Optimizing Agentic AI Applications Under Budget Constraints

Hamta Sedghani
hamta.sedghani@polimi.it
Politecnico di Milano
Milan, Italy

Federica Filippini
federica.filippini@unimib.it
University of Milano-Bicocca
Milan, Italy

Zahra Seyedi
zahrasadat.seyedi@mail.polimi.it
Politecnico di Milano
Milan, Italy

Blerina Spahiu
blerina.spahiu@unimib.it
University of Milano-Bicocca
Milan, Italy

Michele Ciavotta
michele.ciavotta@unimib.it
University of Milano-Bicocca
Milan, Italy

Danilo Ardagna
danilo.ardagna@polimi.it
Politecnico di Milano
Milan, Italy

Abstract

AI systems increasingly rely on multi-step agentic workflows that orchestrate heterogeneous computational agents, including Large and Small Language Models (LLMs and SLMs), each configurable at runtime with different inference methods and parameters. While this flexibility enables specialization and resource efficiency, it introduces complex trade-offs between accuracy, latency, and energy consumption, particularly under strict operational budgets. The sequential nature of these workflows further amplifies such challenges: errors propagate across steps, and resource decisions made early in the pipeline constrain downstream options. To address these challenges, this paper presents a formal optimization framework for configuring multi-step agentic pipelines under global time and energy constraints. Given a task decomposition and a heterogeneous pool of agent configurations, the framework selects exactly one agent–inference method–parameter triple per task to maximize end-to-end workflow quality. We formalize this as a Mixed-Integer Linear Programming problem and introduce two complementary objective formulations: a max-min accuracy objective that prioritizes robustness by improving the least accurate step, and a multiplicative accuracy objective that captures cumulative performance across the workflow. We further propose an iterative solution strategy that re-optimizes at each step, adapting to deviations between predicted and realized resource consumption. Experimental results demonstrate distinct accuracy-resource trade-offs between the two formulations and show that the approach scales effectively to workflows with up to a thousand steps.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Agentic AI, Large Language Models (LLMs), Resource-Constrained Optimization, Energy-Aware AI, Latency-Constrained Inference

ICPE Companion '26, Florence, Italy

ACM ISBN 979-8-4007-2326-1/2026/05
https://doi.org/10.1145/3777911.3800631

ACM Reference Format:
Hamta Sedghani, Federica Filippini, Zahra Seyedi, Blerina Spahiu, Michele Ciavotta, and Danilo Ardagna. 2026. Optimizing Agentic AI Applications Under Budget Constraints. In *Companion of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '26), May 04–08, 2026, Florence, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3777911.3800631

1 Introduction

Artificial intelligence (AI) has become a pervasive enabling technology across nearly all sectors, and among its emerging directions, agentic AI has rapidly gained prominence in both industrial and academic settings. Recent market analyses project exceptionally strong growth in this area, with estimates indicating that the agentic AI market will expand from USD 7.06 billion in 2025 to USD 93.20 billion by 2032, corresponding to a compound annual growth rate of approximately 44.6%[1]. Agentic AI systems are realized as coordinated assemblies of heterogeneous computational agents, typically comprising large language models (LLMs) and small language models (SLMs), that collaborate to solve complex, multi-step tasks [27]. Recent evidence suggests that such heterogeneity is increasingly reflected in *multi-model* deployments: while 40.9% of deployed agents still rely on exactly one model, the majority orchestrate two or more models (27.3% use two, 18.2% use three, and 13.6% use four or more) to balance quality, latency, and cost, or to support different modalities and operational constraints [21]. In contrast to traditional single-model agents [24], such systems must determine not only which agent is assigned to each task in a decomposed workflow, but also how that agent is configured, including the choice of inference strategy, parameterization, and tooling. Because these decisions can be updated at runtime, the system can exploit test-time compute by dynamically reallocating computational effort during execution [25]. This capability enables the selection of specialized or compact models that may outperform general-purpose LLMs on narrowly scoped tasks, while simultaneously reducing computational overhead and energy consumption [3]. Although this flexibility can yield substantial performance improvements, it also introduces a complex landscape of resource–accuracy trade-offs. Agents differ significantly in latency, energy requirements, and reliability, and the quality of the outcome depends critically on the robustness of the entire chain of intermediate computations.

[1] https://www.marketsandmarkets.com/Market-Reports/agentic-ai-market-208190735.html

These challenges are further amplified in practical deployments. Many real-world applications impose strict global constraints on execution time, energy consumption, or both, arising from latency requirements, hardware limitations, or operational cost budgets. As multi-step pipelines are inherently sequential, end-to-end latency must remain tightly bounded, and delays incurred at early stages directly propagate through the remainder of the pipeline. Moreover, errors introduced by the weakest-performing step can cascade downstream, while resource-intensive decisions made early in the execution restrict the feasible design space of subsequent stages. Consequently, the development of principled methods for navigating these interdependent trade-offs is essential to achieving robust, efficient, and scalable agentic AI systems.

This work develops a formal framework for optimizing multi-step agent AI applications under global resource budgets. We consider a heterogeneous pool of models, each with multiple inference methods and parameter spaces, and a prompt-specific sequence of tasks produced by an orchestrator. For each step, only a subset of agents is viable, and each agent–method–parameter choice induces stepwise accuracy, latency, and energy consumption. We formalize the Agent–Inference Method–Parameter Selection (AIMPS) problem as a Mixed-Integer Linear Programming (MILP) model, introducing two complementary objective functions that capture distinct notions of robustness in sequential reasoning: (1) **Max–min accuracy**, which maximizes the accuracy of the *lowest-accuracy step*. This objective reflects failure-sensitive pipelines, where overall task success is limited by the least reliable step. (2) **Multiplicative accuracy**, which maximizes the product of step accuracies. This alternative captures settings where end-to-end correctness degrades multiplicatively along the task chain: if each step accuracy is interpreted as a per-step success rate (equivalently, $1 - \text{error rate}$), then the end-to-end success of a multi-step pipeline that depends on correct intermediate outputs is well-approximated by a multiplicative combination of per-step success rates. The multiplicative formulation promotes balanced configurations and globally efficient resource usage, offering a complementary view of the accuracy–cost trade-off. Both formulations yield constrained optimization problems over a large combinatorial decision space. Together, they provide a unified lens for studying resource allocation, agent selection, and inference configuration in agentic AI systems.

