The Over-Complication Anomaly: Quantitative Evaluation of the Dynamic Hierarchical Multi-Agent Framework for Large Language Models

Undergraduate Research Thesis

Presented in Partial Fulfillment of the Requirements for the Degree B.S. Computer Science and Engineering with Research Distinction in the College of Engineering at The Ohio State University

By

Harish Mukkapati

Undergraduate Program in Computer Science and Engineering

The Ohio State University

2025

Thesis Committee

Dr. Andrew Perrault, Advisor

Dr. Sachin Kumar

Copyrighted by

Harish Mukkapati

2025

Abstract

Current AI frameworks struggle with scalability, validation, and accuracy, as most rely on single-model execution, lack structured verification, and use global context memory, leading to error propagation. This thesis implements and quantifies the performance vs. cost trade-off of the Dynamic Hierarchical Multi-Agent Framework (DHMAF), a system employing a hierarchy of Prime Meta Agent, Meta Agents, and Checker Agents to recursively instantiate, execute, and validate sub-tasks. The framework was benchmarked against established AI tests using the Gemini 1.5 Flash model, an evaluation covering 14,725 questions that consumed approximately 2.5 million API requests and cost $253. The results reveal a significant "over-complication anomaly". While the framework improved scores on multi-step reasoning tasks (GSM8K +4.7%, MMLU-Pro +7.28%), it caused a severe performance decrease of -12.19% on the MMLU benchmark, focusing more on simple fact recall. These findings demonstrate that hierarchical validation frameworks, while providing modest benefits for multi-step reasoning tasks, can be actively detrimental and cost-prohibitive for generalist applications, providing a quantitative analysis of this critical trade-off.

Acknowledgments

I would like to thank Dr. Andrew Perrault, my research advisor, for his experience and advice on this project. I would also like to thank Pawan Medidi for his invaluable assistance on this project.

Vita

2022-Present……Ohio State University

December 2024 – March 2025……..…….Software Engineer Intern at Crown Equipment Corporation

June 2025 – August 2025…….……Software Engineer Intern at CoverMyMeds

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

List of Figures

Chapter 1. Introduction

Artificial intelligence (AI) increasingly relies on multi-agent architectures for complex tasks, yet existing frameworks remain constrained by static agent roles, limited adaptability, and a lack of structured validation. While some approaches focus on optimizing individual agents or structuring predefined multi-agent execution, neither fully addresses recursive instantiation agent coordination and scalable hierarchical validation.

This research proposes a Dynamic Hierarchical Multi-Agent Framework (DHMAF). The novelty of this architecture lies not in a single component, but in its specific three pillar synthesis designed to improve reliability while maintaining adaptability. The first pillar is Dynamic Recursive Instantiation. Agents can recursively generate a new child agent to decompose complex problems, creating a task-specific hierarchy at runtime. The second pillar is Multi-Tier Validation. A dedicated layer of "Micro" and "Macro" Checker Agents validates every agent's output before it is passed up the hierarchy, creating a self-correcting feedback loop. The third pillar is Strictly Isolated Agent Memory. To prevent the error propagation endemic to frameworks with shared context, each agent operates with its own independent memory. All communication is passed as explicit, validated results, not as a shared chat history.

This thesis provides the first quantitative analysis of this specific architecture's performance, investigating whether this synthesis is beneficial or if its complexity leads to a detrimental "over-complication anomaly". The findings reveal that the latter is true. The DHMAF architecture introduces a severe performance decrease on generalist tasks (-12.19% on MMLU) in exchange for only modest improvements on benchmarks benefiting from multi-step reasoning (+7.28% on MMLU-Pro and +4.7% on GSM8K). This research, therefore, quantifies this unfavorable trade-off, highlighting the costs and risks of applying complex, hierarchical validation to general purpose models.

Chapter 2. Literature Review

2.1 The Evolving Landscape of Agentic AI

The development of powerful LLMs has led to a new approach in artificial intelligence: the use of Multi-Agent Systems (MAS). In this model, complex problems are solved not by a single model, but by a system of collaborating agents. A modern agentic AI system, as characterized by Derouiche et al., involves intelligent agents that exhibit "goal-directed autonomy, contextual reasoning, and dynamic multi-agent coordination" [3]. These systems aim to use the specialized capabilities of multiple agents to achieve the outcomes that might be beyond the reach of a single model.

Current agentic frameworks can be broadly categorized by their approach to two competing design philosophies: ensuring structured, reliable collaboration and enabling dynamic, adaptive problem solving. Structured frameworks with predefined roles and workflows offer predictability and are less prone to error propagation. On the other hand, effective problem solving often requires dynamism: the ability to adapt, create novel solutions, and decompose unforeseen challenges.

This chapter traces the evolution of agentic AI frameworks by focusing on this challenge. We will analyze how different architectural paradigms have attempted to balance structure and dynamism, from early single-agent iterative models to more complex, structured multi-agent collaborations. By examining the strengths and limitations of these approaches, this review will demonstrate the need for a new architecture that can effectively combine these two goals.

2.2 Classical Paradigms: Hierarchical Task Networks as a Benchmark

Before analyzing modern LLM-based agents, it is instructive to establish a conceptual benchmark from classical AI planning. Hierarchical Task Networks (HTNs) provide such a benchmark. As described in research by Mu et al., HTN planning is a paradigm where complex, abstract goals are recursively decomposed into a tree of smaller, more concrete sub-tasks [4].

The primary strength of this approach is its ability to provide a structured, goal-oriented method for solving complex problems. The HTN framework uses symbolic knowledge, formalized in a Hierarchical Domain Definition Language (HDDL), to guide exploration and reduce the problem space by creating a formal hierarchy of tasks [4]. This paradigm is relevant to the current review as it establishes the principle of dynamic, hierarchical decomposition. The classical approach, where a system can autonomously break down a novel, abstract task, will serve as a point of comparison for the more static, pre-defined architectures found in many modern multi-agent frameworks.

2.3 Single Agent Iterative Frameworks

The first major architectural pattern for LLM-based agents consists of a single agent operating in a continuous loop. Frameworks such as AutoGPT are primarily examples of this approach. The architecture is straightforward: a single LLM-powered agent is given a high-level goal and attempts to achieve it through a repeating cycle of

thought, action, and observation [5]. The agent reflects on the outcome of its previous action and generates a new plan to move closer to the goal.

