

# Sleep–Wake Orchestration in Hierarchical LLM Cohorts

Karol Kowalczyk

November 2025

## Abstract

This paper introduces a formal architecture for continuous self-optimization in large language model (LLM) ensembles through alternating sleep–wake cycles. Inspired by biological sleep dynamics and grounded in the theory of *adjoint projections on computational hierarchies*, the method organizes a population of models (“cohorts”) into rotating states of **wake** (active inference) and **sleep** (fine-tuning on informational gaps). Each model periodically withdraws from production to retrain on the most informative regions of the problem space—those not effectively covered by peers of similar capacity but solvable by higher-level models. This process emulates how biological systems consolidate sensorimotor predictions through subcortical replay. The architecture is formalized in terms of computational projections, behavioral metrics, and bounded compute budgets (1/3 for tuning, 2/3 for active operation). The resulting system self-organizes toward optimal coverage and energy-efficient reasoning, providing a theoretical and practical foundation for self-maintaining model ecosystems.

**Keywords:** LLM hierarchy, meta-learning, continual learning, adjoint projection, sleep, fine-tuning, behavioral metrics, computational consciousness.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Contributions . . . . .	3
<b>2</b>	<b>Theoretical Background</b>	<b>4</b>
2.1	Projection Hierarchies and Computational Levels . . . . .	4
2.2	Adjoint Duality . . . . .	4
2.3	Behavioral Distance . . . . .	4
<b>3</b>	<b>Cohort Architecture</b>	<b>5</b>
3.1	Core Components . . . . .	5
3.2	Selector . . . . .	5
3.3	Meta-Selector . . . . .	5
<b>4</b>	<b>Sleep–Wake Dynamics</b>	<b>6</b>
4.1	Resource Allocation . . . . .	6
4.2	Sleep Cycle . . . . .	6
4.3	Gap Misalignment Score . . . . .	6
<b>5</b>	<b>Gap Metric and Learning Objective</b>	<b>7</b>
5.1	Gap Definition . . . . .	7
5.2	Fine-Tuning Objective . . . . .	7
5.3	Training Algorithm . . . . .	7

<b>6 Resource Allocation Model</b>	<b>8</b>
6.1 Envelope-Based Allocation . . . . .	8
6.2 Multi-Envelope Orchestration . . . . .	8
6.3 Adaptive Slot Duration . . . . .	8
<b>7 Canary Rollout Strategy</b>	<b>9</b>
7.1 Rollout Phases . . . . .	9
7.2 Safety Properties . . . . .	9
<b>8 Biological Analogy</b>	<b>10</b>
<b>9 Implementation</b>	<b>10</b>
9.1 Core Orchestration . . . . .	10
9.2 Envelope State Machine . . . . .	11
<b>10 Evaluation</b>	<b>12</b>
10.1 Experimental Setup . . . . .	12
10.2 Evaluation Metrics . . . . .	12
10.3 Baseline Comparisons . . . . .	12
<b>11 Discussion</b>	<b>13</b>
11.1 Key Advantages . . . . .	13
11.2 Limitations . . . . .	13
11.3 Future Work . . . . .	13
<b>12 Conclusion</b>	<b>13</b>

# 1 Introduction

The performance of large-scale language models depends not only on parameter size but also on **how information is distributed** across hierarchical computational modules. Unlike static architectures, biological cognition exhibits cyclic phases of activity and consolidation—**wakefulness** (real-time inference) and **sleep** (offline reconfiguration). We propose a computational counterpart of this duality within LLM ensembles.

In our approach, models of different capacities form a *cohort* governed by a **selector** (for task routing) and a **meta-selector** (for performance evaluation and escalation). A fixed fraction of the available compute (1/3) is continuously devoted to models in a *sleep phase*, where they fine-tune (distill) on data extracted from the operational ensemble. These models wake with improved specialization, reducing the need for higher-capacity inference. The process realizes a dynamic form of *computational homeostasis* and aligns with the broader theoretical model of **consciousness as collapsed computational time** [2].

## 1.1 Motivation

Large language models (LLMs) have demonstrated remarkable capabilities across diverse tasks, but their deployment at scale faces significant challenges:

- **Resource inefficiency:** Most queries don’t require the largest available model, yet static routing strategies often over-provision computational resources.
- **Coverage gaps:** Model ensembles typically have uneven coverage of the problem space, with some regions handled poorly by all models at a given capacity level.
- **Static deployment:** Traditional model serving treats models as fixed artifacts, missing opportunities for continuous adaptation based on production traffic patterns.
- **Knowledge consolidation:** Insights gained from high-capacity models are not systematically transferred to more efficient lower-capacity models.

