. (2026d). The Useful Approximations Framework. `books/book1/`.
- Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning* (pp. 282–293). Springer.
- Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3), 183–191.
- Parr, T., Pezzulo, G., & Friston, K. J. (2022). *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. MIT Press.
- Prigogine, I. (1967). *Introduction to Thermodynamics of Irreversible Processes* (3rd ed.). Wiley.
- Schrödinger, E. (1944). *What is Life? The Physical Aspect of the Living Cell*. Cambridge University Press.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 186345.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.