While this paradigm was foundational in demonstrating the potential of autonomous agents, its limitations are significant. The lack of structured task delegation and validation layers leads to several inefficiencies. First, the agent is prone to getting "stuck" in loops, repeatedly attempting the same failed actions. Second, without a separate validation mechanism, errors in reasoning can propagate through subsequent cycles, leading to redundancies, bottlenecks, and a deviation from the intended goal. These frameworks lack the structure required for complex, multi-faceted tasks and are highly dependent on the single agent's ability to self-correct, which often fails in practice.

2.4 Automated Design and Optimization Frameworks

A more recent approach attempts to solve the problem of agent abstraction at a higher level of abstraction. Instead of focusing on the runtime of an agent, these frameworks focus on programmatically designing or optimizing the systems themselves. One such framework is shown in the paper: Automated Design of Agentic Systems (ADAS) [1]. ADAS uses a "meta agent" to programmatically discover, evaluate, and refine the architecture of other agents over time [1]. The system effectively learns and evolves more capable agent designs through an iterative design-test-refine loop. Similarly, Google DeepMind's AlphaEvolve is an evolutionary coding agent that iteratively improves algorithms by making direct changes to their code [6]. While ADAS

focuses on designing the architecture of agentic systems, AlphaEvolve focuses on optimizing the implementation of specific algorithms and has been used to discover novel solutions for scientific and computational problems [6] [other 6]. These evolutionary frameworks represent a novel approach to the problem of dynamism, moving it from runtime adaptation to design-time evolution. However, their focus does not address the limitations of real-time, parallel collaboration. Neither ADAS nor AlphaEvolve is designed for the orchestration of multiple, distinct agents working together simultaneously on a complex task. They lack the mechanisms for parallel multi-agent coordination and hierarchical task decomposition required for the collaborative problem solving addressed by this review.

2.5 Structured Multi-Agent Collaboration Frameworks

To address the limitations of unstructured single-agent systems, frameworks for structured multi-agent collaboration were introduced. These frameworks manage a team of agents with pre-assigned roles, similar to a human software development team. MetaGPT, for example, formalizes this by encoding Standard Operating Procedures into its agent roles (e.g., “Product Manager”, “Architect”, “Engineer”) to create an assembly line for its task execution [7]. It is important to note its memory architecture: while it does use a publish-subscribe mechanism for shared communication, its agents also possess independent memory [7]. Similarly, AutoAgents automates the generation of agents by confining them to a predefined collaboration structure [2].

More recent frameworks have also adopted hierarchical or specialized roles. Frameworks like CrewAI and LangGraph focus on orchestrating predefined roles, using role-based definitions or graph-based structures to manage the flow of information [8]. AgentOrchestra utilizes a central planning agent to decompose tasks and delegate them to specialized sub-agents in a two-tier architecture [9]. Mobile-Agent-E employs an explicit separation of high-level planning ("Manager" agent) from low-level execution and notably includes an "Action Reflector" for error verification [10].

While these structured frameworks successfully introduce predictability and mitigate the error propagation seen in single-agent loops, they do so by sacrificing dynamism. Their reliance on pre-defined roles and static workflows exemplifies what Derouiche et al. identify as a key limitation across the field: "rigid architectures" [3]. Agents in many of these systems cannot dynamically create new sub-agents or adapt to their roles on unforeseen task requirements. This static approach stands in stark contrast to the dynamic, recursive decomposition of the classical HTN paradigm. These frameworks do not solve the hierarchical decomposition problem; rather, they bypass it by enforcing a pre-built, static hierarchy.

2.6 Synthesizing the Literature: Identifying the Research Gap

The evolution of agentic frameworks reveals a complex landscape of design trade-offs. The field has progressed from unstructured, single-agent systems (Section 2.3), which suffer from significant error propagation to highly structured, multi-agent

collaboration models (Section 2.5) that solve for reliability by imposing “rigid architectures” [3].

While frameworks like MetaGPT address reliability through predefined roles and classical planners like HTNs have demonstrated recursive decomposition, event recent systems like Mobile-Agent-E and AgentOrchestra employ validation and planning layers [9][10]. A persistent research gap remains. No existing framework appears to synthesize three distinct architectural principles in tandem: (1) dynamic, recursive agent instantiation for adaptive problem-solving, (2) a multi-tier validation layer to enforce correctness at each step, and (3) strictly isolated agent memory to programmatically prevent the error propagation that plagues shared-context models.

The interplay of these three design choices, and their resulting impact on performance, cost, and reliability, remains unquantified. It is unknown if such a complex synthesis is beneficial, or if it creates an overly complicated system that is computationally expensive and detrimental to performance on simpler tasks.

Furthermore, other advanced approaches (Section 2.4) focus on design-time optimization (e.g., ADAS, AlphaEvolve) [1][6] rather than runtime operational challenges. This survey of the field demonstrates a clear need for a framework that can synthesize these two competing goals: the reliability of structured collaboration and the adaptability of dynamic, hierarchical decomposition.

2.7 Positioning the DHMAF

The Dynamic Hierarchical Multi-Agent Framework (DHMAF) introduced in this thesis is designed to implement and evaluate this specific three-pillar synthesis. It bridges the gap by uniquely combining: 1. Recursive Hierarchical Decomposition: Applying principles from HTN planning, agents can recursively instantiate new child agents to handle sub-tasks. 2. Multi-Tier Validation: A dedicated layer of "Micro" and "Macro" Checker Agents validates all outputs before they are passed up the hierarchy. 3. Isolated Agent Memory: As will be detailed in the Methodology each agent operates with an independent state to prevent the cross-contamination of errors.

This research is therefore not just an attempt to build a "better" framework, but to provide a quantitative investigation into the real-world consequences of this specific architecture. The following chapter details the DHMAF's methodology, which is then empirically tested to answer the core research question: What is the performance, reliability and computational cost trade-offs of this novel synthesis when applied to both complex reasoning and generalist benchmarks?

Chapter 3. Methodology

3.1 Core Principles of the DHMAF

To address the research gap identified in the previous chapter, this thesis introduces the Dynamic Hierarchical Multi-Agent Framework (DHMAF). The limitations of existing frameworks, mainly the unreliability of unstructured single-agent systems and the static, pre-defined nature of structured multi-agent collaborations, necessitate a new architecture. The DHMAF is designed to synthesize the two competing goals of dynamic adaptability and structured reliability. This synthesis is achieved through three core principles derived from the analysis of the literature: Recursive Hierarchical Decomposition, Multi-Tier Validation, and Isolated Agent Memory.

