OmniMind AI : Coordinating Diverse Large Language Models for Context-Aware Response Generation

Dr. Himanshu V. Taiwade[1], Uday Sanjay Wahile[2], Sumit Harish Dhale[3], Gaurav Ramesh Thakre[4], Yogesh Dinesh Yadav[5], Ritik Minesh Dhodharmal[6]
[1,2,3,4,5,6] Ritik Minesh Dhodharmal, *Department of Computer Science and Engineering, Priyadarshini College of Engineering, Nagpur, Maharashtra, India*

***Abstract*—No single large language model (LLM) is the best choice for every kind of question — one model may reason through mathematics more reliably, another may write more naturally, and a third may hold more current factual knowledge. OmniMind AI is proposed as a web platform that sends a single user query to several LLMs in parallel, scores and compares their responses, and merges the strongest parts of each into one polished answer rather than leaving the user to open several separate chat tools and compare them manually. This paper reviews recent research on multi-LLM systems — routing, cascading, ensembling and debate-based approaches — and studies how their design choices around cost, latency and answer quality apply to a platform built on the MERN stack (MongoDB, Express.js, React.js and Node.js). Techniques such as preference-based routing, pairwise ranking with generative fusion, and layered aggregator models are compared for their suitability to a real-time, browser-based product. The paper also lays out the proposed OmniMind AI architecture, its intended advantages and application areas, and the open challenges — latency from parallel API calls, response-fusion quality, and per-query cost — that any production-grade multi-LLM aggregator has to address.**

***Index Terms*—Multi-LLM Systems, Response Aggregation, LLM Routing, Ensemble Learning, MERN Stack, Prompt Orchestration, Large Language Models**.

I. INTRODUCTION

Large language models have moved from research demos to everyday tools in barely a couple of years, and it is now common for a single person to keep tabs on more than one of them — switching to a different chatbot when the first one gives a shallow answer, misses a recent event, or fumbles a calculation. This habit is a practical admission that no one model consistently wins across every task; a model tuned heavily for conversation may not be the strongest at code or arithmetic, and a model with a fast, lightweight architecture may trade away some depth of reasoning for speed. OmniMind AI is conceived around this observation. Instead of asking a person to open several tabs and eyeball the differences, the platform is designed to query multiple LLMs behind one interface, compare what comes back, and hand over a single response that draws on whichever model actually answered the question well.

The idea of combining multiple LLMs, rather than betting on one, is already an active research direction. Ong et al. showed that a lightweight router trained on preference data can decide, per query, whether a cheap or an expensive model is enough — cutting reliance on costlier models substantially without hurting answer quality [1]. Jiang, Ren and Lin took a different route with LLM-Blender, which first ranks candidate answers pairwise and then fuses the best of them into a single generated response rather than simply picking a winner [2]. A recent survey of this whole area confirms that most methods fall into routing before inference, combining outputs during generation, or fusing complete responses after inference finishes [17], and this paper leans mainly on the first and third of those groups since they fit a real-time product better than token-level fusion.

Because OmniMind AI is meant to run as an interactive web application, the choice of implementation stack matters almost as much as the aggregation logic itself. The MERN stack — MongoDB for storing conversation history and per-model metadata, Express.js and Node.js for handling concurrent outbound calls to several LLM providers, and React.js for a responsive front end that can stream

partial answers — is well suited to this kind of I/O-heavy, real-time orchestration work, since Node's non-blocking model handles many simultaneous API calls without the request queue backing up [20]. This review therefore looks at both sides of the problem: the machine-learning literature on combining LLM outputs, and the practical architecture needed to bring that combination to a live, low-latency product.

