Before You Build a Multi-Agent System: An Escalation Framework for LLM Adaptation

Junhyeong Lee[1], Joon-Young Kim[2,3], and Seunghwa Ryu[1,3,4,*]

[1]KAIST InnoCORE PRISM-AI Center, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea
[2]Industrial Intelligence Research Group, AI/DX Center, Institute for Advanced Engineering (IAE), Yongin, Republic of Korea
[3]Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea
[4]Department of AX, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea
[*]Corresponding author: ryush@kaist.ac.kr

Abstract

Multi-agent systems (MAS) have become a popular framework for deploying large language models (LLMs), yet their operational complexity—increased latency, compounding errors, and difficult-to-optimize orchestration—is often unnecessary. Beyond MAS, a rich ecosystem of LLM adaptation strategies exists—spanning input-level methods, parameter updates, and harness-based orchestration—yet few principled frameworks guide practitioners who deploy LLMs in real-world systems in choosing among them. To address this gap, we view LLMs as parametric mappings, which makes explicit three adaptation handles ordered by complexity and cost: input-level adaptation ($\mathbf{X}$), parameter-level adaptation (θ), and harness-based orchestration ($\mathcal{H}$). Building on this view, we propose an escalation framework that guides practitioners in selecting the most appropriate adaptation strategy for a given task—and, crucially, in exhausting simpler levels before ascending to more complex ones.

Keywords: Large language models; Multi-agent systems; Fine-tuning; Prompt engineering; LLM adaptation

1 Introduction

Large language models (LLMs) are being adopted across a remarkably wide range of domains—from drug discovery and materials design to software engineering, legal analysis, and scientific writing [1–5]. As this adoption scales, multi-agent systems (MAS)—networks of coordinated LLM calls, tools, and retrievers—have emerged as an increasingly popular architectural pattern.

The appeal is intuitive: specialized agents can divide labor, run in parallel, and access external tools that a single model call cannot.

Yet this architectural complexity comes at a cost. Each additional agent increases latency and token expenditure [6], errors in early pipeline stages propagate and compound downstream [7], and the orchestration layer itself becomes a non-trivial engineering burden. What is often missing is a principled basis for deciding *when* MAS is actually warranted—and when a simpler intervention would suffice.

A rich ecosystem of adaptation techniques exists below the MAS level: prompt engineering [8, 9], retrieval-augmented generation (RAG) [10], supervised fine-tuning (SFT) [11], and parameter-efficient adaptation [12, 13]. Dedicated surveys cover each family in depth [14–17], yet few provide practitioners with a unified framework for deciding which intervention to try first—or when to escalate.

We propose an escalation framework grounded in treating LLMs as parametric mappings. Treating the LLM as $f_\theta : \mathcal{V}^* \to \Delta^{|\mathcal{V}|-1}$ makes visible three handles a practitioner can adjust: the input $\mathbf{X}$, the model parameters θ, and the harness $\mathcal{H}$. These handles differ not only in mechanism but in the operational complexity they introduce. The argument of this paper is that each level should be fully explored before ascending to the next.

The remainder of this perspective is organized as follows. Section 2 formalizes the LLM as a parametric mapping, identifying the three adaptation handles ($\mathbf{X}$, θ, $\mathcal{H}$) that structure the framework. Sections 3 and 4 cover input-level adaptation (prompt engineering, RAG) and parameter-level adaptation (SFT, PEFT, preference optimization), respectively. Section 5 examines harness-level adaptation. Section 6 synthesizes these strategies into a practical escalation framework.

2 LLMs as Parametric Mappings

Data-driven deep learning is, at its core, the task of learning a parametric map $f_\theta : \mathcal{X} \to \mathcal{Y}$ fitted to example pairs $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$ [18]. Despite the diversity of architectures, they share this structure: a parameterized mapping, fitted to data, that produces outputs for new inputs via patterns extracted from training data.

Like other deep learning models, LLMs can be understood within this same mapping framework. Admittedly, LLMs encompass a variety of architectures; however, decoder-only transformers trained with a next-token prediction objective [19] represent the dominant paradigm in practical deployment, and we accordingly restrict our scope to this class.