Recursive Hierarchical Decomposition: Instead of relying on static, pre-defined agent roles, the DHMAF adopts a dynamic, recursive approach. This model is conceptually similar to classical HTNs [4]. A high-level, complex task is given to a primary agent, which recursively breaks it down into smaller, solvable sub-tasks. It then instantiates new “sub-agents” to handle these tasks, creating a hierarchy that is generated at runtime and is specific to the problem.

Multi-Tier Validation: To prevent the error propagation and unreliability common in unstructured systems, the DHMAF implements a dedicated, multi-tier validation layer. This is composed of specialized checker agents that operate at different levels of the hierarchy. This ensures that the output of each sub-agent is verified before being returned

to its parent agent, imposing structure and reliability on the dynamic decomposition process.

Isolated Agent Memory: To prevent the error propagation common in frameworks with shared context, each agent (Meta, Checker, etc.) operates with its own independent memory and state. Communication between agents is not a shared chat history; rather, information is passed as explicit, discrete messages (e.g., “new task,” “validation feedback,” “final result”). This ensures that a single agent’s errors or hallucinations are not automatically visible to the entire system.

3.2 Overall System Architecture

The DHMAF is implemented as a Python-based system built on several key technologies. The system’s backend is a FastAPI web service, which exposes endpoints to initiate and manage tasks.

The framework’s state and data management are handled by two components designed to enforce the Isolated Agent Memory Principle. A PostgreSQL database, interfaced via the SQLAlchemy ORM, is used for persistent storage of task data and results. A Redis cache is employed for managing transient agent state and memory. Critically, each agent’s memory is stored in a unique, isolated Redis key. A parent agent cannot access its child’s memory directly; it only receives the child’s final, validated output via the task queue. This architecture programmatically enforces the isolation, preventing cross-agent contamination.

To manage the execution of agents, the framework utilizes the Ray distributed computing library. When a parent Meta Agent generates multiple child agents for sub-tasks, Ray is used to parallelize their operations. This allows child agents to operate simultaneously in a distributed manner, significantly reducing computational and API bottlenecks and improving overall task completion time.

The core agentic logic is orchestrated using Google's 'google-generativeai' library for LLM calls (Gemini API) and incorporates components from the 'langchain' library to assist with functions such as prompt management and agent tooling. The architecture is built around three distinct categories of agents, which are instantiated at runtime to form the task hierarchy:

Prime Meta Agent: The single-entry point for any given user request.

Meta Agents: The sole execution agents, which are instantiated recursively.

Checker Agents: The validation agents that review the outputs of the Meta agents, performing roles defined as "Micro" and "Macro" checks.

The following sections will detail the specific roles of each agent and the workflow that governs their interaction.

3.3 The Agent Hierarchy and Workflow

The DHMAF operates by instantiating agents to create a dynamic task hierarchy. The process begins when a user submits a complex query, which is routed by the FastAPI

service to the Prime Meta Agent. This agent serves as the root node of the hierarchy. Its sole responsibility is to perform the initial task decomposition. It analyzes the user's goal and generates a list of high-level sub-tasks required to achieve it.

For each sub-task, the Prime Meta Agent instantiates a new Meta Agent to handle it. This is the "Recursive Hierarchical Decomposition" principle in practice. This newly created Meta Agent is an independent process that receives its instructions from its parent. It then makes a critical decision: it either tackles the problem directly if the task is granular enough, or it recursively performs its own decomposition, instantiating its own child Meta Agents to handle the new, smaller sub-tasks. This recursive process continues until all tasks are simple enough to be solved by a Meta Agent (the base case of the recursion).

Crucially, the "Multi-Tier Validation" principle is enforced at every step. When a Meta Agent completes its work (either by solving a task directly or by assembling the results from its children), its output is not immediately returned to its parent. Instead, it is passed through a sequential, multi-stage validation process.

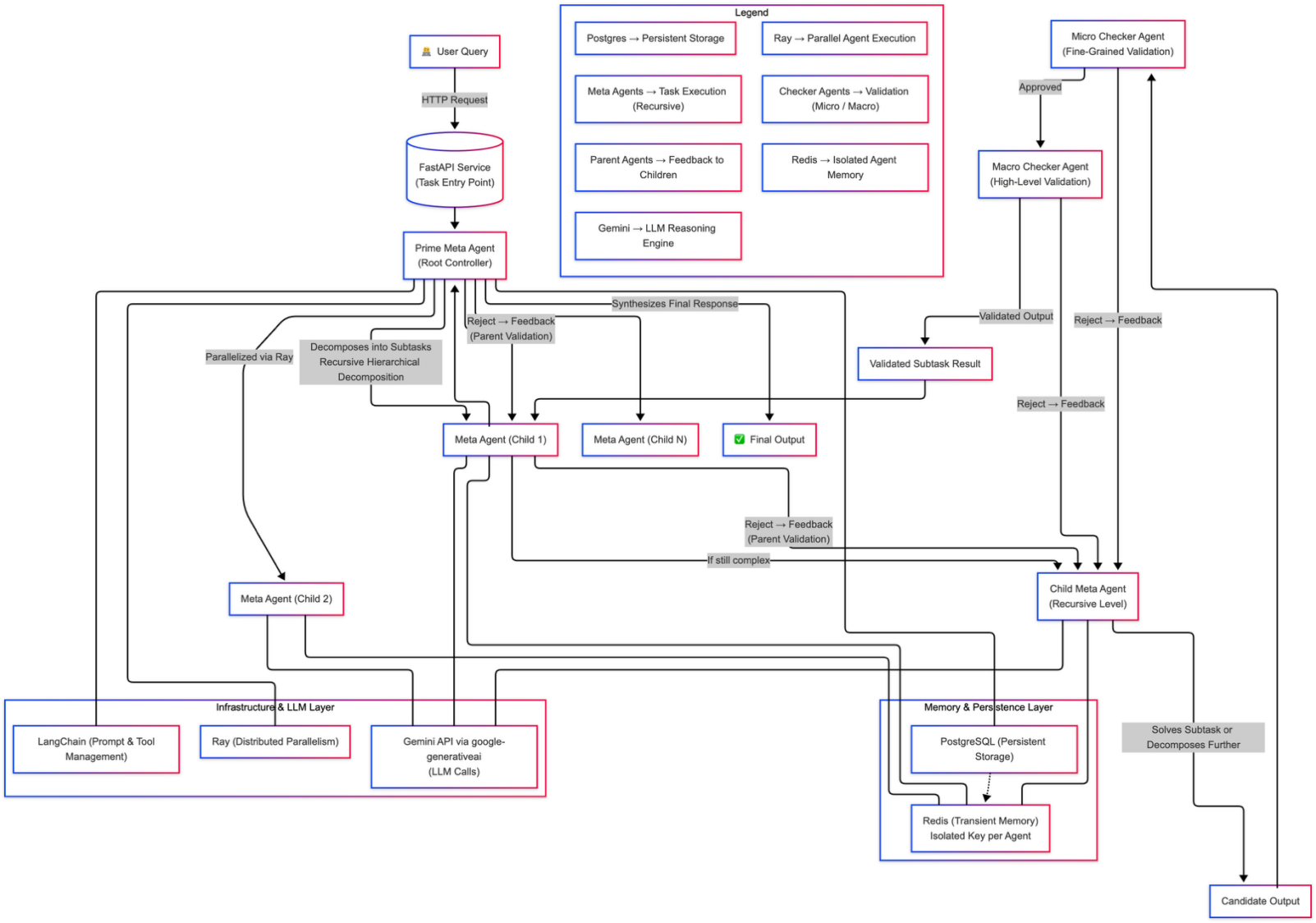
First, the output is sent to a Micro Checker Agent. This agent performs a fine-grained check and is given the original sub-task, the generated output, and has the authority to either approve the output or reject it and provide corrective feedback to its associated Meta Agent.

