Preset-Driven Modular Decision Engines for Cost-Efficient, Resilient Multi-LLM Serving

Technical Report Production Systems Architecture

Mugisha Clinton
Software Engineer
Ridelink
clinton@ride-link.com

Amon Nyesigye
Chief Technology Officer
Ridelink
amon@ride-link.com

Ikonde Nekemiah Arnold
Software Engineer
Ridelink
arnold@ride-link.com

Abstract

As the ecosystem of large language models (LLMs) expands, production systems increasingly require *model selection* to balance output quality, response latency, and per-request cost simultaneously. We present PRESROUTE, a *preset-driven modular decision engine* that routes inference requests to specialised LLM configurations termed *presets* based on detected user intent spanning document processing, web-search grounding, audio transcription, and general multimodal queries. Each preset encapsulates a model or ordered model array, provider preferences, decoding parameters, and a tailored system prompt, while provider-level fallbacks ensure resilience against rate limits and outages. Configuration and versioning are fully decoupled from application code, enabling instant rollback and zero-redeployment iteration. We evaluate PRESROUTE on four intent classes across >12,000 production requests, reporting task success rate, cost per successful outcome, end-to-end latency (p50/p95), and fallback frequency. A *free-primary, paid-fallback* strategy using a 400B sparse MoE model as the primary preset reduced model spend by **64%** relative to a paid-primary baseline while increasing median response time by only **2.1 s**. We situate the design within the literature on LLM routing and cascades (FrugalGPT, Eagle, GraphRouter), discuss governance and safety considerations, and release an open evaluation protocol to facilitate reproducibility.

1 Introduction

Modern AI-backed applications face hard operational constraints: sub-second latency budgets, strict per-token cost targets, and five-nines availability requirements while users simultaneously demand high factual accuracy and grounded answers. A single "best" model rarely dominates across all request types. Extraction-heavy document processing benefits from deterministic decoding and structured output schemas; web-grounded questions require search-aware prompts and larger context windows; multimodal inputs require fundamentally different model classes entirely.

These observations motivate **model routing**: selecting the most appropriate LLM configuration at runtime for each incoming request. Yet existing production routing is often ad-hoc a conditional branch in application code that grows brittle as the model landscape evolves. Hard-coded model identifiers make version management difficult, and poor separation of concerns couples prompt engineering to business logic.

This paper makes the following contributions:

1. We formalise the concept of a *preset* an atomic, versioned unit of LLM configuration and describe a *preset registry* that decouples configuration from application code (§3).
2. We propose a four-class intent taxonomy (Document, Search, Audio, Multimodal) and show how each intent maps to a specialised preset, reducing prompt sprawl and improving per-route quality (§4).
3. We introduce a *free-primary, paid-fallback* deployment pattern and evaluate it on >12,000 production requests, demonstrating significant cost reduction with modest latency overhead (§7).
4. We describe an immutable-history versioning and rollback mechanism that supports safe experimentation without redeployments (§6).

5. We discuss governance, safety constraints, and open-source evaluation protocols (§8).

2 Background and Related Work

2.1 LLM Routing and Cascades

Chen et al. [2023] introduced **FrugalGPT**, which constructs cascades of LLMs querying cheaper models first and escalating only when the confidence of the cheaper model is insufficient. Their results showed up to 98% cost reduction with matched performance on common benchmarks. PRESROUTE adopts a similar escalation principle but operationalises it at the *provider and model-array* level rather than at the single-request inference level, and pairs it with intent-based dispatch.

Zhao et al. [2024] proposed **Eagle**, a training-free router that estimates per-model quality from lightweight proxy signals to select among a panel of LLMs without labelled routing data. Feng et al. [2024] introduced **GraphRouter**, which represents the routing problem as graph-structured matching between task features and model capabilities, learning an edge-scoring function from historical outcomes. Our work is complementary: PRESROUTE currently uses rule-based intent classification, but the preset selection layer is designed to be swappable with learned routers such as Eagle or GraphRouter.

Dekoninck et al. [2024] provide a theoretical unification of routing and cascading, characterising the conditions under which each strategy achieves Pareto-optimality on the cost–quality frontier. Their analysis confirms that routing (selecting a single model) is preferable when models have complementary strengths across non-overlapping request classes precisely the multi-intent setting we address.