II. PROBLEM IDENTIFICATION

- Fragmented Experience: Users who want a well-rounded answer must currently open several separate LLM applications and compare the outputs by hand, which is slow and easy to get wrong.
- Single-Model Bias: Any one LLM carries the strengths and blind spots of its own training data and architecture, so relying on a single model risks propagating its specific weaknesses into every answer.
- Latency of Naive Aggregation: Simply calling several LLM APIs one after another and waiting for all of them multiplies response time, which works against a responsive real-time interface.
- Weak Fusion Logic: Showing several raw answers side by side is not the same as producing one coherent, deduplicated response; genuine fusion needs a scoring and merging step.
- Cost Management: Each additional LLM called per query adds to the API bill, so an aggregation platform needs some way to decide when calling extra models is actually worth it.

III. LITERATURE SURVEY

A) Literature Reviews

On the routing side, I. Ong et al. [1] (2025) proposed RouteLLM, a framework that trains lightweight router models on human preference data to dynamically choose between a stronger, costlier LLM and a weaker, cheaper one for each incoming query. Their routers cut inference costs by more than half in some settings without a meaningful drop in response quality, and the learned routing policy transferred well even when the underlying strong and weak models were swapped out.

Rather than choosing a single model, D. Jiang et al. [2] (2023) introduced LLM-Blender, which combines a pairwise ranking module (PairRanker) that compares candidate responses two at a time using cross-attention, with a generative fusion module (GenFuser) that merges the top-ranked outputs into a single improved response. To support large-scale evaluation, they also released MixInstruct, a benchmark built from multiple instruction datasets.

Extending the idea of layered collaboration, J. Wang et al. [3] (2025) presented Mixture-of-Agents (MoA), a framework in which several LLMs are organized into layers, with each layer's agents conditioning their responses on the outputs of the agents in the layer before it. This progressive refinement allowed an MoA built entirely from open-source models to exceed GPT-4 Omni on the AlpacaEval 2.0 leaderboard.

Addressing the cost side of the problem, L. Chen et al. [4] (2024) introduced FrugalGPT, which examines the highly uneven pricing of commercial LLM APIs and proposes three complementary cost-reduction strategies: prompt adaptation, LLM approximation, and LLM cascades. Their cascade implementation matched GPT-4-level accuracy at a fraction of the cost, or improved on GPT-4's accuracy at equal cost.

S. Chen et al. [5] (2024) observed that many existing routers struggle in cases where several candidate LLMs are all reasonably well-suited to a query, and proposed RouterDC, a query-based router trained with two contrastive objectives, one that pulls a query's embedding toward strong-performing models and away from weak ones, and another that groups similar queries together for more stable training. This dual-contrastive design outperformed both individual top models and prior routing methods on in- and out-of-distribution tasks.

T. Feng et al. [6] (2025) took a structural approach with GraphRouter, which represents tasks, queries, and candidate LLMs as nodes in a heterogeneous graph and predicts the expected effect and cost of each possible query-model pairing as an edge-prediction problem. Because new LLMs can simply be added as new nodes, GraphRouter generalizes to previously unseen models without retraining, delivering a

substantial performance improvement over prior routers.

R. Jin et al. [7] (2025) proposed RadialRouter, which uses a lightweight transformer with a radial architecture (RadialFormer) to represent the relationship between a query and the pool of candidate LLMs, trained with a combined KL-divergence and query-query contrastive objective. On RouterBench, RadialRouter outperformed prior routing methods under both balanced and cost-sensitive routing scenarios while adapting well to changes in the available model pool.

A separate line of work focuses on combining models through structured reasoning rather than routing to a single one. Y. Du et al. [8] (2023) proposed a multiagent debate framework in which several instances of a language model propose answers and then critique one another's reasoning over multiple rounds before converging on a final response, showing measurable gains in mathematical and strategic reasoning tasks along with fewer factual errors. In a related but single-model approach, X. Wang et al. [9] (2023) introduced self-consistency, a decoding strategy that samples multiple independent chain-of-thought reasoning paths for the same query and selects the final answer by majority agreement across paths rather than committing to a single greedy decoding run, yielding sizable accuracy gains on arithmetic and commonsense reasoning benchmarks.

K. Lu et al. [10] (2023) proposed Zooter, a reward-guided routing approach that distills reward-model judgments on training queries into a lightweight routing function capable of directing each query to the LLM with the most relevant expertise, avoiding the heavy computational cost of ranking every candidate's output with a reward model at inference time.