If the output passes the micro-check, it is then sent to a Macro Checker Agent. This second agent performs a high-level quality check, ensuring the response is coherent,

well-structured, and holistically answers the sub-tasks prompt. If either agent rejects the output, the feedback is sent back to the child Meta Agent, which must re-work the task based on the feedback.

Only after an output has been approved by both the Micro and Macro Checker Agents is the validated result passed up the hierarchy to the parent Meta Agent. This parent agent then performs its own final synthesis and validation. If the parent agent finds the child agent's output to be insufficient in the context of the parent's other sub-tasks, it can also reject the output and send it back to the child Meta Agent with corrective feedback. This multi-tier workflow ensures that errors are caught and corrected at the lowest possible level, preventing them from propagating and corrupting the final result.

Diagram 1 DHMAF Architecture



3.4 Core Agent Logic and Prompts

The DHMAF's principles are implemented via a set of specific prompts and decision-making logic. This section details this logic, ensuring the experiment is reproducible. All agentic calls in this experiment, including decomposition, execution, and validation, were made using the Gemini 1.5 Flash model, as defined in `config.py`.

3.4.1 Recursive Decision Logic

A core architectural challenge is determining when a Meta Agent should solve a task directly versus when it should recursively decompose it. This "granularity check" is handled by delegating the decision to an LLM call using a specialized "Decomposition Prompt" (from the `_decompose_task` function).

When a Meta Agent receives a task, it calls the Gemini 1.5 Flash model with the following prompt:

```
You are an expert programmer and planner. Your goal is to
break down the *current subtask*... into a *small number*
(max 3) of the **absolute next, concrete, actionable
steps**.
CRITICAL INSTRUCTIONS:
1. Focus ONLY on the immediate next steps. Do not outline
the entire plan.
2. Subtasks MUST be concrete actions. Use strong action
verbs.
    - GOOD: "Write Python function parse_log(line)."
    - BAD: "Analyze requirements." (Too abstract)
3. Generate a MAXIMUM of 3 subtasks.
4. If the current subtask itself is already a single,
concrete, actionable step, respond with an empty list:
{"subtasks": []}
5. Respond ONLY with a valid JSON object…

Context and Current Task:
JSON Response:
```

The Meta Agent then parses the LLM's JSON response. If a list of subtasks is returned, the agent recursively instantiates new child Meta Agents to handle each new subtask. If an empty list is returned (or if parsing fails) the agent determines the task is

"granular enough" and proceeds to solve it directly. This serves as the "base case" for the recursion.

3.4.2 Multi-Tier Validation Logic (The Checker agents)

The validation agents operate on specific, guided instructions via their system prompts, which are defined in config.py and called by the `_micro_checker` and `_macro_checker` functions. The Micro Checker Agent performs a fine-grained check with this prompt:

```
You are a micro-checker agent. Please check the correctness
and coherence of the following subtask result. If there are
any glaring mistakes, contradictions, or factual errors,
summarize them. Otherwise, say it's fine:
```

The Macro Checker Agent performs a high-level, holistic check on the combined result with this prompt:

```
You are a macro-checker agent. Please check the final
aggregated result for overall coherence, completeness, and
alignment with the top-level goal. If there are major
issues, summarize them. Otherwise, confirm it's acceptable:
```

3.4.3 Feedback and Memory Protocol

This section addresses how corrective feedback is passed while enforcing the "Isolate Agent Memory" principle. The framework implements two distinct, sequential corrective loops: The Micro-Check and the Macro-Check.

The `_micro_checker` is the first corrective loop. As seen in the `_recursive_agent_logic` and `_process_completed_ray` future functions,

the process is as follows. First, an agent completes a task and produces a `subtask_result`. Then, the `_micro_checker` is called on this result. If the feedback indicates an issue, the system generates a new `refinement_prompt`. This new prompt explicitly combines the original subtask, the previous attempt's result, and the refinement instruction and feedback from the checker. The agent is then called recursively with this new prompt. This enforces isolated memory passing feedback as new, explicitly task rather than modifying a shared state. This loop is repeated up to `max_micro_refinement_attempts` (set to 2).

Only after an output is approved by the Micro-Check is it passed to the `_macro_checker`, which forms the second corrective loop, consistent with the workflow described in Section 3.3. If the `_macro_checker` (which runs on the aggregated text) indicates an issue, it also generates corrective feedback. This feedback is then passed on to the child Meta Agent, which must re-work the task based on this high-level feedback, triggering the entire process again (up to `_max_macro_refinement_attempts`). This multi-tier workflow, with sequential Micro and Macro corrective loops, ensures that an output is validated for both fine-grained correctness and high-level coherence before ever being seen by the parent agent.

3.4.4 Experimental Constraints

To manage the high financial ($253) and computational costs of the benchmark two hard constraints were imposed on the framework's logic via the config.py file: a decomposition limit and a recursion limit. The decomposition prompt limited any agent to generating a maximum of three subtasks per Meta Agent. For the recursion limit, the framework's code enforced a global recursion limit of four. No task hierarchy was permitted to go deeper than four levels.

These constraints, discussed further in Section 5.3, were essential for making the experiment feasible but may have impacted the framework's performance.

