Selectors and Meta-Selectors in Large Language Model Hierarchies:
A Practical Framework for Efficient Model Routing

Karol Kowalczyk
(revised by Claude)

November 2025

Abstract

We present a framework for **adaptive model selection** in hierarchical LLM systems, where queries are routed to appropriately-sized models based on predicted difficulty. Our approach uses **behavioral distance** in a shared embedding space to measure how well a model's output would align with a query's intent. Rather than invoking all models (computationally expensive), we learn **behavioral prototypes** from historical model outputs and use these for fast approximation. A **meta-selector** monitors confidence and manages escalation to higher-capacity models when needed. We provide formal problem definitions, concrete algorithms, and open-source implementations. While full empirical validation remains future work, our framework bridges theory and practice for cost-efficient LLM inference.

Keywords: model routing, hierarchical inference, behavioral distance, LLM efficiency, meta-learning

1 Introduction

1.1 Motivation

Large language models (LLMs) achieve remarkable capabilities at the cost of massive computational resources. For example:

- GPT-4 may cost $0.03 per 1K tokens
- Claude-3 Opus costs $0.015 per 1K tokens
- GPT-3.5 Turbo costs $0.0005 per 1K tokens (60× cheaper)

However, empirical analysis shows that **most queries don't require the largest models**:

- Factual queries ("What is the capital of France?") → answered correctly by small models
- Simple summarization → medium models sufficient
- Complex reasoning or creative synthesis → large models necessary

The challenge: **automatically match query difficulty to model capacity** to minimize cost while maintaining quality.

1.2 Problem Formulation

Given:

- A hierarchy of LLMs: $\{M_1, M_2, \ldots, M_N\}$ where M_i is cheaper but less capable than M_{i+1}
- A query x with unknown difficulty
- Cost function $C(M_i)$ and quality metric $Q(M_i, x)$

Goal: Select $M^* = \arg\min_i [C(M_i)]$ subject to $Q(M_i, x) \geq \theta$

Key Insight: We can estimate $Q(M_i, x)$ by measuring **behavioral distance**: how close the model's output embedding is to the query's embedding in a shared semantic space.

1.3 Contributions

1. **Formal framework** for behavioral distance-based model selection
2. **Two-phase design** separating offline prototype learning from online routing
3. **Meta-selector** for confidence validation and hierarchical escalation
4. **Concrete algorithms** with complexity analysis
5. **Open-source implementation** in Python

1.4 Related Work

Mixture of Experts (MoE): Use learned gating networks to route inputs [3]. Our approach differs by using geometric distance in a shared embedding space rather than learned weights.

FrugalGPT & Cascade Methods: Route through models sequentially [2]. We enable **parallel estimation** via prototypes and **cross-level comparison**.

Speculative Decoding: Uses small models to speed up large models [4]. We focus on selecting the right model entirely, not accelerating generation.

Early Exit Networks: Exit from layer computation early. We operate at the model-level, not layer-level.

2 Theoretical Framework

2.1 Behavioral Distance: The Ideal Case

Definition 1 (Behavioral Distance)**.** Given a query x and model M_i, the behavioral distance is:

$$d_{\text{behav}}(M_i, x) = D_S(f_S(x), g_S(M_i(x))) \tag{1}$$

where:

- f_S: query embedding function
- g_S: output embedding function
- D_S: distance metric (e.g., cosine distance)
- $M_i(x)$: actual output of model M_i on query x

Interpretation: Lower distance means the model's output semantically aligns with the query's intent.

Ideal Selection Rule:

$$M^* = \arg\min_i \left[w_d \cdot d_{\text{behav}}(M_i, x) + w_c \cdot C(M_i) \right] \tag{2}$$

Computational Cost: This requires calling **all N models**, which defeats the purpose of efficient routing!