Moving beyond output-level combination, F. Wan et al. [11] (2024) explored knowledge fusion of LLMs with FuseLLM, which transfers the collective knowledge of several structurally different source models into a single target model through lightweight continual training on their generative output distributions, rather than through weight merging, which is typically restricted to models sharing an identical architecture. X. Lu et al. [12] (2024) proposed a related but simpler technique called Blending, which integrates the outputs of several small-to-moderate chat models and showed that blending just three models in the 6–13B parameter range could match or exceed the performance of a much larger proprietary model.

C. Mavromatis et al. [13] (2025) introduced PackLLM, a test-time fusion method that estimates each candidate LLM's expertise on a given prompt by measuring its perplexity, then solves a lightweight optimization problem to weight each model's contribution accordingly, allowing new LLMs to be incorporated at inference time without any additional training.

J. Chen et al. [14] (2024) proposed ReConcile, a round-table framework in which diverse LLM agents engage in multiple rounds of discussion, exchange confidence-weighted explanations, and are guided toward consensus using demonstrations of how humans revise incorrect answers, an approach that improved reasoning performance both for individual agents and for the group as a whole.

H. Zhang et al. [15] (2025) extended routing beyond a single dispatch decision with Router-R1, a reinforcement-learning framework in which the router itself is an LLM that interleaves internal reasoning steps with explicit routing actions across multiple rounds, accumulating responses from different models into its evolving context and optimizing a reward that balances task accuracy against inference cost.

Questioning a core assumption behind mixture-based ensembling, W. Li et al. [16] (2025) examined whether combining outputs from different LLMs is actually necessary for the gains seen in Mixture-of-Agents-style methods. Their proposed Self-MoA, which aggregates multiple outputs from only the single best-performing model rather than mixing different models, outperformed standard MoA across a range of benchmarks, suggesting that model diversity is not always the source of ensembling gains.

Finally, Z. Chen et al. [17] (2025) provided the first systematic survey of the LLM ensemble literature, organizing existing methods into ensemble-before-inference, ensemble-during-inference, and ensemble-

after-inference categories, and outlining open research directions across routing, fusion, and reward-guided aggregation approaches — a taxonomy that closely mirrors the structure of the present work.

B) Literature Summary

- Multi-LLM research broadly splits into three families: routing (send the query to one suitable model), cascading (escalate through models by cost), and ensembling/fusion (combine several answers into one).
- Routing methods such as RouteLLM, RouterDC, GraphRouter, Router-R1 and RadialRouter focus on picking the right model quickly and cheaply, which suits low-latency systems [1],[5],[6],[7],[15].
- Fusion-based methods such as LLM-Blender, Mixture-of-Agents and knowledge-fusion approaches produce noticeably better answers than any single model but add extra generation rounds and therefore extra latency and cost [2],[3],[11].
- Cost-aware cascading, as in FrugalGPT, shows that most queries do not need the most expensive model, which is directly useful for controlling API spend in a live product [4].
- Debate-style, self-consistency and blending approaches improve reliability on reasoning-heavy queries but are generally too slow, or too model-internal, for a chat-style, real-time interface [8],[9],[12],[14].
- Recent surveys confirm this three-way split and note that very little of the literature is evaluated inside an actual deployed web application, leaving real-time UX constraints under-explored [17].

C) Research Gap

- Most routing and ensembling research is evaluated offline on fixed benchmarks, not inside a live, interactive product where users expect an answer within a few seconds.
- Few studies combine routing (deciding which models to call) with fusion (merging what they return) in a single, cost-aware pipeline suited to a browser-based interface.
- There is limited published work on how a full-stack web architecture, such as MERN, should be structured to run several concurrent LLM API calls without degrading responsiveness.
- Persisting and reusing conversation-level context across multiple LLM providers, rather than a single provider, is not well addressed in the existing routing literature.
- Balancing per-query API cost against answer quality in a way that is visible and controllable for the end user remains an open, mostly product-side problem.