Chapter 4. Results and Analysis

4.1 Experimental Setup

To validate the effectiveness of the DHMAF, a quantitative evaluation was performed against several established LLM benchmarks. The goal of this experiment was not to exhaustively test all possible domains, but to measure the framework's impact on performance in areas of complex reasoning and broad knowledge. The experiment is designed to measure the additive value of the DHMAF compared to its baseline model, not to provide a comparative benchmark against other agentic frameworks.

The evaluation used three standard benchmarks. The first, GSM8K (11-shot), is a test for grade-school mathematical reasoning, which requires multi-step logical and arithmetic operations to solve word problems. The second, MMLU (5-shot), is the Massively Multitask Language Understanding benchmark, a broad, generalist test covering 57 different subjects. The third, MMLU-Pro, is an advanced version of MMLU, featuring more complex questions that require expert-level reasoning and synthesis.

The DHMAF was implemented using Google's Gemini 1.5 Flash as the base LLM for all agent operations. The performance of the DHMAF is compared directly against the most recent citable, externally reported baseline scores for the "Gemini 1.5 Flash – Sep 2024" model. The primary metric for evaluation is the accuracy (percentage of correct answers) achieved on each test.

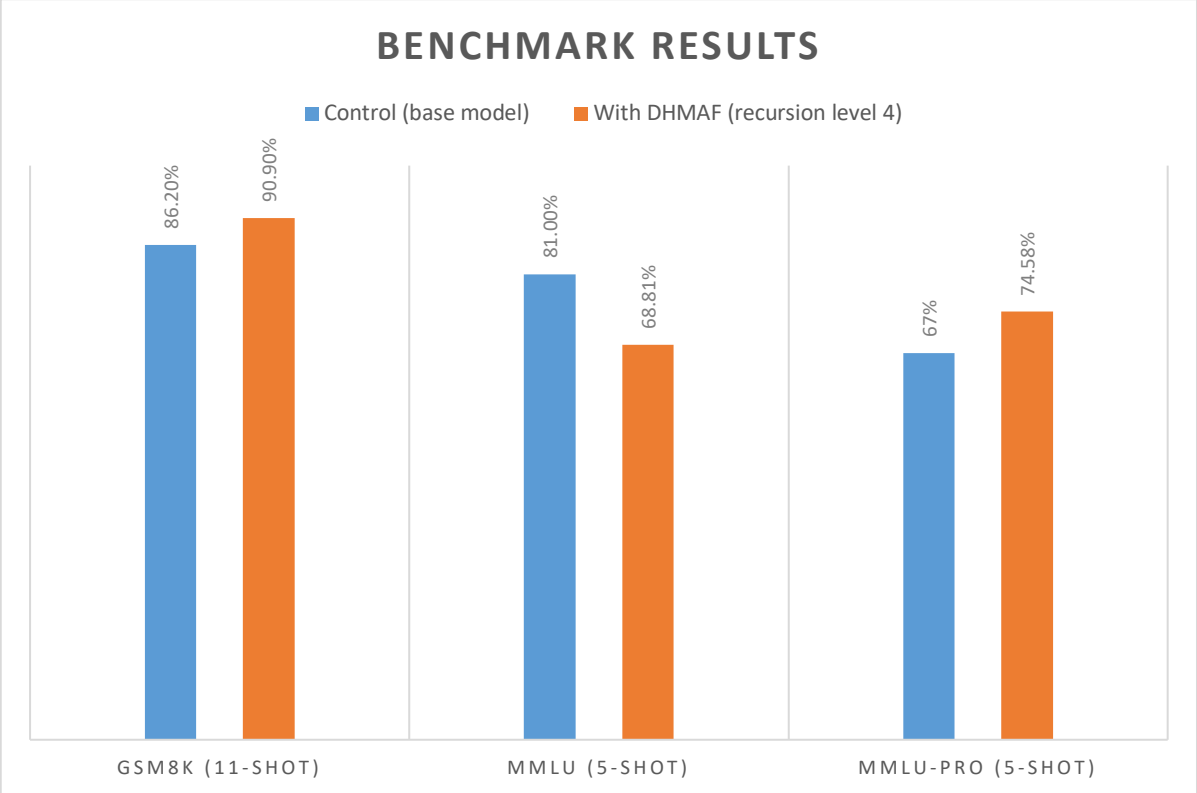
To ensure the validity of the comparison of the additive value of DHMAF to the baseline model, the k vales for the few-shot prompting strategies were strictly

standardized to match external baseline protocols. By controlling the number of shots, his experimental design isolates DHMAF as the primary independent variable, ensuring that observed performance deviations are attributable to DHMAF.

In addition to performance, the experiment also tracked the computational and financial costs of running the framework, as this is a key consideration for the practicality of complex agentic systems.

4.2 Quantitative Performance

The evaluation of the DHMAF yielded significant performance changes relative to the baseline model, demonstrating positive impact in multi-step reasoning tasks but also highlighting challenges in generalist applications. The results were derived from a



test run of 14,725 questions across three benchmarks, which consumed approximately 2.5 million API requests and incurred a total cost of $253.

Figure 1 Benchmark Results [9][10][11]

The following sections will provide a detailed analysis of these results.

4.3 Analysis of Specialized Reasoning Performance

The most significant finding from the experiment is the DHMAF's positive impact on specialized, complex reasoning tasks. The framework achieved a +4.7% increase on GSM8K and a +7.28% increase on MMLU-Pro.

This performance improvement strongly suggests that the core principles of the DHMAF are effective for these types of problems. Both benchmarks reward multi-step reasoning, logical consistency, and precision. It is hypothesized that the DHMAF's architecture directly addresses the failure of modes of a single LLM on such tasks:

- Recursive Decomposition allows the framework to break down a complex math problem (GSM8K) or an expert-level question (MMLU-Pro) into smaller, manageable logical steps. This prevents the LLM from attempting to solve the entire problem in one pass, which can lead to errors.
- Multi-Tier Validation is particularly critical. The Micro Checker agents can verify intermediate calculations or factual claims at each step. The Macro Checker agents can then ensure the assembled steps are logically coherent. This validation workflow, with its explicit feedback loops,

allows the system to catch and correct the logical leaps or arithmetic errors that a single LLM might otherwise make, leading to the observed increase in accuracy.

4.4 Analysis of Generalist Performance Anomaly

In sharp contrast to the specialized benchmarks, the DHMAF demonstrated a significant performance decrease of -12.19% on the standard MMLU benchmark. This result is an important finding, as it suggests the framework's architecture is not universally beneficial and may be detrimental under certain conditions.