The remainder of this manuscript is organized as follows. Section 2 reviews related work on agentic AI systems and resource-aware model selection. Section 3 introduces the system model, including the prompt structure, agent configurations, and accuracy metrics. Section 4 formalizes the AIMPS problem studied in this work, presenting both the max–min and multiplicative accuracy objectives under global resource constraints. Section 5 discusses solution strategies for these formulations. Section 6 reports experimental results evaluating the proposed framework, and Section 7 concludes the paper.

2 Related Work

Recent research on agentic AI has outlined a broad conceptual and technical landscape in which language models function as autonomous or semi-autonomous agents capable of interacting with external tools, services, environments, and other agents. Related work is organized into (i) agentic workflows and heterogeneous agent pools, (ii) cost-aware model selection, routing, and cascading, and (iii) resource allocation for sequential tasks under constraints, reflecting the three components our problem combines: agentic execution, cost-aware selection, and constrained sequential decision-making.

Agentic workflows and heterogeneous agent pools: This body of work spans multiple research threads, including agent architectures, tool use and orchestration, planning and reasoning, and multi-agent coordination. Collectively, these studies investigate how LLMs can move beyond passive text generation toward goal-directed behavior, persistent state, and adaptive decision-making. Sang et al. [23] describe a shift from pipeline-based systems, where planning and tool use are orchestrated externally, to model-native agentic AI, where capabilities such as planning and memory are increasingly internalized in the model via reinforcement learning. Complementary to this, Belcak et al. [3] argue that SLMs are often more suitable and economical for agentic workloads consisting of repetitive, specialized tasks, and advocate heterogeneous systems that combine LLMs and SLMs.

Complementing architectural perspectives, recent work has also highlighted the practical challenges of deploying and evaluating agentic systems in real settings. Pan et. al. [21] study agents in production and emphasize that real-world deployments expose failure modes and system-level bottlenecks, such as tool latency, reliability, and interaction overhead, that are often obscured in offline benchmarks, reinforcing the need for resource-aware orchestration and budgeting. More broadly, Tran et al. [27] survey collaboration mechanisms in LLM-based multi-agent systems, categorizing collaboration types, communication structures, and coordination protocols, which motivates heterogeneous, role-specialized agent pools and clarifies the orchestration design space within which our optimizer operates. While these works motivate multi-agent, tool-augmented workflows, they typically do not study how to optimize step-wise model and inference choices under global resource budgets for multi-step reasoning.

Cost-aware model selection, routing, and cascading: A growing literature studies cost-effective model selection for LLM serving, focusing on single-step decisions that map each query to a model, cascade, or ensemble under an accuracy-cost trade-off.

FrugalGPT [4] introduces a suite of cost-reduction mechanisms, including adaptive LLM cascades, function-specific routing templates, and response reuse, designed to reduce monetary inference cost while sustaining accuracy across diverse tasks. Dekoninck et al. [6] further formalize the routing problem by providing a unified theoretical framework for both routing and cascading, showing that each can be expressed as a solution to an expected accuracy–cost optimization problem. Their proposed cascade routing generalizes classical strategies and yields provably optimal policies under query-dependent cost constraints, clarifying the conditions under which escalating through a sequence of models is preferable to committing to a single model. Alongside these formulations, RouteLLM [20] learns query-level routing decisions from preference data to trade off quality and cost without invoking multi-call cascades. For each query, it estimates whether a larger model will outperform a smaller one and applies a threshold to select which model to invoke, avoiding multi-call cascades. Trained on Chatbot Arena preferences (with

optional augmentation), it reports large cost savings and shows generalization to unseen model pairs.

Beyond cascades, several systems learn routing or assignment policies over heterogeneous model pools. CPLS [17] approaches the assignment problem by learning compositional prediction structures that embed queries into a feature space, cluster them by similarity, and estimate per-cluster performance profiles across a heterogeneous model pool. This enables efficient group-wise scheduling of queries to LLMs under a global cost budget, exploiting both query correlations and cross-model behavior to reduce inference expenditure while maintaining accuracy. OptLLM [18] formulates LLM routing as a bi-criteria optimization problem balancing predicted accuracy and inference cost. The system employs lightweight accuracy predictors and Pareto-front exploration, using destruction–reconstruction heuristics to iteratively refine candidate assignments and uncover high-quality accuracy–cost trade-offs; empirical evaluations show substantial cost reductions without degrading accuracy relative to strong baselines. Finally, ThriftLLM [11] studies cost-aware ensemble selection, framing the problem as selecting an optimal subset of models that maximizes classification accuracy under a strict cost budget. The authors provide approximation guarantees for their combinatorial selection algorithm and demonstrate that carefully chosen ensembles can outperform individual large models at a fraction of the inference cost. Collectively, these approaches highlight the benefit of heterogeneity for cost efficiency, but they largely treat each query as an isolated decision and optimize primarily monetary or expected-cost budgets, rather than global time-energy budgets over sequential reasoning pipelines.

Resource allocation for sequential tasks under constraints: Closer to our setting, [1] studies allocating a task composed of sequential subtasks to a pool of heterogeneous agents while respecting time constraints. They formalize subtask allocation as an optimization problem parameterized by subtask difficulty, workload, and reward, with agents characterized by costs and capabilities, and they compare LLM-based orchestration with classical optimization methods in static and dynamic environments. However, this line of work focuses on assigning subtasks to agents given a task decomposition, and does not consider step-wise choices over inference strategies and parameter configurations, nor joint optimization under multi-dimensional (time and energy) global budgets.