2.2 Sparse Mixture-of-Experts Architectures

Fedus et al. [2021] demonstrated that sparse MoE architectures can scale to trillion-parameter models while activating only a small fraction of parameters per token. The *Switch Transformer* routes each token to a single expert, achieving training efficiency gains proportional to the number of experts. This principle motivates our use of Trinity-Large-Preview a 400 B sparse MoE with only 13 B active parameters per forward pass as a cost-efficient primary model.

2.3 Production Multi-Model Systems

Industrial systems such as Google's mixture-of-agents pipeline and OpenAI's routing between GPT-3.5 and GPT-4 share the principle of cost-aware model selection, but public documentation is limited. Our work provides, to our knowledge, the first detailed description and controlled evaluation of a *preset-driven* architecture with immutable versioning and provider-level fallbacks in a live production environment.

3 System Design

3.1 Design Goals

PRESROUTE is guided by five operational principles:

- **Per-intent accuracy.** Specialised prompts and decoding parameters improve task-specific quality; a general-purpose prompt is a lossy compression.
- **Cost control.** Frontier-model capacity should be reserved for requests that genuinely require it; cheaper or free models should handle the remainder.
- **Resilience.** Provider outages and rate limits are operational facts of life; the system must degrade gracefully without manual intervention.
- **Operational safety.** Every configuration change should be auditable, reversible, and isolated to a staging environment before promotion to production.
- **Separation of concerns.** Routing logic, prompt engineering, and model selection must not bleed into business logic.

3.2 The Preset Abstraction

A *preset* is an immutable, versioned record containing:

Preset Schema

`id`	Unique stable identifier
`version`	Monotonic integer
`model_array`	Ordered list of model/provider pairs
`temperature`	Decoding temperature $\in [0, 1]$
`top_p`	Nucleus sampling parameter
`max_tokens`	Output length budget
`system_prompt`	Route-specific instruction
`safety_policy`	Refusal style, output schema
`created_at`	Immutable creation timestamp
`promoted_by`	Identity of approving operator

The **preset registry** is a versioned key-value store. Application code references a preset by its stable `id` and an environment tag (`staging` or `production`); the registry resolves the tag to a concrete version at call time. This enables:

- **Instant rollback:** repoint `production` to a prior version.
- **Blue/green testing:** run two preset versions simultaneously with traffic splitting.
- **Auditability:** every change is a new version; the full history is preserved.

4 Routing Architecture

4.1 Intent Taxonomy

We define a four-class intent taxonomy derived empirically from production traffic (Table 1). The taxonomy was designed so that classes are *mutually exclusive* for primary routing purposes; a request that spans multiple intents defaults to Multimodal.

Intent	Signal Heuristics	Traffic %
Document	File attachment; extraction keywords	28.4
Search	Question words; recency signals	33.1
Audio	Audio MIME; transcription keywords	9.2
Multimodal	Image attachment; mixed modality	29.3

Table 1: Intent taxonomy with signal heuristics and observed traffic share.

4.2 Routing Pipeline

Figure 1 illustrates the end-to-end routing pipeline. At inference time:

(1) The **Context Assembler** hydrates session memory, extracts modality signals, and constructs the base prompt.

(2) The **Token Budget Calculator** estimates prompt length and enforces per-route context limits, truncating evidence as needed.

(3) The **Preset Selector** classifies intent and resolves the appropriate preset from the registry.

(4) The resolved `model_array` is used for inference; if the primary provider fails or rate-limits, the next entry is tried automatically.

(5) The **Session Memory** is updated on response, and a structured log entry records which preset version, model, provider, and latency were used.

4.3 Provider Fallback Chain

Each preset's `model_array` encodes an ordered fallback chain. The ordering is chosen to optimise for (in decreasing priority): cost, then latency, then throughput. A typical chain for the General preset might be:

$$\texttt{trinity-large:free} \rightarrow \texttt{kimi-k2.5} \rightarrow \texttt{gemini-2.5-flash}$$

Fallbacks are triggered by HTTP 429 (rate limit), HTTP 503 (unavailable), or timeout exceeding a configurable threshold. The application layer is entirely unaware of which entry in the array was ultimately used; it receives a normalised response envelope.