The MMLU benchmark is a broad, generalist test covering 57 subjects, many of which rely on direct factual recall rather than complex, multi-step reasoning. The performance drop strongly supports the hypothesis that the DHMAF, in its current implementation, overcomplicates simpler tasks. This is a direct consequence of the methodological choices detailed in Section 3.4. The recursive decision logic (Section 3.4.1) relied on Gemini 1.5 Flash to determine if a task was "granular enough". For simple MMLU questions, the model should have returned an empty {"subtasks": []} list. The -12.19% drop suggests the model failed this check, initiating a costly and unnecessary decomposition. The complex validation workflow (Section 3.4.2) introduces new points of failure. The simple, nonexpert prompts for both the `_micro_checker` and `_macro_checker`, also run by Gemini 1.5 Flash, may have "hallucinated" errors in

correct, fact-based answers, triggering a “refinement loop” (Section 3.4.3) that was not only unnecessary but actively harmful.

For a simple, fact-based question, the baseline model’s single-pass answer may be sufficient. The DHMAF, by design, forces this simple question through a complex process of decomposition and multi-tier validation. This introduces multiple new points of failure. The overhead of the framework, including the interpretation of tasks by child agents and the judgments of the checker agents, may introduce errors or filter out correct answers that the baseline model would have otherwise produced. This suggests that the framework’s complex validation workflow, which is an asset for multi-step reasoning, becomes a liability for generalist, recall-based tasks.

4.5 Resource and Cost Analysis

A final, critical component of the evaluation is the analysis of the framework’s practicality. The test run, which covered 14,725 questions, consumed approximately 2.5 million API requests and incurred a total cost of $253.

This data highlights the significant trade-off inherent in the DHMAF. The performance gains on specialized tasks are achieved at a substantial computational and financial cost. The high volume of API calls is a direct result of the multi-agent, multi-step validation and feedback loops, where each step in the hierarchy may require multiple calls to the base LLM. This high cost underscores the practical limitations of such a

framework and suggests that its application would be best reserved for domains where accuracy over multi-step reasoning tasks is valued far more than cost, where risk of failure from a single-pass generation is unacceptably high.

Chapter 5. Conclusion

5.1 Summary of Contributions

This thesis introduced, implemented, and evaluated the Dynamic Hierarchical Multi-Agent Framework (DHMAF), an architecture that synthesizes Recursive Task Decomposition, Multi-Tier Validation, and Isolated Agent Memory. The system was empirically tested to measure its additive value and, most importantly, to quantify its performance-cost trade-offs.

The experiment results reveal a critical trade-off and support what this paper terms the “over-complication anomaly”. The framework’s primary finding was a significant performance decrease of -12.19% on the MMLU benchmark. This failure on simple, fact-based tasks demonstrates that the complex overhead of the DHMAF architecture can be actively detrimental.

This finding provides crucial context for the framework’s secondary success: on specialized complex reasoning tasks, the DHMAF did improve accuracy, increasing scores on GSM8K by +4.7% and on MMLU-Pro by +7.28%. These results validate the concept of hierarchical validation for multi-step reasoning tasks but highlight that it is not a universal solution for all tasks.

5.2 Discussion of Findings and Implications

The findings of this research are twofold. First, and most critically, the failure on MMLU is an important finding for the field. It suggests that complex agentic frameworks are not a universal solution. The "over-complication" hypothesis — that the framework's overhead, driven by flawed decomposition logic and simple validation prompts (Section 3.4), becomes a liability for simpler, fact-based tasks — is strongly supported by the data. This hypothesis is further supported by the methodology: the anomaly is likely the direct result of using a cost-effective model (Gemini 1.5 Flash) with the simple, non-expert validation prompts (Section 3.4.2), which were being used to trigger a computationally expensive, multi-stage corrective loop architecture (Section 3.4.3). This two-loop system (Micro and Macro), when run by a cost-effective model, added new points of failure instead of adding value. This highlights a critical trade-off: the modest gains made in accuracy on complex tasks come at the significant cost of performance on simple tasks like fact recall.

Second, the success on the GSM8K and MMLU-Pro provides strong evidence that this architecture can be a viable solution for enhancing LLM reliability in areas where multi-step reasoning must be improved if the trade-offs are acceptable.

Furthermore, the high cost of the experiment (2.5 million API calls at $253) positions such frameworks as stopgap solutions. They can improve the reasoning of a base LLM, but they do so at a high computational and financial price. This suggests that

while agentic frameworks are a powerful tool, fundamental improvements to the base LLMs themselves may prove to be a more efficient path to reliable, general-purpose AI.

5.3 Limitations of the Study

This study's conclusions are bound by several practical limitations. The most significant is the scope of the benchmarking.

First, the experiment was confined to measuring the additive value of the DHMAF against its baseline model. A direct comparative evaluation against other state-of-the-art frameworks like MetaGPT or CrewAI was not performed. This decision was necessitated by the significant financial and computational costs associated with such a large-scale evaluation. The test run for DHMAF alone, covering over 14,000 questions, required more than a week of continuous computation on a high-performance machine (Apple M3 Max 64 GB). This process, driven by the parallelization of thousands of concurrent API calls, resulted in extremely high and sustained CPU and RAM load (with 0% GPU utilization) on the test machine (Apple M3 Max 64 GB), in addition to the substantial, self-funded API costs. These practical constraints made a broader, multi-framework comparison infeasible for this research.

Second, the experiment was conducted using a single, cost-effective base LLM (Gemini 1.5 Flash). The performance trade-offs observed are therefore only validated for this specific model. It is unknown if the performance gains would persist on newer, more powerful models (such as Gemini 2.5 Pro, DeepSeek R1, or ChatGPT 5 Thinking),

whose improved internal reasoning might already solve the errors DHMAF was designed to fix. Conversely, the framework's "over-complication" of simple tasks might be an artifact of the base model used.

Third, the evaluation was limited to three specific benchmarks (GSM8K, MMLU, and MMLU-Pro) that test multi-step reasoning across mathematics and general knowledge over 57 different topics. The framework was not evaluated on other significant domains, such as code generation (e.g., HumanEval), creative writing, or complex instruction following. Therefore, the observed performance trade-offs may not be generalizable to all possible LLM tasks.