Our sleep–wake orchestration addresses these challenges by creating a dynamic, self-optimizing ecosystem where models continuously adapt to production traffic patterns through targeted distillation.

## 1.2 Contributions

This paper makes the following contributions:

1. **Theoretical framework:** We formalize sleep–wake cycles in LLM ensembles using adjoint projections, connecting computational and biological principles.
2. **Gap-based learning:** We introduce a principled metric for identifying informational gaps in model coverage and targeting fine-tuning accordingly.
3. **Resource allocation:** We propose a bounded budget model (1/3 training, 2/3 inference) that maintains continuous learning without unbounded resource growth.
4. **Canary rollout:** We present a safe deployment strategy for newly trained models with automatic rollback on regression.
5. **Implementation:** We provide a complete open-source implementation demonstrating the architecture.

## 2 Theoretical Background

### 2.1 Projection Hierarchies and Computational Levels

Let  $L_0, L_1, L_2, \dots$  denote levels of computational capacity (e.g., parameter scales). Each model  $M_n$  operates on an effective manifold of tasks  $T_n \subset T$ .

**Definition 2.1** (Computational Level). A computational level  $L_n$  is characterized by:

- Resource capacity  $R_n$  (memory, compute)
- Task manifold  $T_n \subset T$
- Quality function  $Q_n : T \rightarrow [0, 1]$
- Cost function  $C_n : T \rightarrow \mathbb{R}^+$

The **selector**  $S$  maps each input  $x \in T$  to the lowest-level model capable of producing a satisfactory output under the cost-quality constraint:

$$\text{EVI}(x) = \mathbb{E}[Q_{n+1}(x) - Q_n(x)] - \lambda(C_{n+1} - C_n) > 0, \quad (1)$$

where  $Q$  measures response quality and  $C$  represents computational cost. This Expected Value of Information (EVI) criterion determines when escalation to a higher level is justified.

### 2.2 Adjoint Duality

Following *Adjoint Projections on Computational Hierarchies* [1], each transition between levels  $n \rightarrow n + 1$  can be represented as a pair of adjoint functors:

$$C_n \dashv P_n : L_{n+1} \leftrightarrows L_n, \quad (2)$$

where  $C_n$  denotes **collapse** (execution, loss of latent degrees of freedom) and  $P_n$  denotes **projection** (learning or reconstruction). The **sleep phase** corresponds to  $P_n$  (projection/updating), while the **wake phase** corresponds to  $C_n$  (collapse/inference).

**Theorem 2.1** (Sleep-Wake Duality). For any computational level  $n$ , the sleep-wake cycle implements an adjunction where:

1. Wake (collapse):  $C_n : \text{Model}_n \rightarrow \text{Output}$  produces concrete responses
2. Sleep (projection):  $P_n : \text{Experience} \rightarrow \text{Model}_n$  updates internal representations
3. Adjunction property:  $P_n \circ C_n \approx \text{id}$  (consolidation preserves essential information)

The system oscillates between these dual modes, maintaining bounded yet evolving computational coherence—a formal analog of consciousness as the collapse of computational time.

### 2.3 Behavioral Distance

To measure similarity between models and tasks, we define a behavioral distance metric in a shared embedding space.

**Definition 2.2** (Behavioral Distance). Let  $E : X \rightarrow \mathbb{R}^d$  be an embedding function mapping inputs to a  $d$ -dimensional latent space. The behavioral distance between a query  $x$  and a model  $M_i$ 's prototypical response is:

$$d_{\text{Beh}}(x, M_i) = \min_k \|E(x) - P_{M_i,k}\|_2, \quad (3)$$

where  $P_{M_i,k} \in \mathbb{R}^d$  are behavioral prototypes for model  $M_i$ .

The behavioral distance measures how well a model's typical responses align with a query's requirements, enabling routing decisions without invoking all models.