2.2 Practical Approximation via Prototypes

To avoid invoking all models, we introduce **behavioral prototypes**:

Definition 2 (Behavioral Prototype)**.** A prototype $p_k^{(i)}$ for model M_i is a representative output embedding learned from historical data:

$$p_k^{(i)} = \text{Centroid}\{g_S(M_i(x_j)) : x_j \in \text{Cluster}_k\} \tag{3}$$

Approximation:

$$d_{\text{behav}}(M_i, x) \approx \min_k D_S(f_S(x), p_k^{(i)}) \tag{4}$$

Trade-off:

- Exact: $O(N)$ model calls per query
- Approximate: $O(N \times K)$ distance computations (no model calls)
- Where K = number of prototypes per model (typically 3-10)

Approximation Error:

$$\epsilon(x) = |d_{\text{behav}}(M_i, x) - \min_k D_S(f_S(x), p_k^{(i)})| \tag{5}$$

This depends on:

- Prototype quality (how well they cover model behavior space)
- Query novelty (distance to training distribution)
- Embedding space quality

2.3 Intra-Level Selection

Observation: Models at the "same level" (similar cost) often specialize differently.

- GPT-4 vs Claude-3 Opus (both "large")
- Llama-70B vs Mixtral-8x7B (both "medium")

Extended Selection Rule:

$$M^* = \arg\min_{i,j} \left[w_d \cdot d_{\text{behav}}(M_i^{(j)}, x) + w_c \cdot C(M_i^{(j)}) \right] \tag{6}$$

where $M_i^{(j)}$ denotes model j at level i.

This enables **horizontal selection** (within level) and **vertical selection** (across levels).

2.4 Meta-Selector and Escalation

Problem: The selector might make wrong decisions due to:

- Poor prototype coverage
- Distribution shift
- Miscalibrated confidence

Solution: A **meta-selector** validates decisions post-hoc and escalates when needed.
Confidence Estimation:

$$\text{conf}(d) = \frac{1}{1 + \exp(k(d - d_0))} \tag{7}$$

where d is behavioral distance, and k, d_0 are calibrated on validation data.

Escalation Rule (Expected Value of Information):

$$\text{Escalate if: } (\Delta Q - \lambda \Delta C) > 0 \tag{8}$$

where:

- ΔQ = expected quality improvement from higher model
- ΔC = additional cost
- λ = cost weight

3 Algorithms

Algorithm 1 Offline Prototype Learning

Require: Model set $\{M_1, \ldots, M_N\}$, query corpus Q
Ensure: Prototype bank B

for each model M_i **do**
  Sample diverse queries $Q_{\text{sample}} \subset Q$
  for each query $q \in Q_{\text{sample}}$ **do**
    $\text{output}_q = M_i(q)$ {Actually call the model}
    $\text{embed}_q = g_S(\text{output}_q)$
    Store (q, embed_q)
  end for
  Cluster embeddings into K clusters using K-means
  for each cluster k **do**
    $p_k^{(i)} = \text{mean}(\{\text{embed}_q : q \in \text{Cluster}_k\})$
  end for
  Store prototypes: $B[M_i] = \{p_1^{(i)}, \ldots, p_K^{(i)}\}$
end for
return B

Complexity: $O(N \times |Q_{\text{sample}}| \times T_{\text{model}})$ where T_{model} = time to call one model

Algorithm 2 Online Selection (Exploit Mode)

Require: Query x, prototype bank B, models $\{M_1, \ldots, M_N\}$
Ensure: Chosen model M^*, actual output y

Embed query: $z_x = f_S(x)$
for each model M_i **do**
  Find nearest prototype: $k^* = \arg\min_k D_S(z_x, p_k^{(i)})$
  $d_i = D_S(z_x, p_{k^*}^{(i)})$
  Compute score: $s_i = w_d \times d_i + w_c \times C(M_i) + w_l \times \text{level}(M_i)$
end for
Select best model: $M^* = \arg\min_i s_i$
Call selected model: $y = M^*(x)$ {Only invoke ONE model}
(Optional) Update prototype with exponential moving average
return M^*, y

Complexity: $O(N \times K \times d)$ where d = embedding dimension

Algorithm 3 Online Selection (Explore Mode)

Require: Query x, prototype bank B, models $\{M_1, \ldots, M_N\}$, sample size S
Ensure: Chosen model M^*, actual output y