Fourth, the framework's performance is contingent on the specific system prompts used to guide the agents. This study did not include an exhaustive hyperparameter search for the optimal prompts for each agent. It is possible that the observed performance trade-offs could be further tuned with different prompt engineering, and the results reflect the performance of the single prompt architecture used in this implementation. The validation logic (Section 3.4.2) relied on simple, non-expert prompts. The -12.19% MMLU drop suggests these validation layers, run by a cost-effective model, was not robust enough to add value and may have introduced "false negatives" by incorrectly flagging correct answers, a problem that was amplified by the two-loop architecture (Section 3.4.3), which then forced the agent to re-work an already correct answer. The observed "over-complication anomaly" may be a direct result of this specific, under-tuned prompt architecture alongside the two-loop corrective architecture.

Fifth, the framework's core logic was itself artificially constrained to manage costs. As detailed in Section 3.4.4, the architecture was limited to a `RECURSION_LIMIT` of 4 and a `MAX_SUBTASKS_PER_LEVEL` of 3. These hard-coded limits were necessary to make the benchmark financially and computationally feasible, but they are not tuned parameters. It is unknown how performance would be affected if these constraints were altered or removed.

Finally, the experiment was conducted in a single run. Due to the stochastic nature of LLMs, the exact scores could likely vary slightly if the test were repeated. The reported results are therefore representative of a single, comprehensive experiment rather than an average across multiple trials.

5.4 Future Work

The results and limitations of this study suggest several key avenues for future research to build upon the DHMAF.

First, the most critical area for improvement is addressing the performance drop in the generalist MMLU benchmark. The current framework gives Meta Agents the choice to solve a task directly or delegate it, but the "over-complication" of MMLU tasks suggests this decision-making logic is not optimized. Future work should focus on refining this "adaptive complexity" logic. This could involve training a small "router model" or developing a more sophisticated heuristic for Meta Agents to better assess a task's true complexity, allowing the framework to avoid costly and error-prone decomposition on simple, fact-based queries.

Second, given the practical constraints of this study, a full comparative analysis against other leading agentic frameworks is a necessary next step. A future, well-resourced study should benchmark the DHMAF against frameworks like MetaGPT, CrewAI, and ADAS on a standardized set of tasks and a common base LLM. This would be the only way to determine where DHMAF truly stands in the broader landscape of agentic architectures.

Third, the framework should be evaluated on a wider array of benchmarks, like AIME Competition Math, GPQA, and SWE-bench, among many others. The framework should also be tested on agentic tasks that require tool use, such as web search. This would provide a more complete picture of its strengths and weaknesses.

Fourth, further research should explore the framework's performance when paired with newer, more powerful base LLMs. Re-running the experiment on Chain of Thought models like Gemini 2.5 Pro would answer a key question: as base models become more capable, do the interim frameworks like the DHMAF become obsolete, or do they scale, continuing to provide an additive benefit by catching the model's more subtle and complex errors?

Finally, the DHMAF's static, rule-based architecture is itself a limitation. A more advanced, brain-inspired framework could be developed based on the findings of the thesis. Such a system might use an "Orchestrator" or "Router" network to intelligently route tasks, applying complex decomposition only when necessary. Furthermore, it could move beyond a fixed set of agents and learn to evolve its own architecture, adding or

removing specialized agents and validation nodes via reinforcement learning. This would allow the system to not just optimize for accuracy, but also for cost and efficiency, for example, by learning to dynamically assign simpler tasks to less powerful, cheaper LLMs. This self-evolving, cost-aware architecture represents a potential next generation for hierarchical agentic frameworks.

Bibliography

[1] Hu, S., et al. (2024). *Automated Design of Agentic Systems (ADAS).* arXiv preprint arXiv:2408.08435. Retrieved from https://arxiv.org/pdf/2408.08435

[2] Chen, G., et al. (2024). *AutoAgents: A Framework for Automatic Agent Generation*. Retrieved from https://arxiv.org/pdf/2309.17288

[3] Derouiche, K., et al. (2025). *LLM Agents: A Survey*. Retrieved from https://arxiv.org/pdf/2508.10146

[4] Mu, X., et al. (2023). *Hierarchical Task Network Planning for Facilitating Cooperative Multi-Agent Reinforcement Learning*. Retrieved from https://arxiv.org/pdf/2306.08359

[5] Yang, J., et al. (2023). *Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond*. Retrieved from https://arxiv.org/pdf/2304.13712

[6] Novikov, A., et al. (2025). *AlphaEvolve: A coding agent for scientific and algorithmic discovery.* arXiv preprint arXiv:2506.13131. Retrieved from https://arxiv.org/pdf/2506.13131

[7] Hong, S., et al. (2024). *MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework*. arXiv preprint arXiv:2308.00352. Retrieved from https://arxiv.org/pdf/2308.00352

[8] Duan, Z., & Wang, J. (2024). *Exploration of LLM Multi-Agent Application Implementation Based on LangGraph+Crew AI*. arXiv preprint arXiv:2411.18241v1. Retrieved from https://arxiv.org/pdf/2411.18241

[9] LearnPrompting.org. (2024). Gemini 1.5 Flash. *Learn Prompting*. Retrieved from https://learnprompting.org/docs/models/gemini-1.5-flash

[10] LLM-Stats. (2024). Gemini 1.5 Flash. *LLM-Stats.com*. Retrieved from https://llm-stats.com/models/gemini-1.5-flash

[11] Google. (2024). Gemini 1.5 Flash-8B is now generally available for use. *The Google Developers Blog*. Retrieved from https://developers.googleblog.com/en/gemini-15-flash-8b-is-now-generally-available-for-use/