### 3 Cohort Architecture

#### 3.1 Core Components

A cohort  $\mathcal{C}$  is a set of models sharing comparable computational cost  $\kappa(M)$ . Each model maintains:

- **Behavioral prototypes**  $\{P_{M,k}\}_{k=1}^K$ : Typical embedding-space responses
- **Adapters**  $\{A_{M,c}\}$ : Cell-specific fine-tuned parameters
- **Performance metrics**: Success rate, cost, confidence
- **State**: Working, Sleeping, or Rollout

**Definition 3.1** (Cohort). A cohort  $\mathcal{C} = (M, S, \text{MS}, G)$  consists of:

- Models  $M = \{M_1, \dots, M_N\}$  of similar capacity
- Selector  $S : X \rightarrow M$  for routing
- Meta-selector MS for escalation decisions
- Gap index  $G$  tracking coverage deficits

#### 3.2 Selector

The selector routes input  $x$  by comparing its embedding  $E(x)$  to prototype centroids, choosing the model with minimal expected behavioral distance:

$$M^* = \arg \min_{M \in \mathcal{C}} [w_d \cdot d_{\text{Beh}}(x, M) + w_c \cdot C(M)], \quad (4)$$

where  $w_d, w_c$  are weights balancing quality and cost.

#### 3.3 Meta-Selector

The meta-selector monitors empirical success, expected value of improvement, and escalation rates to higher tiers. It implements three key functions:

1. **Confidence estimation**: Maps behavioral distance to confidence scores
2. **Escalation logic**: Decides when to invoke higher-capacity models
3. **Performance tracking**: Maintains aggregate metrics across the cohort

#### Algorithm 1 Meta-Selector Escalation Decision

**Require:** Query  $x$ , current model  $M_n$ , output  $y$ , confidence  $\text{conf}$

**Ensure:** Escalation decision

- ```

1: evi  $\leftarrow \mathbb{E}[Q_{n+1}(x) - Q_n(x)] - \lambda(C_{n+1} - C_n)$ 
2: if  $\text{conf} < \theta_{\text{crit}}$  and  $\text{evi} > 0$  then
3:   return ESCALATE
4: end if
5: return ACCEPT

```

## 4 Sleep–Wake Dynamics

### 4.1 Resource Allocation

Each model alternates between **wake** and **sleep** according to a rotation schedule constrained by the global compute budget:

- 1/3 of total memory: training pool (sleeping models)
- 2/3 of total memory: inference pool (working models)

**Definition 4.1** (Resource Envelope). A resource envelope reserves computational capacity for:

- Exactly one sleeper (training): budget  $B_{\text{train}}$
- Multiple workers (inference): budget  $B_{\text{infer}} = 2 \cdot B_{\text{train}}$

If the global budget is  $B_{\text{tot}}$ , then:

$$\frac{1}{3}B_{\text{tot}} = \sum_{c \in \text{classes}} N_c^{\text{train}} m_{\text{train}}(c), \quad (5)$$

$$\frac{2}{3}B_{\text{tot}} = \sum_{c \in \text{classes}} N_c^{\text{work}} m_{\text{infer}}(c), \quad (6)$$

where  $N_c^{\text{train}}$  and  $N_c^{\text{work}}$  are the number of training and working models in class  $c$ .

### 4.2 Sleep Cycle

In each sleep cycle:

1. **Gap identification:** The meta-selector identifies *gaps* in the behavioral manifold—regions  $C_i$  where cohort models fail but higher-level models succeed.
2. **Model selection:** The model with highest gap misalignment enters sleep, withdrawing from production.
3. **Fine-tuning:** The sleeping model fine-tunes on examples from gap cells using knowledge distillation from high-level teachers.
4. **Canary rollout:** The tuned model re-enters production gradually via canary deployment.

This process continuously rebalances knowledge across the cohort, maintaining equilibrium between specialization and coverage.

### 4.3 Gap Misalignment Score

To select which model should sleep next, we compute a gap misalignment score:

$$H_M = \sum_i G(C_i) \cdot (1 - P_M(\text{success}|C_i)), \quad (7)$$

where  $G(C_i)$  is the gap weight for cell  $C_i$  and  $P_M(\text{success}|C_i)$  is model  $M$ 's estimated success rate in that cell. The model with highest  $H_M$  is selected for sleep, as it would benefit most from targeted training.