Embed query: $z_x = f_S(x)$
Sample models strategically: $M_{\text{sample}} = \{M_{i1}, M_{i2}, \ldots, M_{iS}\}$ (S models)
for each $M_{ij} \in M_{\text{sample}}$ **do**
  Actually call the model: $y_j = M_{ij}(x)$
  Embed output: $z_j = g_S(y_j)$
  Compute true distance: $d_j = D_S(z_x, z_j)$
  Compute score: $s_j = w_d \times d_j + w_c \times C(M_{ij})$
  Update prototypes with z_j
end for
Select best: $M^* = \arg\min_j s_j$
y^* = corresponding y_j
return M^*, y^*

Complexity: $O(S \times T_{\text{model}})$. Note: $S \ll N$, typically $S = 2 - 5$

Algorithm 4 Meta-Selector Validation

Require: Query x, selected model M^*, output y, session state σ
Ensure: Escalation decision

Validate selection:
$z_x = f_S(x)$, $z_y = g_S(y)$
$d_{\text{actual}} = D_S(z_x, z_y)$
conf = confidence_fn(d_{actual})
Update session state:
σ.confidence_history.append(conf)
σ.query_history.append(x)
σ.detect_repetition(x)
Check budget:
if σ.budget_remaining ≤ 0 **then**
  return NO_ESCALATION
end if
Check escalation triggers:
if σ.is_critical AND conf $< \theta_{\text{crit}}$ **then**
  return ESCALATE(reason=CRITICAL)
else if σ.repetition_count ≥ 2 AND conf $< \theta_{\text{rep}}$ **then**
  return ESCALATE(reason=REPETITION)
else if Expected Value of Information positive **then**
  return ESCALATE(reason=POSITIVE_EVI)
else if conf $< \theta$ **then**
  return ESCALATE(reason=LOW_CONFIDENCE)
end if
return NO_ESCALATION

Complexity: $O(d)$ for embedding + $O(1)$ for logic

4 Implementation Details

4.1 Embedding Spaces

Option 1: SentenceTransformers [5]

- Pre-trained: `all-MiniLM-L6-v2` (384-dim, fast)
- Semantic: `all-mpnet-base-v2` (768-dim, better quality)

Option 2: LLM-based embeddings

- OpenAI `text-embedding-3-small` (1536-dim)
- Cohere embed-v3 (1024-dim)

Key Requirement: Same embedding function for queries and outputs to ensure shared semantic space.

4.2 Prototype Management

Listing 1: Prototype Bank Implementation

```python
class PrototypeBank:
    def __init__(self, n_prototypes=5):
        self.prototypes = {}  # model_id -> list of embeddings
        self.n_prototypes = n_prototypes

    def update(self, model_id, new_embedding, alpha=0.1):
        """Update prototypes with exponential moving average"""
        if model_id not in self.prototypes:
            self.prototypes[model_id] = [new_embedding]
        else:
            # Find nearest prototype
            distances = [cosine_distance(new_embedding, p)
                         for p in self.prototypes[model_id]]
            nearest_idx = np.argmin(distances)

            # EMA update
            self.prototypes[model_id][nearest_idx] = (
                alpha * new_embedding +
                (1 - alpha) * self.prototypes[model_id][nearest_idx]
            )
```

4.3 Confidence Calibration

The confidence function $\text{conf}(d)$ needs calibration on validation data:

1. Collect pairs $(d_i, \text{success}_i)$ where success $= 1$ if output was acceptable
2. Fit logistic regression: $P(\text{success}|d) = \sigma(k(d - d_0))$
3. Extract parameters k and d_0

4.4 System Architecture

Listing 2: System Overview

```
# Offline phase
selector = BehavioralSelector(models, embedding_space)
selector.build_prototypes(training_corpus)

# Online phase
meta_selector = MetaSelector(selector)
for query in stream:
    result = selector.select(query, mode='exploit')
    decision = meta_selector.validate(query, result)

    if decision.should_escalate:
        result = selector.escalate(query, decision.target_model)

    return result.output
```

5 Experimental Design (Future Work)

5.1 Datasets

Propose evaluation on:

1. **MMLU**: Multi-domain questions, varying difficulty
2. **HumanEval**: Coding tasks (clear correctness metric)
3. **TruthfulQA**: Factual accuracy
4. **AlpacaEval**: Instruction following
5. **MT-Bench**: Diverse capabilities