IV. RESEARCH METHODOLOGY

A) Criteria for selecting this study:

- Relevance: Studies were chosen because they directly address combining, routing, or ensembling multiple large language models.
- Recency: Preference was given to 2023–2025 publications so the review reflects the current generation of routing and fusion techniques.
- Methodological Diversity: Preference-based routing, contrastive routing, graph-based routing, generative fusion, layered mixtures, distribution fusion, and debate-based methods were all included.
- Deployment Relevance: Studies that reported latency, cost, or scalability figures were prioritized, since these matter directly to a real-time platform.
- Research Quality: Peer-reviewed venues (ACL, ICLR, NeurIPS, EMNLP, COLM) and well-cited arXiv preprints were considered.
- Comparative Value: Selected studies differ meaningfully in technique, benchmark, and cost/latency trade-offs so the comparison table below is informative rather than repetitive.

B) Method of analysis:

- Literature Collection: Recent papers on LLM routing, cascading, ensembling and debate were gathered from conference proceedings and arXiv.
- Thematic Classification: Papers were grouped into routing-based, fusion-based, and debate/self-consistency based approaches.
- Cost-Latency Analysis: Each method was examined for how it affects the number of model calls per query, since this drives both response time and API cost.

- Architecture Mapping: Findings were mapped against a MERN-based system to judge which techniques are realistic for a browser-facing, real-time platform.
- Gap Synthesis: Findings were combined to identify what a practical multi-LLM aggregation platform such as OmniMind AI still needs to solve.

V. HIGHLIGHTING TRENDS, ADVANCEMENTS, AND CHALLENGES

A) Trends

- Research is shifting from a single best-model mindset toward systems that treat multiple LLMs as complementary resources.
- Routers are increasingly learned from data (preference pairs, contrastive embeddings, graph structure) rather than hand-coded rules [1],[5],[6].
- Cost-awareness is becoming a first-class design goal alongside accuracy, not an afterthought [4],[15].
- Fusion techniques that merge multiple answers into one, at both the response and the probability-distribution level, are gaining attention as a way to beat single-model ceilings [2],[3].

B) Advancements

- Preference-trained routers such as RouteLLM cut reliance on expensive models while keeping answer quality close to the strongest model available [1].
- Pairwise ranking and generative fusion, as in LLM-Blender, show that a combined answer can genuinely exceed any single contributing model [2].
- Graph-based and contrastive routers adapt more gracefully when new models are added to a pool, which matters for a platform that keeps adding providers [5],[6].
- Reinforcement-learning-based routers such as Router-R1 let the router itself reason across multiple rounds before deciding which model to call next [15].
- Lightweight structured routers such as RadialRouter demonstrate that routing decisions themselves do not have to be computationally expensive [7].

C) Challenges

- Calling several LLM APIs per query multiplies latency and cost compared with a single-model system, which works against a responsive chat interface.
- Fusing multiple answers well is harder than simply picking the best one; naive concatenation of answers reads as repetitive rather than genuinely combined.
- Different LLM providers return responses in different formats, at different speeds, and with different rate limits, complicating orchestration.
- Recent work shows that mixing different LLMs is not always better than reusing one strong model repeatedly, so a platform must be selective about when to fuse at all [16].
- Keeping per-user cost predictable while still calling multiple models intelligently remains an open, largely product-level problem.

VI. DISCUSSION

A) Synthesis of findings from literature

- The literature confirms that no single LLM dominates across all query types, which supports the core premise of OmniMind AI.
- Routing methods are the more practical fit for a real-time interface, since they add at most one extra lightweight decision step before a normal model call.
- Fusion methods produce noticeably better final answers but are costlier, so they are better reserved for queries flagged as complex or ambiguous rather than applied to every request.
- Cost-aware cascading is a natural complement to routing, letting the platform escalate to stronger (and more expensive) models only when needed.
- Almost none of the reviewed work discusses the front-end and backend engineering needed to make these techniques feel instantaneous to an end user, which is the gap this project's architecture aims to fill.