BAMAS [29] addresses how to construct multi-agent LLM systems under explicit budget constraints by jointly optimizing which models to include and how they collaborate. It first selects a budget-feasible model set via an Integer Linear Programming (ILP) formulation that balances a performance proxy (tiered ranks) against estimated token cost, and then uses an RL-based topology selector to choose an interaction structure (e.g., linear, star, feedback, planner-driven) that maximizes a composite success–efficiency reward. Empirically, BAMAS achieves comparable accuracy to strong agent-construction baselines while reducing cost by up to 86%, and adapts topology complexity as budgets tighten or loosen.

CoRL [12] proposes a centralized multi-LLM system where a controller LLM decides whether to answer directly or selectively call specialized expert models, avoiding the high cost of decentralized "call-everyone" frameworks. The coordination policy is trained with reinforcement learning using dual objectives that reward task performance while penalizing exceeding a cost budget,

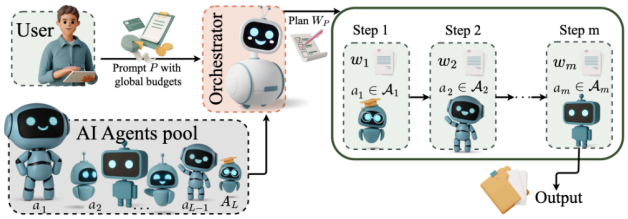

Figure 1: Agentic system overview: the orchestrator maps prompt P to a sequential pipeline and assigns an agent to each step.

and it is trained to be controllable across multiple budget modes (e.g., low/medium/high) by conditioning on budget prompts and budget-specific rewards. Experiments show a single trained coordinator can operate effectively across budget regimes, remaining economical in low-budget settings while leveraging experts to reach or exceed top-expert performance in higher-budget modes.

In contrast to prior work, which largely optimizes single-step routing or ensemble selection under monetary or expected-cost objectives, or allocates pre-specified subtasks to agents under a single resource constraint, we study prompt-defined, sequential multi-step pipelines where decisions are coupled across steps. Our method selects not only the agent/model at each step but also the inference strategy and parameter configuration, explicitly modeling their joint impact on accuracy, latency, and energy. We further impose global time–energy budgets over the entire pipeline and optimize end-to-end performance using two complementary accuracy-aware objectives, yielding a unified formulation for budgeted agentic computation that is not addressed by existing routing, agent-construction, or multi-agent allocation approaches.

3 System Model

The AIMPS problem analyzed in this work concerns the step-wise selection, from a finite pool of AI agents, of an LLM or SLM and its inference-time configuration to maximize the quality of the response to a given prompt P, subject to global constraints on execution time and energy consumption. We model the system as a sequential multi-agent computation pipeline. A central orchestrator receives the prompt P from a user and decomposes it into a sequence of interdependent tasks (see, e.g., [21]), referred to as an *execution plan* $\mathcal{W}_P$. For each task w of the plan, the orchestrator assigns a feasible *aimp* configuration, namely a triple consisting of an agent a, an inference method $s \in \mathcal{S}_a$, and a parameter configuration $r \in \mathcal{R}_s$. The agent pool is heterogeneous: LLMs offer higher capability, while SLMs provide lighter-weight alternatives for specialized tasks. Moreover, each agent exposes multiple inference-time methods and configuration spaces, enabling fine-grained control of the quality–resource trade-off. The core challenge is to orchestrate these choices across multiple dependent steps while respecting global latency and energy budgets, as overall task quality depends on both the pipeline structure and the allocation of resources across its stages [21, 29].

As illustrated in Figure 1, the main entities of the system are:

- **User:** Provides a prompt P, specifying the overall objective to be achieved together with implicit or explicit expectations

on response quality and resource usage, expressed as global latency and energy budgets.

- **Orchestrator:** Receives the prompt P, constructs an execution plan $\mathcal{W}_P$ by decomposing the computation into an ordered sequence of tasks w, and selects a feasible *aimp* configuration for each task. The orchestrator enforces global budgets and ensures that intermediate outputs are passed from one agent to the following one (since there is still not consensus in the literature, see e.g., [21], in the following we will use the terms tasks and steps interchangeably).
- **AI Agents:** A heterogeneous set of AI agents, including LLMs and SLMs. Each agent supports multiple inference methods and parameter configurations, and executes the tasks of $\mathcal{W}_P$ as assigned by the orchestrator.

In the following, we describe the structure of a prompt and the structure of AI agents used in the system.

3.1 Prompt Structure

A prompt is represented as the tuple $P = (p, d_P, E_P, T_P)$, where:

- p: The input text provided by the user, specifying the overall objective to be achieved.
- $d_P \in \mathbb{R}_{\geq 0}$: A scalar score estimating the difficulty or complexity of the prompt, computed by the Orchestrator via a meta-model or heuristic (e.g., predicting whether a higher-capability model is needed), following learned routing approaches such as RouteLLM [20]. This score allows the orchestrator to adjust the execution plan and the set of candidate agents.
- $E_P \in \mathbb{R}_{\geq 0}$: The global energy budget (in watt-hours, Wh) available to execute the prompt. Since different agents and inference-time configurations entail different energy costs, the orchestrator must ensure that the selected strategy respects this limit.
- $T_P \in \mathbb{R}_{\geq 0}$: The global time budget (in seconds) available to complete the prompt, ensuring that the response meets latency constraints.

The orchestrator decomposes the prompt P into an execution plan $\mathcal{W}_P$, modeled as a sequential pipeline: each step must complete before the following one starts. This structure captures realistic multi-step workflows, where each step depends on intermediate outputs produced by previous ones. For each task $w \in \mathcal{W}_P$, we denote by $\mathcal{A}_w \subseteq \mathcal{A}$ the subset of candidate agents that can execute it. In practice, not all agents in the pool $\mathcal{A}$ are suitable for every task; therefore, $\mathcal{A}_w$ is often a strict subset of $\mathcal{A}$, reflecting heterogeneous capabilities and specialization. In this setting, agentic systems can explicitly leverage specialized agents (e.g., AgentGroupChat [9]): some models excel at mathematical reasoning (e.g., Minerva [14] or Llemma [2]), while others are better suited for natural language understanding (e.g., BERT [7] or DeBERTa [10]).