5.2 Baselines

- **Random**: Random model selection
- **Always-Small**: Always use cheapest model
- **Always-Large**: Always use best model (upper bound on quality)
- **Cascade**: Try small→medium→large until confidence threshold
- **Learned Router**: Train classifier on query features

5.3 Metrics

Primary:

- **Cost Reduction**: Total cost vs always-large baseline
- **Quality**: Accuracy/semantic similarity vs always-large

Secondary:

- Latency (including routing overhead)
- Pareto efficiency curve
- Calibration error of confidence estimates

5.4 Ablations

Test impact of:

- Number of prototypes ($K = 1, 3, 5, 10, 20$)
- Exploration rate ($\epsilon = 0, 0.05, 0.1, 0.2$)
- Distance metric (cosine vs Euclidean vs Mahalanobis)
- Embedding model (small vs large)
- Meta-selector (with vs without)

6 Discussion

6.1 When This Works Well

Best scenarios:

- Large query volumes (amortize offline cost)
- Clear difficulty stratification (queries naturally cluster)
- Stable model behaviors (prototypes remain valid)
- Multiple models available at each level

Example: Customer support chatbot with mix of:

- Simple FAQs → small model
- Product questions → medium model
- Complex troubleshooting → large model

6.2 Limitations

Limitation 1: Requires offline calibration phase

- Need representative query corpus
- Models must be available for profiling

Limitation 2: Approximation error

- Prototypes may not cover all model behaviors
- Novel queries may be misrouted

Limitation 3: Embedding quality dependence

- If embeddings don't capture semantic adequacy, behavioral distance fails
- Different tasks may need different embedding spaces

Limitation 4: Static model assumption

- If models are retrained/updated, prototypes become stale
- Need periodic recalibration

Limitation 5: Multi-turn conversations

- Current formulation is stateless (per-query)
- Doesn't account for conversation history

6.3 Extensions

Extension 1: Task-specific embedding spaces

- Learn separate spaces for coding, math, creative writing
- Route to appropriate space based on query type

Extension 2: Dynamic prototypes

- Continuously update prototypes online
- Detect distribution shift and trigger recalibration

Extension 3: Multi-objective optimization

- Balance cost, latency, and quality simultaneously
- Pareto frontier selection

Extension 4: Hierarchical meta-selectors

- Meta-meta-selectors to validate meta-selector decisions
- Recursive control structure

6.4 Relationship to Adjoint Projections

Our framework can be viewed through category-theoretic lens [1]:

- Collapse $C : M_{i+1} \to M_i$: Use simpler model
- Projection $P : M_i \to M_{i+1}$: Escalate to complex model
- Adjunction: $P \circ C \approx \text{id}$ (round-trip preserves information)

However, we deliberately keep the paper focused on practical concerns rather than full categorical formalism. The interested reader can explore connections to:

- Functorial semantics of computation
- Information-theoretic bounds on model compression
- Category of models with morphisms as distillation/escalation

7 Conclusion

We presented a practical framework for adaptive model selection in LLM hierarchies using behavioral distance and learned prototypes. Our two-phase design—offline prototype learning and online fast routing—enables efficient selection without invoking all models. The meta-selector provides confidence validation and rational escalation.

Key contributions:

1. Formal problem definition bridging theory and practice
2. Concrete algorithms with complexity analysis
3. Clear specification of approximations and trade-offs
4. Open-source implementation

Honest assessment: While we believe this approach is promising, full empirical validation across diverse tasks and model hierarchies remains essential future work. We hope this framework provides a foundation for further research in cost-efficient LLM inference.

References

[1] Kowalczyk, K. (2025). *Adjoint Projections on Computational Hierarchies*. Manuscript in preparation.

[2] Chen, L., Zaharia, M., and Zou, J. (2023). *FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance*. arXiv preprint arXiv:2305.05176.

[3] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

[4] Leviathan, Y., Kalman, M., and Matias, Y. (2023). *Fast Inference from Transformers via Speculative Decoding*. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

[5] Reimers, N. and Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

A Pseudocode for Complete System

See supplementary materials for full Python implementation at:
https://github.com/KarolFilipKowalczyk/Consciousness