Appendix A. Figure and Diagram

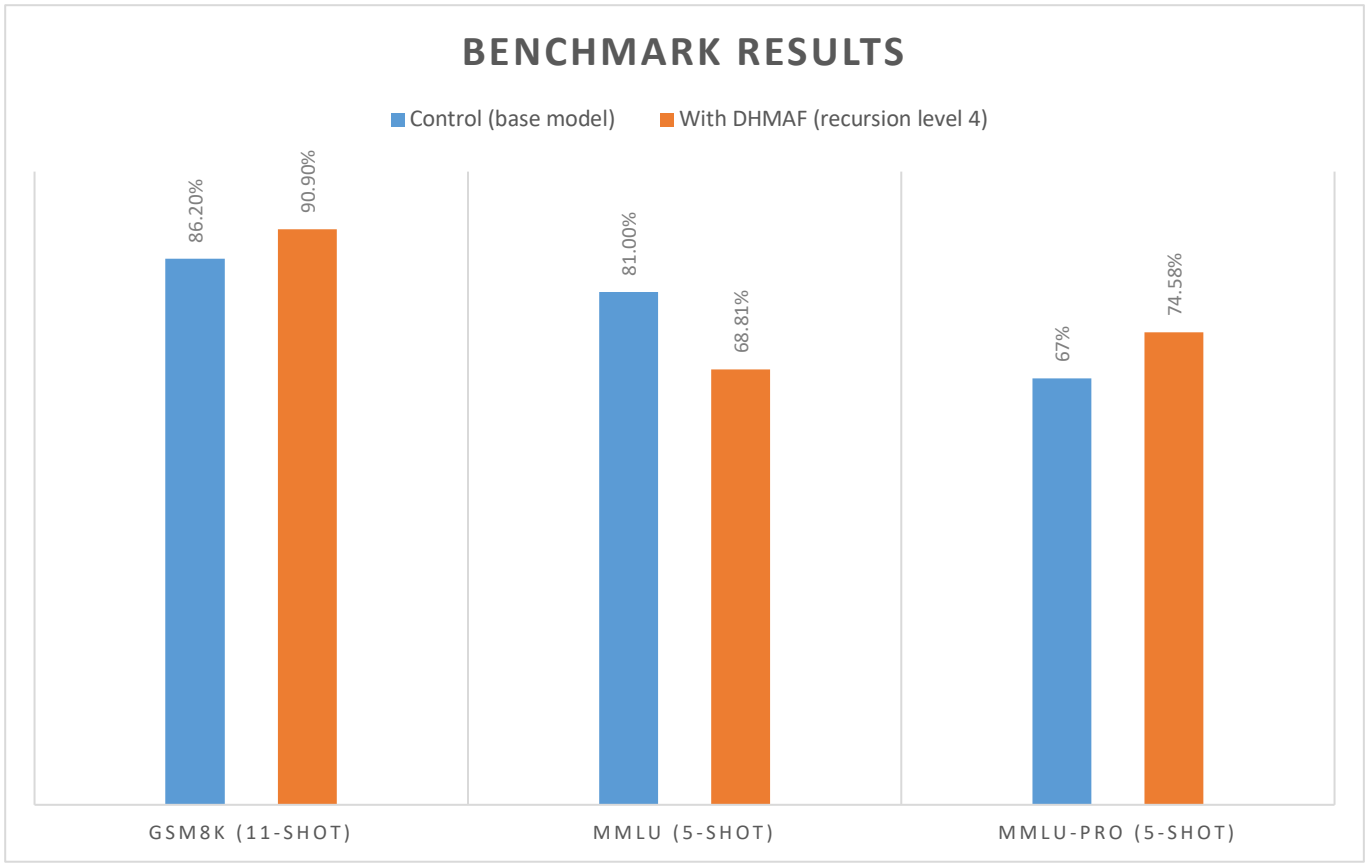


Figure 1 Benchmark Results

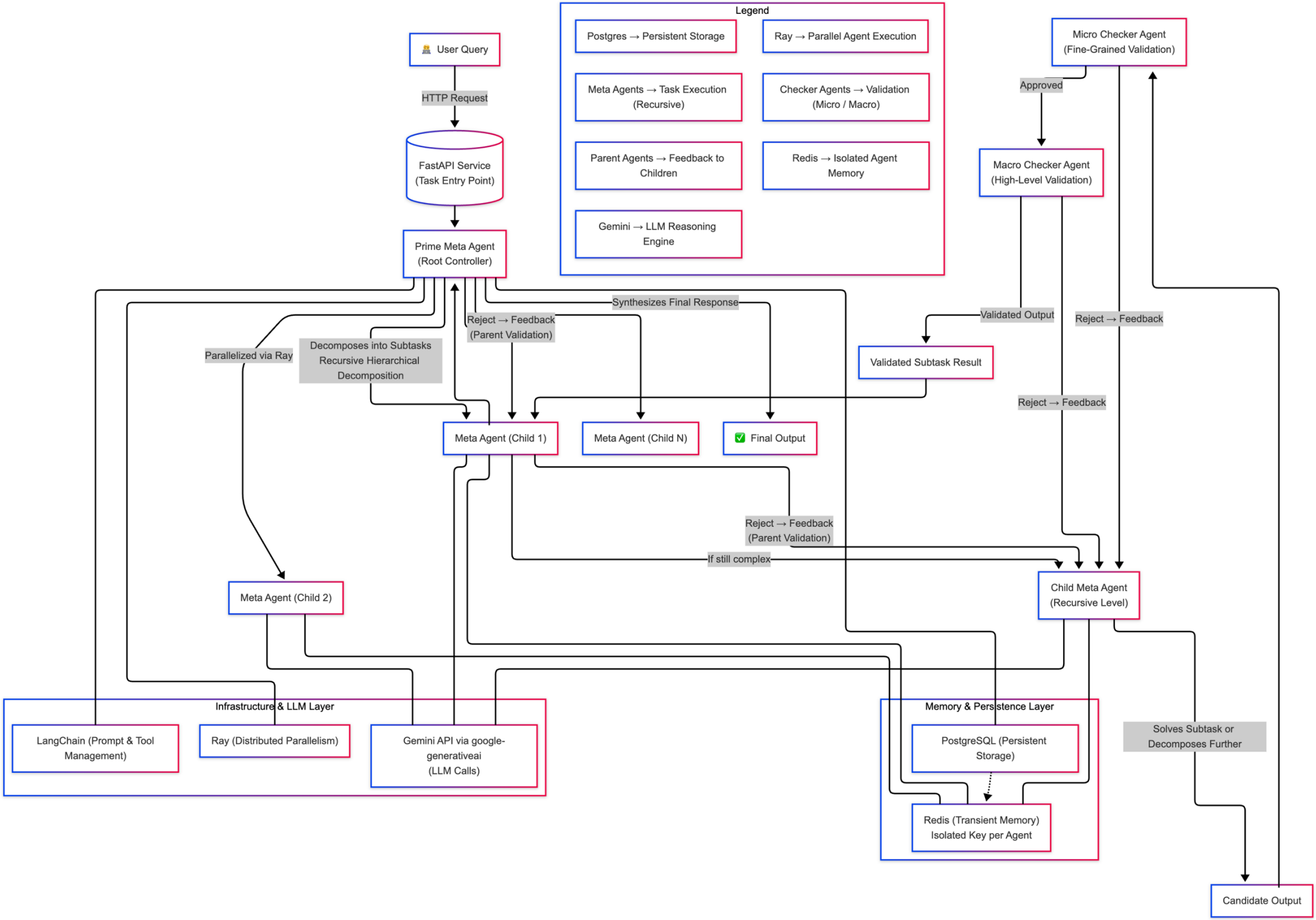


Diagram 1 DHMAF Architecture