3.2 Agents Structure and Performance Metrics

The system has access to a finite set of agents per task w ($\mathcal{A}_w$), and each agent is not simply a monolithic model but rather a configurable computational unit. In particular, each agent $a \in \mathcal{A}_w$ is characterized by:

- $\mathcal{S}_a$: the set of inference-time methods available for agent a (e.g., best-of-N [13], beam search [28], and lookahead [30]).
- $\mathcal{R}_s$: the set of admissible parameter configurations for method $s \in \mathcal{S}_a$. For example, if s = best-of-N, then $\mathcal{R}_s = \{1, 2, 3, \ldots, R_s\}$ may represent the number of samples.

This nested representation, in which each agent supports multiple inference-time methods and configurations, enables fine-grained control of the accuracy–resource trade-off. More sophisticated methods and settings can improve task-specific performance, but typically incur higher computational and energy costs, thus requiring careful balancing under global resource constraints.[2]

Performance metrics: The performance of any agent-inference method-parameter (*aimp*) configuration is characterized by four fundamental metrics, all of which are task-dependent. In particular, for a specific agent a executing task w of prompt P with inference method s parameterized by r, we consider:

- $\hat{a}_{w,a,s,r} \in (0, 1]$: Expected accuracy of agent a on the task w, when running inference method s with parameter r.
- $\hat{t}_{w,a,s,r} \in \mathbb{R}_{\geq 0}$: Expected execution time (in seconds) for agent a on w, when running inference method s with parameter r.
- $\hat{e}_{w,a,s,r} \in \mathbb{R}_{\geq 0}$: Expected energy consumption (in Wh) for agent a on w, when running inference method s with parameter r.

We assume the orchestrator has access to time and energy consumption estimates for each *aimp* configuration, obtained from a profiling/calibration phase on representative tasks. For a new w, the orchestrator retrieves estimates from the most similar profiled tasks (e.g., via embedding-based clustering), following common practice in learned routing and assignment methods that build per-query/per-cluster performance models (CPLS [17], OptLLM [18], RouteLLM [20]). Runtime and energy are measured using standard inference benchmarking and power-instrumentation methodology (e.g., MLPerf Inference [22]), and updated online using obtained measurements.

The accuracy estimate $\hat{a}_{w,a,s,r}$ quantifies the quality of executing step w with the selected *aimp* configuration. We interpret $\hat{a}_w$ as an estimated step success rate, namely the probability that task w produces an intermediate output that is correct, or at least usable by downstream steps in the sequential plan.[3] In practice, $\hat{a}_w$ can be estimated during a lightweight calibration phase on representative instances of each task type. When ground-truth labels or automatic checks are available (e.g., exact match, constraint satisfaction, unit tests), $\hat{a}_w$ can be estimated empirically. Otherwise, it can be approximated using a learned verifier or an LLM-based judge applied to intermediate (step-level) outputs, which assigns a correctness score (and, when calibrated, a proxy probability), as in verifier-based selection for reasoning tasks [5] and LLM-as-a-judge evaluation frameworks [16, 31].

To evaluate the end-to-end quality of the sequential pipeline, we adopt two complementary criteria based on step-level success estimates: (i) a multiplicative aggregation $\prod_w \hat{a}_w$, which captures

[2]This abstraction over inference-time methods and configuration spaces allows our system model to cover both standalone LLM/SLM-based solutions and tool-augmented agents.

[3]Note that the downstream impact of a step depends on error propagation and on the robustness of subsequent steps, hence end-to-end quality can only be approximated by simple aggregations of step-level success estimates.

error compounding along a sequential chain and provides a conservative proxy of the probability that all steps succeed, and (ii) a worst-step criterion $\min_w \hat{a}_w$, which emphasizes robustness by preventing a single low-accuracy step from dominating overall performance. Both product and minimum reductions over step-level success estimates are common in step-based evaluation settings (e.g., process-supervised scoring [15]).

4 Problem Formulation

Given the execution plan $\mathcal{W}_P$ and the pool of AI agents and associated configurations introduced in the previous section, solving the AIMPS problem means selecting exactly one *aimp* configuration for each task $w \in \mathcal{W}_P$, maximizing the overall response accuracy while meeting the global resource budgets (E_P, T_P). This section introduces the Mixed-Integer Linear Programming (MILP) formulations we propose for the AIMPS problem, which consider two alternative objective functions to capture different notions of solution quality in multi-step agentic pipelines.

We first formalize the decision space and objective functions.

4.1 Decision Vector and Assignment Constraints

The decision space Π denotes the set of all feasible *aimp* assignments for the execution plan $\mathcal{W}_P$. Each element of Π specifies, for every task $w \in \mathcal{W}_P$, a triple consisting of an agent, an inference method, and a parameter configuration. Formally,

$$\Pi = \left\{ \{(a,s,r)\}_{w \in \mathcal{W}_P} \;\middle|\; \begin{array}{l} a \in \mathcal{A}_w, \\ s \in \mathcal{S}_a, \\ r \in \mathcal{R}_s \end{array} \right\},$$

We encode a specific assignment using a binary decision tensor $\mathbf{x} = (x_{w,a,s,r})$, where $x_{w,a,s,r} \in \{0,1\}$ and $x_{w,a,s,r} = 1$ if task w is assigned to agent a with method s and configuration r, and 0 otherwise. To ensure that exactly one *aimp* configuration is selected for each step, the assignment must satisfy

$$\sum_{a \in \mathcal{A}_w} \sum_{s \in \mathcal{S}_a} \sum_{r \in \mathcal{R}_s} x_{w,a,s,r} = 1, \qquad \forall w \in \mathcal{W}_P. \tag{1}$$

Additionally, the following constraints must hold:

$$\sum_{w \in \mathcal{W}_P} \sum_{a \in \mathcal{A}_w} \sum_{s \in \mathcal{S}_a} \sum_{r \in \mathcal{R}_s} x_{w,a,s,r}\, \hat{t}_{w,a,s,r} \le T_P \tag{2}$$

$$\sum_{w \in \mathcal{W}_P} \sum_{a \in \mathcal{A}_w} \sum_{s \in \mathcal{S}_a} \sum_{r \in \mathcal{R}_s} x_{w,a,s,r}\, \hat{e}_{w,a,s,r} \le E_P \tag{3}$$

$$x_{w,a,s,r} \in \{0,1\} \quad \begin{array}{ll} w \in \mathcal{W}_P, & a \in \mathcal{A}_w, \\ s \in \mathcal{S}_a, & r \in \mathcal{R}_s. \end{array} \tag{4}$$

Equation (2) ensures the total time consumption across all sequential steps does not exceed T_P. Equation (3) constrains the total energy consumption across all steps to be no more than E_P.