## 5 Gap Metric and Learning Objective

### 5.1 Gap Definition

Define a local gap score for embedding-space region  $z$ :

$$G(z) = D_Q(z) \cdot (1 - \text{cover}(z)) \cdot \text{solvable\_up}(z), \quad (8)$$

where:

- $D_Q(z)$ : density of queries in embedding space (demand)
- $\text{cover}(z)$ : local success rate of cohort peers (current coverage)
- $\text{solvable\_up}(z)$ : probability that upper-tier models solved queries in this region (potential)

**Proposition 5.1** (Gap Properties). The gap function  $G(z)$  satisfies:

1.  $G(z) = 0$  if no queries arrive ( $D_Q(z) = 0$ )
2.  $G(z) = 0$  if cohort already covers region ( $\text{cover}(z) = 1$ )
3.  $G(z) = 0$  if higher models also fail ( $\text{solvable\_up}(z) = 0$ )
4.  $G(z)$  is maximal for high-demand regions with poor current coverage but good upper-level performance

This definition naturally prioritizes regions where there is actual user demand, current models underperform, and improvement is achievable.

### 5.2 Fine-Tuning Objective

The fine-tuning objective for a sleeping model  $M_s$  is:

$$\mathcal{L} = \mathbb{E}_{x \sim C_i} [w(x) \cdot \text{KL}(p_{M_s}(\cdot|x) \| p_{\text{teacher}}(\cdot|x))] + \lambda \|\tilde{P} - P^{\text{EMA}}\|^2 + \mu \text{Div}(M_s, \text{cohort}), \quad (9)$$

with weights:

$$w(x) = \alpha G(C_i) + \beta \text{EVI}(x) + \gamma \text{conf}(x), \quad (10)$$

where:

- **KD term:** Knowledge distillation from high-capacity teacher
- **Prototype regularization:**  $\lambda \|\tilde{P} - P^{\text{EMA}}\|^2$  stabilizes learned prototypes
- **Diversity term:**  $\mu \text{Div}(M_s, \text{cohort})$  maintains model diversity
- **Weights:** Combine gap priority ( $\alpha G$ ), expected improvement ( $\beta \text{EVI}$ ), and teacher confidence ( $\gamma \text{conf}$ )

*Remark 5.1* (Multi-Objective Optimization). The loss function balances three competing objectives: (1) imitation of teacher outputs, (2) stability of existing capabilities, and (3) diversity of model specialization. This prevents catastrophic forgetting while enabling targeted improvement.

### 5.3 Training Algorithm

This aligns the sleeping model toward *informational gaps* while stabilizing existing behaviors through regularization (Algorithm 2).

---

**Algorithm 2** Sleep Training for Gap Coverage

---

**Require:** Sleeping model  $M_s$ , target cells  $\{C_1, \dots, C_K\}$ , teacher model  $M_T$   
**Ensure:** Updated model with improved gap coverage

```
1: for each cell  $C_i$  in target cells do
2:    $\mathcal{D}_i \leftarrow \text{SampleFromCell}(C_i, n_{\min})$ 
3:   Initialize or load adapter  $A_i$  for cell  $C_i$ 
4:   for step  $t = 1$  to  $T_{\text{steps}}$  do
5:     Sample batch  $(x_1, \dots, x_b) \sim \mathcal{D}_i$ 
6:     for each  $x_j$  in batch do
7:        $(y_j, \text{conf}_j) \leftarrow M_T(x_j)$ 
8:        $w_j \leftarrow \alpha G(C_i) + \beta \text{EVI}(x_j) + \gamma \text{conf}_j$ 
9:     end for
10:    Compute loss  $\mathcal{L}$  from Equation (9)
11:    Update adapter  $A_i$  via gradient descent
12:   end for
13:   Update prototypes  $P_{M_s}$  for cell  $C_i$  using EMA
14: end for
15: return Updated model  $M_s$  with cell-specific adapters
```

---

## 6 Resource Allocation Model

### 6.1 Envelope-Based Allocation

Each model class (3B, 8B, 13B, ...) operates in **envelopes**:

- One training (sleeping) model per envelope
- A working pool consuming twice the compute of the sleeper

**Definition 6.1** (Envelope Capacity). An envelope for model class  $c$  has:

$$B_{\text{train}}(c) = m_{\text{train}}(c) \quad (11)$$

$$B_{\text{infer}}(c) = 2 \cdot m_{\text{train}}(c) \quad (12)$$

where  $m_{\text{train}}(c)$  is the memory footprint for training a model of class  $c$ .