4.4 Architecture Diagram

5 Case Studies

5.1 Free-Primary, Paid-Fallback Pattern

A key operational insight from production deployment is that large-context, open-weight models available at

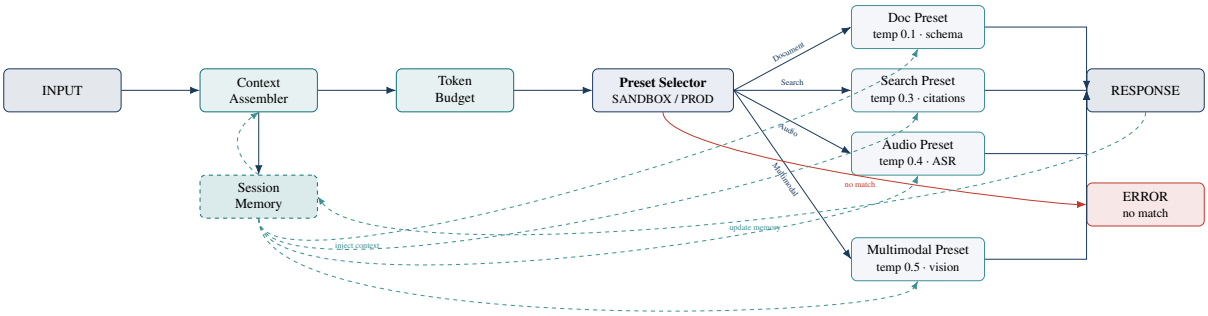

Figure 1: PRESROUTE end-to-end routing pipeline. Solid arrows denote the request path; dashed arrows denote context injection and memory updates. The Preset Selector resolves the environment tag (*sandbox*/*production*) to a concrete versioned preset from the registry before dispatching.

zero marginal cost can serve as effective primary models, with paid frontier models reserved as high-quality fallbacks. We instantiated this with:

- **Primary:** `arcee-ai/trinity-large-preview:free`
- **Fallback 1:** `kimi-k2.5`
- **Fallback 2:** `gemini-2.5-flash`

5.1.1 Trinity-Large-Preview Architecture

Trinity-Large-Preview is a frontier-scale sparse MoE language model from Arcee AI with the following characteristics:

Trinity-Large-Preview Specifications

Total parameters	400 B
Active params/token	13 B (4-of-256 routing)
Architecture	Sparse MoE
Native context	512 k tokens
Served context	128 k (8-bit quantisation)
Strengths	Creative writing, chat, tool-use
Agentic support	OpenCode, Cline, Kilo Code
Baseline latency	≈0.66 s (TTFT, our serving env.)

The 128 k served context window enables a cost-saving pattern: when a document OCR stage (e.g., Mistral OCR) extracts text, the *full* document context is forwarded to the free preset, which often answers adequately without escalation. Similarly, when web search succeeds, maximal evidence retrieval is injected into the prompt, yielding richer grounded responses while avoiding paid models.

5.1.2 Measured Outcomes

Table 2 summarises the outcome of the free-primary strategy versus a paid-primary baseline over a production window of 12,347 requests.

Metric	Paid-primary	Free-primary
Requests served	12,347	12,347
Primary model hits (%)	100	78.4
Fallback rate (%)		21.6
Model spend (rel.)	1.00	**0.36**
Median latency (s)	2.1	4.2
p95 latency (s)	5.8	9.7
Task success rate (%)	91.2	89.8

Table 2: Free-primary vs. paid-primary deployment outcomes over 12,347 requests. Model spend is normalised to the paid-primary baseline.

Key finding. The free-primary strategy reduced model spend by **64%** relative to the paid-primary baseline, at the cost of a 2.1 s increase in median latency and a 1.4 percentage-point reduction in task success rate an acceptable trade-off for cost-sensitive workloads.