B) Methodology for future research directions

Proposed System:

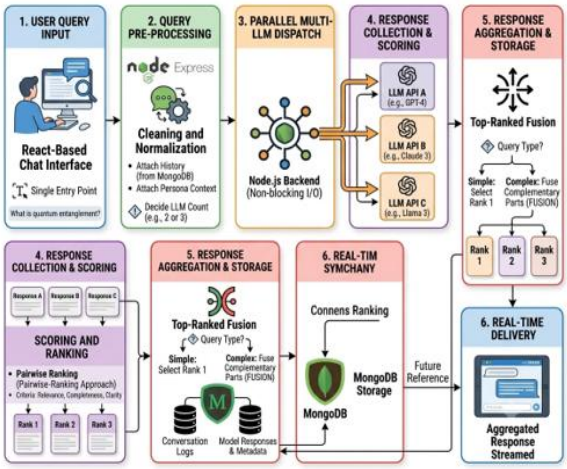

Figure 1. Proposed System

- User Query Input: The user types a question or prompt into the React-based chat interface, which is the single entry point regardless of how many LLMs will eventually be consulted.
- Query Pre-processing: The Node.js/Express backend cleans and normalizes the prompt, attaches relevant conversation history retrieved from MongoDB, and decides how many models to consult based on query complexity.
- Parallel Multi-LLM Dispatch: The backend fires concurrent, asynchronous API calls to the connected LLM providers using Node's non-blocking I/O so the calls run side by side rather than one after another.
- Response Collection: As each provider responds, its output is captured along with metadata such as response time and a rough confidence signal, and stored temporarily for comparison.
- Scoring and Ranking: A lightweight ranking step, inspired by pairwise-ranking approaches in the literature, scores the candidate responses on relevance, completeness, and clarity.
- Response Aggregation: The top-ranked responses are merged — either by selecting the strongest single response for simple queries or by fusing complementary parts of several responses for more complex ones — into one final answer.
- Real-Time Delivery: The aggregated response is streamed back to the React front end, and MongoDB stores the conversation, the individual model responses, and the final aggregated answer for future reference.

VII. ADVANTAGES

- Unified Interface: Removes the need to open multiple separate chat tools to compare answers.
- Balanced Answers: Draws on more than one model's strengths instead of depending on a single model's blind spots.
- Cost Control: Routing and cascading logic can limit expensive model calls to queries that genuinely need them.
- Real-Time Response: A MERN-based, asynchronous backend is built to keep multi-model orchestration fast enough for interactive use.
- Extensible Design: New LLM providers can be added to the pool without restructuring the whole platform.
- Transparent Comparison: Users can optionally view the individual model responses behind the final aggregated answer.

VIII. APPLICATIONS

- Research and Study Assistance: Offers a more balanced answer for students and researchers comparing perspectives across models.
- Content Drafting: Blends style and factual strengths of different models for writing, editing and summarizing tasks.
- Technical and Coding Help: Cross-checks code explanations or debugging suggestions across models before presenting a final answer.
- Customer Support Tools: Can be integrated where a more reliable, cross-checked automated answer is valuable.
- Decision Support: Useful where an aggregated, less biased answer is preferable to a single model's output.
- Educational Platforms: Can highlight where different models agree or disagree, which is itself informative for learners.

IX. CONCLUSION

OmniMind AI is framed in this review as a response to a problem that has become familiar to anyone who

regularly uses more than one LLM: no single model is reliably the best choice for every query, yet comparing several models by hand is slow and inconvenient. The reviewed literature shows a clear, active research effort around this exact problem — learned routers that send a query to the right model, cascading systems that escalate only when necessary, and fusion techniques that merge multiple answers into one improved response. Each family of methods brings a different trade-off between answer quality, latency and cost, and a production platform cannot simply adopt the most accurate technique from a benchmark paper without also asking whether it will still feel fast inside a browser. Building OmniMind AI on the MERN stack is proposed specifically to address that engineering half of the problem, since Node.js's non-blocking model is well matched to firing off several concurrent LLM API calls without stalling the interface. Future work on the platform should focus on combining lightweight routing with selective fusion — calling extra models only for queries that plausibly benefit from it — along with keeping per-user cost visible and predictable, since the literature suggests that this combination, rather than any single technique alone, is what practical multi-LLM aggregation will need.