4.2 Objective Functions

We formulate the AIMPS optimization objectives using the accuracy metric introduced in Section 3.2. As mentioned, the overall accuracy of a multi-step pipeline can be evaluated either through the performance of its weakest step or through the joint contribution of all steps. These two perspectives naturally lead to two complementary objective functions, which we present in the following sections.

4.2.1 Model 1: Max-Min Accuracy Objective. We first consider a robust objective that explicitly accounts for the sequential structure of the execution plan $\mathcal{W}_P$. As the reasoning steps are executed in sequence and each step depends on the output of the previous one, an error at any stage can propagate and significantly degrade the quality of the final response. As a result, the quality of the final output depends heavily on the weakest-performing step. To capture this effect, we adopt a max-min accuracy objective, which seeks to maximize the minimum expected accuracy across all steps:

$$\max_{\mathbf{x},z} \quad z \tag{5}$$

subject to Equations 1–4, and,

$$\sum_{a \in \mathcal{A}_w} \sum_{s \in \mathcal{S}_a} \sum_{r \in \mathcal{R}_s} x_{w,a,s,r}\, \hat{a}_{w,a,s,r} \;\ge\; z, \quad \forall w \in \mathcal{W}_P \tag{6}$$

$$z \in (0,1] \tag{7}$$

Equation (6) enforces that the accuracy achieved at each task w is at least z, thereby implementing the max-min objective.

4.2.2 Model 2: Multiplicative Exponent Form. While the max-min formulation emphasizes robustness, it may be overly conservative in scenarios where overall task success depends on the joint performance of all steps rather than on the worst-case step alone. To capture this alternative notion of quality, we consider a second objective that maximizes the product of step-wise accuracies. This formulation rewards configurations that achieve consistently high accuracy across the entire pipeline and naturally penalizes poor performance at any individual step:

$$\max_{\mathbf{x}} \prod_{w \in \mathcal{W}_P} \left(\prod_{a \in \mathcal{A}_w} \prod_{s \in \mathcal{S}_a} \prod_{r \in \mathcal{R}_s} \left(\hat{a}_{w,a,s,r}\right)^{x_{w,a,s,r}} \right) \tag{8}$$

To preserve the linearity of the formulation, we apply a logarithm to the previous equation, which yields:

$$\max_{\mathbf{x}} \sum_{w,a,s,r} x_{w,a,s,r} \log\left(\hat{a}_{w,a,s,r}\right) \tag{9}$$

subject to Equations (1)–(4).

In particular, Equation 9 with Constraints (1)–(4) can be interpreted as a special case of a *multiple-choice, multi-dimensional knapsack problem (MMKP)* [19] with mandatory selection from each class, a problem known to be NP-hard. In this interpretation, each task $w \in \mathcal{W}_P$ defines a choice class from which exactly one item must be selected, where the item identifies a candidate *aimp* configuration (a, s, r) (see Equation 1). The profit associated with selecting this configuration for task w is given by $\log\left(\hat{a}_{w,a,s,r}\right)$, while the resources consumption defines multiple knapsack dimensions, namely execution time and energy, with weights $\hat{t}_{w,a,s,r}$ and $\hat{e}_{w,a,s,r}$. The global budgets T_P and E_P serve as the capacity constraints for the corresponding dimensions, imposing through Equations (2) and (3) the multi-dimensional capacity limits.

5 Solution

To solve the optimization problems introduced in Section 4, we employ an iterative procedure that repeatedly constructs and solves the optimization model on a step-by-step basis. Rather than solving the entire sequence of tasks with fixed resource budgets in a single shot, the orchestrator re-optimizes at every step of the execution plan. This design reflects the practical behavior of agentic systems: measured latency and energy consumption often deviate from estimates, and updating the remaining budgets after each step produces allocations that are more robust to uncertainty and drift. At the beginning of each iteration, the orchestrator builds a new optimization model using the remaining steps, the current residual budgets, and the accuracy, latency, and energy estimates associated with each *aimp* configuration. Solving this restricted model yields an assignment for all remaining steps, but only the configuration for the current step is executed, while the rest serves only as a provisional prediction that is recomputed in the following iteration.

In some iterations, the optimization problem may become infeasible because the remaining time or energy budget is insufficient to accommodate the predicted cost of completing the remaining tasks. To prevent premature termination, we apply a simple feasibility-recovery mechanism: whenever the solver reports infeasibility, both budgets are relaxed by a fixed multiplicative factor (10% in our implementation), after which the model is rebuilt and re-solved. This iterative relaxation continues until feasibility is restored. The number of relaxations required offers insight into how tightly the original constraints bounded the feasible set and quantifies the degree of mismatch between estimated and realized resource consumption.

Once a feasible solution is obtained, the orchestrator executes the selected configuration for the current task w. The system then measures the realized latency and energy consumption. As mentioned, their values may deviate from offline estimates due to system noise, scheduling delays, and model variability, while accuracy is assumed to remain at its nominal value from the offline estimation, as it is difficult to measure reliably at runtime. The remaining budgets are updated and passed to the next iteration; the process continues until all steps in the plan have been resolved.

Algorithm 1 summarizes the complete procedure. Lines 1–5 initialize the measured latency and energy (i.e., the ones obtained from the monitoring system) over the executed steps as empty set T^{meas}, E^{meas}, iteratively solve the optimization model for the current set of unsolved tasks and retrieve the first task from the unsolved task list (w). Lines 6–8 store the estimated results in the first iteration as $\hat{T}, \hat{E}, \hat{A}$, while Lines 10–14 measure the latency and energy consumption (t_w, e_w), update the budgets accordingly and remove the current task from unsolved tasks. The algorithm then proceeds to the next task. The algorithm terminates when no unsolved tasks remain and returns both the estimated and measured performance metrics.