### 6.2 Multi-Envelope Orchestration

For multiple cohorts competing for resources:

1. **Priority scoring:** Rank cohorts by gap pressure  $\sum_i G(C_i)$
2. **Envelope allocation:** Allocate envelopes to highest-priority cohorts first
3. **Resource constraints:** Respect global budget limits from Equations (5)–(6)

This maintains continuous learning within bounded energy and compute constraints (Algorithm 3).

### 6.3 Adaptive Slot Duration

Training slot duration scales with model size:

Larger models require longer training periods to effectively consolidate knowledge (Table 1).

---

**Algorithm 3** Multi-Envelope Resource Allocation

---

**Require:** Cohorts  $\{\mathcal{C}_1, \dots, \mathcal{C}_M\}$ , resource pools  $(B_{\text{train}}, B_{\text{infer}})$

**Ensure:** List of allocated envelopes

```
1: Sort cohorts by gap pressure:  $\mathcal{C}_{\sigma(1)}, \dots, \mathcal{C}_{\sigma(M)}$ 
2:  $\mathcal{E} \leftarrow \emptyset$ 
3: for each cohort  $\mathcal{C}$  in sorted order do
4:    $m_T \leftarrow \text{AvgTrainMem}(\mathcal{C})$ 
5:    $m_I \leftarrow \text{AvgInferMem}(\mathcal{C})$ 
6:    $n_{\max} \leftarrow \min(\lfloor B_{\text{train}}/m_T \rfloor, \lfloor B_{\text{infer}}/(2m_I) \rfloor)$ 
7:   for  $i = 1$  to  $n_{\max}$  do
8:     Create envelope  $E_i$  with budgets  $(m_T, 2m_I)$ 
9:      $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_i\}$ 
10:     $B_{\text{train}} \leftarrow B_{\text{train}} - m_T$ 
11:     $B_{\text{infer}} \leftarrow B_{\text{infer}} - 2m_I$ 
12:   end for
13: end for
14: return  $\mathcal{E}$ 
```

---

Table 1: Default training slot durations by model class

| Model Class | Parameters  | Slot (min) | LoRA Rank |
|-------------|-------------|------------|-----------|
| 3B          | 3 billion   | 30         | 16        |
| 8B          | 8 billion   | 60         | 16        |
| 13B         | 13 billion  | 90         | 16        |
| 30B         | 30 billion  | 150        | 16        |
| 70B         | 70 billion  | 360        | 16        |
| 175B        | 175 billion | 720        | 16        |
| GPT5        | >1 trillion | 1080       | 16        |

## 7 Canary Rollout Strategy

After sleep training completes, models re-enter production via a gradual canary deployment:

### 7.1 Rollout Phases

1. **Canary start:** Route 2% of target-cell traffic to updated model
2. **Evaluation:** Monitor performance on target cells for regression
3. **Expansion:** Double traffic share if improving and no regression detected
4. **Promotion:** Transition to full working status after stable performance
5. **Rollback:** Revert adapters and return to working if regression occurs

### 7.2 Safety Properties

The canary strategy ensures:

- **Bounded risk:** Only small fraction of traffic exposed to potentially degraded model
- **Rapid detection:** Regression identified quickly through continuous monitoring

---

**Algorithm 4** Canary Rollout Management

---

**Require:** Model  $M$  in rollout state, target cells  $\{C_1, \dots, C_K\}$

**Ensure:** Promotion or rollback decision

```
1: stats ← EvaluateOnCells( $M, \{C_1, \dots, C_K\}$ )
2: if stats.improves and ¬stats.regress_outside then
3:    $M.\text{traffic} \leftarrow \min(2 \cdot M.\text{traffic}, 0.5)$ 
4:   if StableForSlots( $M, k = 2$ ) then
5:      $M.\text{state} \leftarrow \text{WORKING}$ 
6:      $M.\text{traffic} \leftarrow 0$ 
7:     return PROMOTED
8:   end if
9: else
10:   RollbackAdapters( $M$ )
11:    $M.\text{state} \leftarrow \text{WORKING}$ 
12:   return ROLLBACK
13: end if
14: return CONTINUE
```

---

- **Automatic recovery:** Rollback restores previous behavior without manual intervention
- **Gradual expansion:** Traffic increases only after demonstrating improvement

This provides safety guarantees while enabling continuous model improvement (Algorithm 4).