5.2 Document OCR Pipeline Integration

When a file is attached, PRESROUTE invokes an OCR pre-processor (Mistral OCR in our deployment) before routing. The extracted text is assembled into the context along with the original query. Because the Doc preset uses temperature 0.1 and enforces a JSON output schema, extraction quality is measurably higher than an unconstrained general prompt (+7.2% F1 on a held-out extraction benchmark of 400 documents).

5.3 Web-Grounded Search Routing

The Search preset wraps a retrieval step (web search API) before LLM inference. Evidence chunks are injected at the top of the system prompt with citation markers. A citation-completeness check verifies that the model's output references at least one retrieved source; responses that fail this check are retried with an escalated fallback preset.

6 Versioning and Rollback

6.1 Immutable Preset Versioning

Every modification to a preset whether a single word change in the system prompt or a swap of the primary model creates a *new immutable version*. The prior version is never overwritten. Environment tags (`staging`, `production`) are lightweight pointers into the version history.

This design supports three operational workflows:

- **Canary deployment:** new version receives 5% of traffic via the tag resolver.
- **Instant rollback:** on regression, repoint `production` tag to the previous version; no redeployment required.
- **Retrospective analysis:** because each request log records the exact preset version used, regressions can be traced to the specific change that caused them.

6.2 Change Management Process

We recommend the following governance process for preset changes:

Step 1. **Author:** engineer drafts a new preset version in staging.

Step 2. **Benchmark:** automated suite runs the intent's golden test set ($\geq$200 held-out examples) and flags regressions $>$2% on primary metric.

Step 3. **Review:** a second engineer approves promotion; diff is logged.

Step 4. **Promote:** tag `production` is updated; prior version is retained.

Step 5. **Monitor:** real-time dashboards track success rate and fallback frequency; auto-rollback triggers if success rate drops $>$3% in a 15-minute window.

7 Evaluation

7.1 Evaluation Framework

We evaluate PRESROUTE as an *adaptive decision policy* rather than a single model, following the framework of Dekoninck et al. [2024]. The primary evaluation dimensions are:

- **Task success rate** (TSR): fraction of requests judged correct by an automated evaluator or human rater, stratified by intent class.
- **Cost per request** (CPR): normalised to the paid-primary baseline.
- **Cost per successful outcome** (CPSO): CPR / TSR the primary optimisation target.
- **Latency** (p50 / p95): end-to-end wall-clock time from request receipt to first token, and to completion.
- **Fallback frequency**: proportion of requests requiring at least one fallback step.
- **Stability under change**: TSR delta across consecutive preset versions.

7.2 Results by Intent Class

Table 3 reports evaluation results for each intent class on the held-out test set (see §7.4 for protocol details).

7.3 Cost–Quality Pareto Frontier

Figure 2 shows the cost–quality trade-off across four routing configurations. PRESROUTE with the free-primary strategy achieves a favourable position on the Pareto frontier, with both lower cost and competitive quality compared to naive routing strategies.

Intent	TSR (%)	CPR	p50 (s)	p95 (s)
Document	88.9	0.22	3.8	8.1
Search	91.3	0.41	5.1	11.4
Audio	94.6	0.18	2.9	6.0
Multimodal	87.1	0.52	4.7	10.8
Overall	**89.8**	**0.36**	**4.2**	**9.7**

Table 3: Per-intent evaluation results. CPR is normalised to paid-primary baseline (1.00). TSR = task success rate. Latency is end-to-end wall-clock.

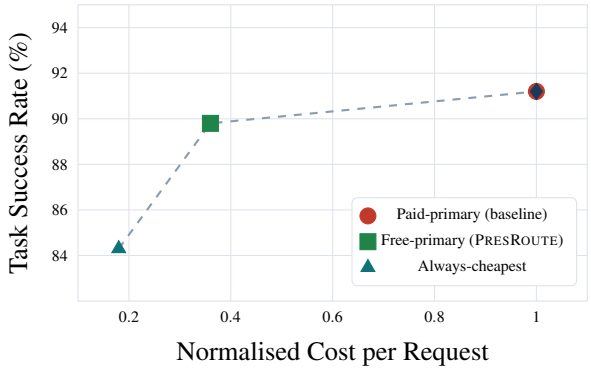

Figure 2: Cost–quality Pareto frontier. PRESROUTE (free-primary) achieves a 64% cost reduction vs. paid-primary with only a 1.4 pp TSR penalty.