Algorithm 1 Sequential Task Resolution

1: Initialize remaining budgets T, E and unsolved tasks $\mathcal{W}_P$
2: Initialize measured metrics $T^{meas} \leftarrow \emptyset$, $E^{meas} \leftarrow \emptyset$
3: **while** $\mathcal{W}_P \neq \emptyset$ **do**
4: Solve model for the remaining tasks $\mathcal{W}_P$ under budgets (T, E)
5: $w \leftarrow$ Pick next task from $\mathcal{W}_P$
6: **if** w is the initial task in $\mathcal{W}_P$ **then**
7: $\hat{T}, \hat{E}, \hat{A} \leftarrow$ Predicted performance for all tasks
8: **end if**
9: Execute task w
10: t_w and $e_w \leftarrow$ Measure the latency and energy consumption
11: $T \leftarrow T - t_w, \quad E \leftarrow E - e_w$ ▷ Update budgets
12: Update T^{meas}, E^{meas} with t_w, e_w, respectively
13: $\mathcal{W}_P \leftarrow \mathcal{W}_P \setminus \{w\}$
14: **end while**
15: **return** $\hat{T}, \hat{E}, \hat{A}, T^{meas}, E^{meas}$

6 Experimental Analysis

In this section, we empirically evaluate the proposed optimization framework for sequential agentic AI pipelines under global time and energy constraints. The experiments are designed to assess three main aspects: (i) how effectively the framework respects global resource budgets during execution, (ii) how the two proposed objective formulations, namely max-min accuracy and multiplicative accuracy, differ in terms of accuracy-resource trade-offs, and (iii) how the solution scales with respect to the number of steps in the execution plan and the size of the configuration space.

All experiments run on a Linux Ubuntu server equipped with a 2.30 GHz Intel CPU (20 cores) and 10 GB of memory. The optimization problems were solved using Gurobi 13.0.0 with multithreading enabled (10 threads). Section 6.1 presents the experimental setup, while Section 6.2 provides the experimental results obtained under these settings.

6.1 Experimental Setup

We consider a synthetic yet representative experimental setting that reflects heterogeneous agent pools and configurable inference strategies, while allowing controlled exploration of scalability. The system parameters used in the experiments are summarized in Table 1. For the small-scale analysis, we consider workflows with $m = 10$ sequential tasks. Each task w admits a single compatible agent ($|\mathcal{A}_w| = 1$), but each agent supports multiple inference methods and parameter configurations. Specifically, each agent provides $|\mathcal{S}_a| = 5$ inference methods, each with $|\mathcal{R}_s| = 5$ possible parameter settings. Execution time, energy consumption, and accuracy values are sampled from uniform distributions over bounded ranges, as summarized in Table 1. Execution-time ranges are obtained from a prior profiling study on representative agent configurations conducted as part of an earlier project [26], while the selected energy-consumption ranges are consistent with empirical measurements reported for large-scale AI inference workloads [8]. Accuracy values are synthetic and serve as nominal indicators to evaluate relative trade-offs between configurations, rather than to model absolute prediction quality. These ranges are chosen to reflect typical variability observed across inference configurations in practice, where more aggressive decoding strategies yield higher accuracy at increased computational cost.

To assess scalability, we then increase the number of tasks to $m \in \{100, 500, 1000\}$, the number of compatible agents up to 10 and enlarge the configuration space by considering up to 10 inference methods and 10 parameter values. Under the largest-scale setting, the resulting MILP contains on the order of 10^6 binary decision

variables x (see Section 4.1). For each setting, both optimization formulations are evaluated under identical conditions. All reported results are averaged over 10 random instances unless otherwise stated.

For the small-scale experiments, we evaluate the system using the full sequential execution procedure described in Algorithm 1, in which the optimization problem is re-solved at every step of the pipeline. Because measured values often deviate from offline estimates due to system noise, scheduling delays, and model variability, after each task w, the time and energy budgets are updated by subtracting the measured latency and energy consumption (t_w, e_w). These measured values are obtained by applying a ±10% random perturbation to the corresponding offline estimates $\hat{t}_{w,a,s,r}$ and $\hat{e}_{w,a,s,r}$ associated with the selected configuration, i.e., when $x_{w,a,s,r} = 1$ (see Section 4.1), thereby capturing stochastic variability between estimated and measured resource costs.

In contrast, for the scalability analysis we focus exclusively on the computational complexity of the underlying optimization problems. To isolate solver performance, the optimization model is solved once per instance using the full set of steps and fixed global budgets, without sequential re-optimization or conservative budget updates. This allows us to assess how solution time scales with the number of steps and configuration options, independently of execution-time variability. Moreover, solving the optimization problem for all tasks simultaneously is the most computationally demanding case, as it involves the largest number of decision variables and feasible configurations, and therefore the most informative to evaluate scalability.

Parameter	Small-Scale analysis	Scalability analysis
$m \triangleq \lvert\mathcal{W}_P\rvert$	10	100, 500, 1000
$\lvert\mathcal{A}_w\rvert$	1	2, 4, 6, 8, 10
$\lvert\mathcal{S}_a\rvert$	5	5, 10
$\lvert\mathcal{R}_s\rvert$	5	5, 10
T_P (seconds)	14.10358	281.68, 1406.26, 2827.6
E_P (Wh)	0.000961	0.0152, 0.0762, 0.153
$\hat{e}$	Uniform in [0.2, 0.29] Wh	
$\hat{a}$	Uniform in [0.5, 0.99]	
$\hat{t}$	Uniform in [1, 2] s	

Table 1: System parameters.