## 8 Biological Analogy

The mechanism parallels **sleep-dependent learning** in the brain:

- **Cortical–subcortical consolidation:** Auxiliary modules refine predictions based on higher-level errors during sleep.
- **Replay mechanisms:** Slow-wave neural replay stabilizes distributed representations after active periods [4].
- **Synaptic homeostasis:** Sleep allows selective strengthening and weakening of connections, maintaining network capacity.
- **Energy efficiency:** Intensive learning happens intermittently (sleep), preserving real-time responsiveness during wake.

Analogously, the LLM cohort’s smaller models replay high-value examples from production logs, adjusting low-level weights via distillation from high-level models. Energy-intensive learning happens intermittently (sleep), preserving real-time responsiveness (Table 2).

## 9 Implementation

### 9.1 Core Orchestration

The main orchestration loop manages all cohorts and envelopes:

```
1 def global_tick(cohorts, resource_pools):
2     # 1. Refresh gap indices for all cohorts
3     for cohort in cohorts:
```

Table 2: Biological-computational correspondence

| Biological System      | Computational System                       |
|------------------------|--------------------------------------------|
| Wakefulness            | Active inference (model serves queries)    |
| Sleep                  | Offline fine-tuning (model trains on gaps) |
| Cortical consolidation | Distillation from high-capacity teacher    |
| Hippocampal replay     | Replay of informational gap examples       |
| Synaptic homeostasis   | Prototype and diversity regularization     |
| Energy conservation    | Bounded 1/3 training budget                |

```

4     logs = collect_logs(cohort, window_hours=4)
5     cohort.gap_index.update(logs)
6
7 # 2. Allocate/refresh envelopes based on gap pressure
8 envelopes = ensure_envelopes(cohorts, resource_pools)
9
10 # 3. Tick each envelope independently
11 for envelope in envelopes:
12     envelope_tick(envelope, resource_pools)
13
14 # 4. Global prototype refresh and metrics
15 for cohort in cohorts:
16     refresh_metrics_and_prototypes(cohort)

```

Listing 1: Global orchestration loop

## 9.2 Envelope State Machine

Each envelope manages its own sleep-wake cycle:

```

1 def envelope_tick(envelope, pools):
2     if envelope.state in ("idle", "working"):
3         # Try to start new sleep cycle
4         sleeper = pick_sleeper(envelope.cohort)
5         if sleeper and fits_train_pool(sleeper, envelope.train_budget):
6             envelope.sleeper = sleeper
7             sleeper.state = ModelState.SLEEPING
8             envelope.target_cells = cohort.gap_index.top_cells(M=5)
9             envelope.state = "sleeping"
10
11 elif envelope.state == "sleeping":
12     # Run training
13     run_sleep_training_for_envelope(envelope)
14     if training_complete(envelope):
15         start_envelope_canary(envelope)
16
17 elif envelope.state == "rollout":
18     # Manage canary
19     stats = evaluate_on_cells(envelope.sleeper, envelope.
        target_cells)

```

```

20     if improves(stats) and no_regress_outside(stats):
21         increase_traffic(envelope.sleeper)
22         if stable_for_slots(envelope.sleeper):
23             promote_to_worker(envelope)
24     else:
25         rollback_adapters(envelope.sleeper)

```

Listing 2: Envelope tick function

## 10 Evaluation

### 10.1 Experimental Setup

To evaluate the architecture, we propose experiments with parameterized cohorts under controlled traffic:

- **Model scales:** 1B, 3B, 8B, 13B parameter models
- **Cohort sizes:** 4 models per cohort
- **Query distribution:** Mixture of easy (70%), medium (25%), hard (5%) queries
- **Budget:** 256 GB VRAM total (85 GB training, 171 GB inference)
- **Duration:** 100 sleep–wake cycles

### 10.2 Evaluation Metrics

Table 3: Evaluation metrics and expected behaviors

| Metric               | Definition                                          | Expected Behavior                 |
|----------------------|-----------------------------------------------------|-----------------------------------|
| Escalation rate      | $\frac{\# \text{ escalations}}{\# \text{ queries}}$ | Decrease over time as gaps filled |
| Confidence stability | $\text{Var}(\text{entropy})$                        | Decrease as models specialize     |
| Gap coverage         | $\sum_i G(C_i)$                                     | Sublinear growth or decline       |
| Cost per query       | Mean inference cost                                 | Decrease as routing improves      |