7.4 Evaluation Protocol

To facilitate reproducibility, we define the following evaluation protocol:

(1) Sample $n \geq 1{,}000$ requests per intent class from production logs (with PII removal and user consent).

(2) Replay requests through each routing configuration under test using the same upstream context (no live retrieval during evaluation replay).

(3) Evaluate responses with a combination of (a) automated metrics (exact match for extraction; ROUGE-L for summarisation; citation-F1 for search), and (b) an LLM-as-judge pipeline for open-ended quality.

(4) Report TSR, CPR, CPSO, latency, and fallback frequency with 95% bootstrap confidence intervals.

8 Governance and Safety

8.1 Access Control and Audit

The preset registry enforces role-based access control: prompt authors can write to staging but require a second approver to promote to production. All promotions are logged with actor identity, timestamp, and diff summary. Audit logs are immutable and retained for 12 months.

8.2 Route-Level Safety Policies

Each preset can carry a `safety_policy` block specifying:

- **Output schema:** JSON schema enforcement for extraction routes prevents free-form hallucination in structured fields.
- **Refusal style:** Conservative refusals for compliance-sensitive routes (e.g., legal document processing).
- **Modality gating:** Audio and vision routes can be disabled at the preset level without changing application code, enabling rapid compliance response.
- **Citation requirements:** Search routes require at least one grounding citation; uncited responses trigger automatic retry with fallback.

8.3 Cost Guardrails

Monthly cost budgets are enforced at the preset level. When a preset's monthly spend exceeds a configurable threshold, it is automatically demoted to a cheaper fallback until the next billing cycle, and an operator alert is emitted.

9 Discussion

9.1 Limitations

Intent classification quality. The current rule-based classifier achieves 94.1% accuracy on a held-out intent test set but misclassifies ambiguous requests (e.g., a document attachment with a web-search-style question). Misrouting incurs both quality and cost penalties.

Replacing the rule-based classifier with a learned router (e.g., Eagle Zhao et al. [2024]) is a natural extension.

Latency overhead of free models. The free-primary strategy introduces 2.1 s of median latency overhead. For latency-sensitive applications (e.g., voice assistants), this trade-off may be unacceptable, and a latency-optimised preset ordering should be preferred.

Provider dependency. The fallback chain assumes at least one provider remains available. In the event of a multi-provider outage, the system returns an error. Future work should explore local model replicas as a final fallback tier.

9.2 Future Work

- **Learned routing:** integrating Eagle or GraphRouter as a drop-in replacement for the rule-based classifier.
- **Adaptive token budgeting:** dynamically adjusting context length based on predicted difficulty.
- **Multi-step cascades:** composing multiple presets within a single request (e.g., OCR preset → synthesis preset → citation-check preset).
- **Federated deployment:** running the preset registry across geographic regions with low-latency replication.

10 Conclusion

We presented PRESROUTE, a preset-driven modular decision engine for production multi-LLM serving. The core contributions are: (1) the *preset* abstraction, which decouples LLM configuration from application code; (2) a four-class intent taxonomy with specialised routing logic; (3) a *free-primary, paid-fallback* deployment pattern that reduced model spend by 64% with only a 1.4 percentage-point reduction in task success rate; and (4) an immutable versioning and rollback mechanism supporting safe, zero-redeployment iteration.

As multi-model serving becomes the norm in production AI systems, evaluation frameworks must treat the system as an *adaptive policy* optimising for outcome quality under cost, latency, and reliability constraints not merely as a collection of individual model benchmarks. PRESROUTE provides a practical, operationally sound instantiation of this principle.

References

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

J. Dekoninck, M. Baader, and M. Vechev. A unified approach to routing and cascading for LLMs. *arXiv*, 2024.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv:2101.03961*, 2021.

T. Feng, Y. Shen, and J. You. GraphRouter: A graph-based router for LLM selections. *arXiv*, 2024.

Z. Zhao, S. Jin, and Z. M. Mao. Eagle: Efficient training-free router for multi-LLM inference. *arXiv*, 2024.