6.2 Experimental Results

We first examine the accuracy behavior of the two objective formulations on small-scale workflows with ten sequential steps. Figure 2 illustrates the stepwise accuracy profile for a representative instance. As expected, the max-min formulation produces a relatively uniform accuracy distribution across steps, reflecting its objective of maximizing the weakest-performing stage. In contrast, the multiplicative formulation exhibits greater variability across steps, allocating additional resources to stages where accuracy improvements have a larger impact on the overall product. As a result, the multiplicative objective achieves higher end-to-end accuracy in this instance, at the cost of increased execution time and energy consumption, as additional resources are devoted to accuracy-sensitive steps. This highlights the fundamental difference between the two formulations: the max-min objective prioritizes robustness against single-step failures under conservative resource

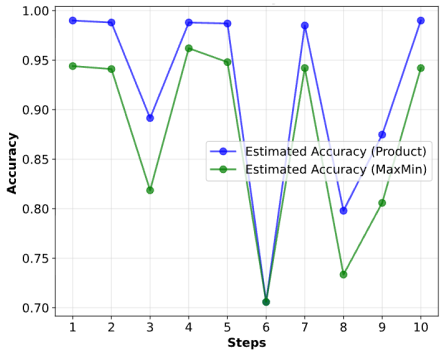

Figure 2: Per-step accuracy in a representative instance.

usage, while the multiplicative objective emphasizes cumulative accuracy gains across the reasoning chain, even when this requires higher time and energy expenditure. Figure 3 reports the evolution of cumulative execution time and energy consumption across steps, averaged over ten instances. Both formulations always satisfy the global time and energy budgets in all experiments. This holds despite the 20260122±10% variability between estimated and measured resource costs introduced to account for runtime uncertainty. However, the multiplicative formulation consistently incurs higher total time and energy consumption than the max-min formulation. This behavior is consistent with its objective: by rewarding accuracy improvements at any step, the multiplicative objective tends to allocate more resources to accuracy-sensitive stages, whereas the max-min formulation often refrains from further resource investment once the minimum accuracy threshold has been sufficiently improved. For both objectives, the estimated and realized resource consumptions closely track each other, indicating that the iterative re-optimization strategy effectively adapts to deviations between predicted and measured costs.

Finally, Table 2 reports the results of the scalability analysis. As the number of steps and configuration options increases, the average solving time grows approximately linearly with the number of steps and the size of the configuration space. Even for workflows with up to 1000 steps and large configuration sets, both formulations remain solvable within practical time limits. The multiplicative formulation exhibits slightly higher computational overhead than the max-min formulation (about 5% but evident only for the largest instances), reflecting the increased complexity of its objective. Notably, prior studies indicate that approximately the 85% of production agentic workflows consist of at most tens of steps, with the majority completing fewer than ten steps [21]. Consequently, our largest evaluated instances exceed typical real-world workloads by more than an order of magnitude, providing strong evidence that the proposed framework scales well beyond practical deployment requirements while exposing meaningful trade-offs between robustness, accuracy, and resource consumption.

7 Conclusion

This paper presented an optimization framework for orchestrating heterogeneous AI agents in sequential, multi-step pipelines under global time and energy constraints. By explicitly modeling

$\lvert\mathcal{W}_P\rvert$	$\lvert\mathcal{A}_w\rvert$	$(\lvert\mathcal{S}_a\rvert, \lvert\mathcal{R}_s\rvert)$	Method	Solving time (s)
100	2	(5,5)	Product	0.37
			Max-Min	0.39
		(10,10)	Product	1.58
			Max-Min	1.48
	4	(5,5)	Product	0.79
			Max-Min	0.74
		(10,10)	Product	3.17
			Max-Min	3.11
	6	(5,5)	Product	1.18
			Max-Min	1.13
		(10,10)	Product	4.94
			Max-Min	4.72
	8	(5,5)	Product	1.59
			Max-Min	1.50
		(10,10)	Product	6.63
			Max-Min	6.34
	10	(5,5)	Product	1.97
			Max-Min	1.92
		(10,10)	Product	8.47
			Max-Min	8.04
500	2	(5,5)	Product	2.01
			Max-Min	1.92
		(10,10)	Product	8.44
			Max-Min	7.86
	4	(5,5)	Product	4.11
			Max-Min	3.94
		(10,10)	Product	17.20
			Max-Min	16.39
	6	(5,5)	Product	6.33
			Max-Min	6.12
		(10,10)	Product	25.89
			Max-Min	24.76
	8	(5,5)	Product	8.49
			Max-Min	8.26
		(10,10)	Product	36.08
			Max-Min	34.62
	10	(5,5)	Product	10.59
			Max-Min	10.21
		(10,10)	Product	48.09
			Max-Min	45.21
1000	2	(5,5)	Product	4.20
			Max-Min	4.03
		(10,10)	Product	17.01
			Max-Min	16.65
	4	(5,5)	Product	8.53
			Max-Min	8.14
		(10,10)	Product	34.75
			Max-Min	33.14
	6	(5,5)	Product	14.24
			Max-Min	13.50
		(10,10)	Product	57.17
			Max-Min	54.19
	8	(5,5)	Product	19.25
			Max-Min	18.23
		(10,10)	Product	73.29
			Max-Min	70.06
	10	(5,5)	Product	22.82
			Max-Min	21.96
		(10,10)	Product	96.22
			Max-Min	90.49

Table 2: Scalability analysis, time averaged across 10 random instances.

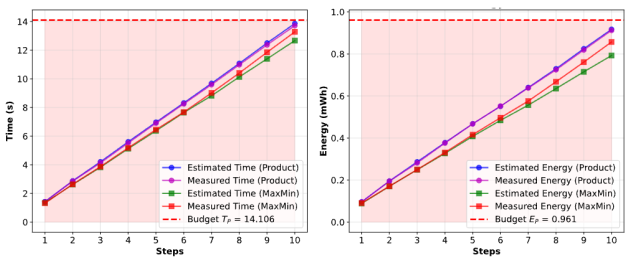

Figure 3: Time and energy consumption, averaging across 10 random instances.

agent selection (with associated inference strategy and parameter configuration), the framework captures the key trade-offs between accuracy, latency, and energy that arise in practical agentic AI systems. We introduced two complementary objective formulations reflecting different notions of robustness. The max-min accuracy objective prioritizes reliability by improving the weakest step in the pipeline, while the multiplicative accuracy objective emphasizes cumulative accuracy gains across tasks. Experimental results highlight the distinct behaviors induced by these objectives: the multiplicative formulation achieves higher overall accuracy at the cost of increased time and energy consumption, whereas the max-min formulation yields more conservative and resource-efficient solutions. Overall, the proposed approach provides a principled basis for resource-aware orchestration in agentic AI systems and scales effectively to large workflows. Future work will explore more complex invocation topologies, richer performance models, dynamic agent availability, and learning-based optimization strategies.