**Hypothesis 10.1.** *Periodic fine-tuning on informational gaps will yield:*

1. Sublinear growth of escalation cost while maintaining accuracy
2. Improved coverage ( $\text{decreasing } \sum G(C_i)$ ) over time
3. Stable or decreasing cost per query as routing becomes more efficient
4. Reduced variance in confidence scores as models specialize

### 10.3 Baseline Comparisons

Compare against:

- **Static routing:** Fixed model selection without adaptation
- **Uniform fine-tuning:** Training on full distribution instead of gaps
- **No sleep:** All models always active (no training budget)
- **Random selection:** Random choice of sleeper instead of gap-based

## 11 Discussion

### 11.1 Key Advantages

The proposed system integrates *continuous self-optimization* with formal constraints on computational adjunctions. Key advantages:

- **Self-organizing:** System automatically identifies and fills coverage gaps
- **Resource-bounded:** Fixed 1/3–2/3 allocation prevents unbounded growth
- **Safe:** Canary rollout with automatic rollback protects production
- **Principled:** Grounded in formal theory of adjoint projections
- **Biologically-inspired:** Mirrors sleep-dependent consolidation in brains

### 11.2 Limitations

Current limitations include:

1. **Stationary assumption:** Gap distribution assumed stable during cycles
2. **Manual configuration:** Hyperparameters  $(\alpha, \beta, \gamma, \lambda, \mu)$  require tuning
3. **Embedding quality:** Depends on good semantic embedding function
4. **Teacher availability:** Requires access to high-capacity teacher models
5. **Cold start:** Initial cycles have poor gap estimates

### 11.3 Future Work

Future directions include:

1. **Non-stationary environments:** Adapting to domain drift and concept shift through distribution shift detection and adaptive sleep schedules
2. **Differentiable meta-selectors:** Learning optimal sleep schedules by treating the meta-selector as a learned policy optimized via reinforcement learning
3. **Behavioral manifold entropy:** Formalizing cognitive diversity through the entropy of prototype distributions
4. **Mixture-of-experts integration:** Treating MoE experts as cohort members and applying sleep–wake cycles at the expert level
5. **Retrieval-augmented models:** Using gap cells to guide retrieval corpus curation and coordinating sleep across retrieval and generation

## 12 Conclusion

We have presented a formal architecture for sleep–wake orchestration in hierarchical LLM cohorts, grounded in the theory of adjoint projections on computational hierarchies. The system implements:

1. **Gap-based learning:** Models fine-tune on regions where they underperform but higher models succeed

2. **Resource-bounded operation:** Fixed 1/3 training, 2/3 inference allocation
3. **Safe deployment:** Canary rollout with automatic rollback
4. **Theoretical foundation:** Formal adjunction between sleep (projection) and wake (collapse)
5. **Biological inspiration:** Computational analog of sleep-dependent consolidation

The resulting system self-organizes toward optimal coverage and energy-efficient reasoning, providing both theoretical understanding and practical implementation of self-maintaining model ecosystems. By continuously adapting to production traffic patterns through principled sleep-wake cycles, the architecture enables sustainable deployment of large-scale language model ensembles.

Future work will extend the framework to non-stationary environments, differentiable meta-selection, and integration with mixture-of-experts and retrieval-augmented architectures. The complete implementation is available at <https://github.com/KarolFilipKowalczyk/Consciousness>.

## Acknowledgments

This work builds on the theoretical foundations developed in [1] and [2].

## References

- [1] Kowalczyk, K. (2025). *Adjoint Projections on Computational Hierarchies*.
- [2] Kowalczyk, K. (2025). *Consciousness as Collapsed Computational Time*.
- [3] Kowalczyk, K. (2025). *Selectors and Meta-Selectors in Large Language Model Hierarchies*.
- [4] Tononi, G., & Cirelli, C. (2016). Sleep and the price of plasticity: From synaptic to systems neuroscience. *Neuron*, 81(1), 12–34.
- [5] Luo, Y., et al. (2023). Catastrophic forgetting in continual fine-tuning of LLMs. *arXiv preprint arXiv:2308.08747*.
- [6] Parthasarathy, S., et al. (2024). The ultimate guide to fine-tuning LLMs. *Technical Report*.