REFERENCES

[1] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "RouteLLM: Learning to route LLMs with preference data," in Proc. 13th Int. Conf. Learning Representations (ICLR), 2025.

[2] D. Jiang, X. Ren, and B. Y. Lin, "LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion," in Proc. 61st Annu. Meeting Assoc. Comput. Linguistics (ACL), Toronto, Canada, 2023, pp. 14165–14178.

[3] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou, "Mixture-of-agents enhances large language model capabilities," in Proc. 13th Int. Conf. Learning Representations (ICLR), 2025, arXiv:2406.04692

[4] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," Trans. Mach. Learn. Res., 2024, arXiv:2305.05176.

[5] S. Chen, W. Jiang, B. Lin, J. Kwok, and Y. Zhang, "RouterDC: Query-based router by dual contrastive learning for assembling large language models," in Proc. 38th Conf. Neural Information Processing Systems (NeurIPS), 2024.

[6] T. Feng, Y. Shen, and J. You, "GraphRouter: A graph-based router for LLM selections," in Proc. 13th Int. Conf. Learning Representations (ICLR), 2025, arXiv:2410.03834.

[7] R. Jin, P. Shao, Z. Wen, J. Wu, M. Feng, S. Zhang, and J. Tao, "RadialRouter: Structured representation for efficient and robust large language models routing," in Findings Assoc. Comput. Linguistics: EMNLP 2025, Suzhou, China, 2025, pp. 14587–14600.

[8] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," arXiv preprint arXiv:2305.14325, 2023.

[9] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in Proc. 11th Int. Conf. Learning Representations (ICLR), 2023.

[10] K. Lu, H. Yuan, R. Lin, J. Lin, Z. Yuan, C. Zhou, and J. Zhou, "Routing to the expert: Efficient reward-guided ensemble of large language models," arXiv preprint arXiv:2311.08692, 2023.

[11] F. Wan, X. Huang, D. Cai, X. Quan, W. Bi, and S. Shi, "Knowledge fusion of large language models," in Proc. 12th Int. Conf. Learning Representations (ICLR), 2024.

[12] X. Lu, Z. Liu, A. Liusie, V. Raina, V. Mudupalli, Y. Zhang, and W. Beauchamp, "Blending is all you need: Cheaper, better alternative to trillion-parameters LLM," arXiv preprint arXiv:2401.02994, 2024.

[13] C. Mavromatis, P. Karypis, and G. Karypis, "Pack of LLMs: Model fusion at test-time via perplexity optimization," in Proc. 1st Conf. Language Modeling (COLM), 2025.

[14] J. Chen, S. Saha, and M. Bansal, "ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs," in Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (ACL), Bangkok, Thailand, 2024, pp. 7066–7085.

[15] H. Zhang, T. Feng, and J. You, "Router-R1: Teaching LLMs multi-round routing and

aggregation via reinforcement learning," in Proc. 39th Conf. Neural Information Processing Systems (NeurIPS), 2025, arXiv:2506.09033.

[16] W. Li, Y. Lin, M. Xia, and C. Jin, "Rethinking mixture-of-agents: Is mixing different large language models beneficial?," arXiv preprint arXiv:2502.00674, 2025.

[17] Z. Chen, X. Lu, J. Li, P. Chen, Z. Li, K. Sun, Y. Luo, Q. Mao, M. Li, L. Xiao, D. Yang, X. Huang, Y. Ban, H. Sun, and P. S. Yu, "Harnessing multiple large language models: A survey on LLM ensemble," arXiv preprint arXiv:2502.18036, 2025.