References

[1] Alfonso Amayuelas, Jingbo Yang, Saaket Agashe, Ashwin Nagarajan, Antonis Antoniades, Xin Eric Wang, and William Wang. 2025. Self-Resource Allocation in Multi-Agent LLM Systems. arXiv:2504.02051 [cs.MA] https://arxiv.org/abs/2504.02051

[2] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631* (2023).

[3] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small Language Models are the Future of Agentic AI. arXiv:2506.02153 [cs.AI] https://arxiv.org/abs/2506.02153

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176 [cs.LG] https://arxiv.org/abs/2305.05176

[5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

[6] Jasper Dekoninck, Maximilian Baader, and Martin Vechev. 2025. A Unified Approach to Routing and Cascading for LLMs. In *PMLR*. https://arxiv.org/abs/2410.10347

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[8] Cooper Elsworth, Keguo Huang, David Patterson, Ian Schneider, Robert Sedivy, Savannah Goodman, Ben Townsend, Parthasarathy Ranganathan, Jeff Dean, Amin Vahdat, Ben Gomes, and James Manyika. 2025. Measuring the environmental impact of delivering AI at Google Scale. arXiv:2508.15734 [cs.AI] https://arxiv.org/abs/2508.15734

[9] Zhouhong Gu, Xiaoxuan Zhu, Yin Cai, Hao Shen, Xingzhou Chen, Qingyi Wang, Jialin Li, Xiaoran Shi, Haoran Guo, Wenxuan Huang, et al. 2025. AgentGroupChat-V2: Divide-and-Conquer Is What LLM-Based Multi-Agent System Need. *arXiv preprint arXiv:2506.15451* (2025).

[10] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654* (2020).

[11] Keke Huang, Yimin Shi, Dujian Ding, Yifei Li, Yang Fei, Laks Lakshmanan, and Xiaokui Xiao. 2025. ThriftLLM: On Cost-Effective Selection of Large Language Models for Classification Queries. *Proc. VLDB Endow.* 18, 11 (July 2025), 4410–4423. doi:10.14778/3749646.3749702

[12] Bowen Jin, TJ Collins, Donghan Yu, Mert Cemri, Shenao Zhang, Mengyu Li, Jay Tang, Tian Qin, Zhiyang Xu, Jiarui Lu, et al. 2025. Controlling Performance and

Budget of a Centralized Multi-agent LLM System with Reinforcement Learning. *arXiv preprint arXiv:2511.02755* (2025).

[13] Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, and Kenshi Abe. 2025. Regularized best-of-n sampling with minimum bayes risk objective for language model alignment. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 9321–9347.

[14] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models, 2022. *URL https://arxiv. org/abs/2206.14858* 1 (2022).

[15] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

[16] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634* (2023).

[17] Yueyue Liu, Hongyu Zhang, Zhiqiang Li, and Yuantian Miao. 2024. CPLS: Optimizing the Assignment of LLM Queries. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 151–162.

[18] Yueyue Liu, Hongyu Zhang, Yuantian Miao, Van-Hoang Le, and Zhiqiang Li. 2024. OptLLM: Optimal Assignment of Queries to Large Language Models. In *2024 IEEE International Conference on Web Services (ICWS)*. 788–798.

[19] Silvano Martello and Paolo Toth. 1990. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., USA.

[20] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. [n. d.]. RouteLLM: Learning to Route LLMs from Preference Data. In *The Thirteenth International Conference on Learning Representations*.

[21] Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, Shu Liu, Tianneng Shi, Xiaoyuan Liu, Jared Quincy Davis, Emmanuele Lacavalla, Alessandro Basile, Shuyi Yang, Paul Castro, Daniel Kang, Joseph E. Gonzalez, Koushik Sen, Dawn Song, Ion Stoica, Matei Zaharia, and Marquita Ellis. 2025. Measuring Agents in Production. arXiv:2512.04123 [cs.CY] https://arxiv.org/abs/2512.04123

[22] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[23] Jitao Sang, Jinlin Xiao, Jiarun Han, Jilin Chen, Xiaoyi Chen, Shuyu Wei, Yongjie Sun, and Yuhang Wang. 2025. Beyond Pipelines: A Survey of the Paradigm Shift toward Model-Native Agentic AI. arXiv:2510.16720 [cs.AI] https://arxiv.org/abs/2510.16720

[24] H. Sedghani, D. Ardagna, M. Matteucci, G.A. Fontana, G. Verticale, F. Amarilli, R. Badia, D. Lezzi, I. Blanquer, A. Martin, and K. Wawruch. 2021. Advancing Design and Runtime Management of AI Applications with AI-SPRINT (Position Paper). In *2021 IEEE 45th COMPSAC*. 1455–1462.

[25] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv:2408.03314 [cs.LG] https://arxiv.org/abs/2408.03314

[26] David Alejandro Torrado Guzman. 2025. *Development and Evaluation of AI Agentic Workflows in LangGraph: An Energy and Computational Performance Analysis*. Master's thesis. M.Sc. Computer Science Engineering - Politecnico di Milano. https://www.politesi.polimi.it/retrieve/71d42dc3-2735-4b07-851f-a24cda63d9d8/Torrado_7_2025.pdf

[27] Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322* (2025).

[28] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[29] Liming Yang, Junyu Luo, Xuanzhe Liu, Yiling Lou, and Zhenpeng Chen. 2025. BAMAS: Structuring Budget-Aware Multi-Agent Systems. *arXiv preprint arXiv:2511.21572* (2025).

[30] Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. 2024. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6344–6355.

[31] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.