Specifically, an LLM with parameters θ is a mapping

$$f_\theta : \mathcal{V}^* \longrightarrow \Delta^{|\mathcal{V}|-1} \tag{1}$$

from sequences of tokens drawn from vocabulary $\mathcal{V}$ to a probability distribution over the next token—that is, a point on the $(|\mathcal{V}| - 1)$-dimensional probability simplex $\Delta^{|\mathcal{V}|-1}$. Each point in $\Delta^{|\mathcal{V}|-1}$ is a vector of $|\mathcal{V}|$ non-negative values summing to one, with each coordinate giving the probability assigned to the corresponding token. For instance, given the prompt "The capital of France is", a well-trained model concentrates most of this probability mass on the single token

"Paris". Here $\mathcal{V}$ is the finite set of all tokens recognized by the model's tokenizer (typically on the order of 100K–256K tokens [20]), and $\mathcal{V}^*$ denotes the set of finite token sequences over $\mathcal{V}$; in practice, inputs are bounded by the model's context window.

Tracing how the model processes an input string to produce a next-token prediction reveals the following sequential stages (Figure 1a):

$$s \xrightarrow{\tau} \mathbf{X} \xrightarrow{\mathbf{E}_\theta} \mathbf{H}^{(0)} \xrightarrow{\Phi_\theta} \mathbf{H} \xrightarrow{W_U,\,\text{softmax}} f_\theta(\mathbf{X}) \xrightarrow{\text{dec}} x_{n+1} \tag{2}$$

The tokenizer $\tau : \mathcal{S} \to \mathcal{V}^*$ [21] converts a string s into a token sequence $\mathbf{X} = (x_1, \ldots, x_n)$; the embedding map $\mathbf{E}_\theta$ lifts each token to a d-dimensional vector and adds positional information [22], yielding $\mathbf{H}^{(0)} \in \mathbb{R}^{n \times d}$; the transformer body Φ_θ [23] applies stacked causally-masked self-attention and feed-forward layers to produce $\mathbf{H}$; the language-model head W_U projects the last-position vector $\mathbf{H}_n$ to vocabulary logits, and softmax normalizes them to $f_\theta(\mathbf{X}) \in \Delta^{|\mathcal{V}|-1}$; finally, a dec$(\cdot)$ [24, 25] selects the next token x_{n+1} from this distribution. Repeating this pipeline—each new token appended to the context—yields the complete output sequence [26], formalized in Eq. (3):

$$\mathbf{Y}_\theta(\mathbf{X}) = (x_{n+1}, \ldots, x_{n+T}), \quad x_{n+t} \sim f_\theta(x_1, \ldots, x_{n+t-1}) \tag{3}$$

where each $x_{n+t} \in \mathcal{V}$ is the t-th generated token, $n = |\mathbf{X}|$ is the input length, and T is the number of generated tokens (with x_{n+T} the end-of-sequence token).

Viewing the LLM as the mapping $f_\theta(\mathbf{X})$, two adaptation targets become immediately visible: the input $\mathbf{X}$—what the model receives at inference time—and the parameters θ—what the mapping itself computes. When multiple LLMs are composed with retrievers and tools, a third handle emerges: the coordination structure governing how these components interact, which we call the *harness* $\mathcal{H}$. These three handles form an escalation ladder ordered by operational complexity and cost (Figure 1b).

3 Input-Level Adaptation

Input-level adaptation operates on $\mathbf{X}$ without modifying θ: the goal is to supply the model with what it needs to behave well—through careful instruction, illustrative examples, or retrieved knowledge—rather than changing what the model itself is. Two representative directions emerge: optimizing the composition of $\mathbf{X}$ itself (how to instruct and demonstrate), and supplying the necessary information into $\mathbf{X}$ (what to retrieve or recall).

3.1 Prompt Optimization

Early techniques manipulated the prompt directly—task instructions, role assignment, few-shot exemplars (ICL [8, 27]), or chain-of-thought prompting [9]—to elicit better responses [28, 29]. More systematic approaches treat $\mathbf{X}$ as the design variable and optimize it against a task metric J over a prompt search space $\mathcal{P} \subseteq \mathcal{V}^*$:

$$\mathbf{X}_{\text{opt}} = \arg\max_{\mathbf{X} \in \mathcal{P}} J(\mathbf{Y}_\theta(\mathbf{X})) \tag{4}$$

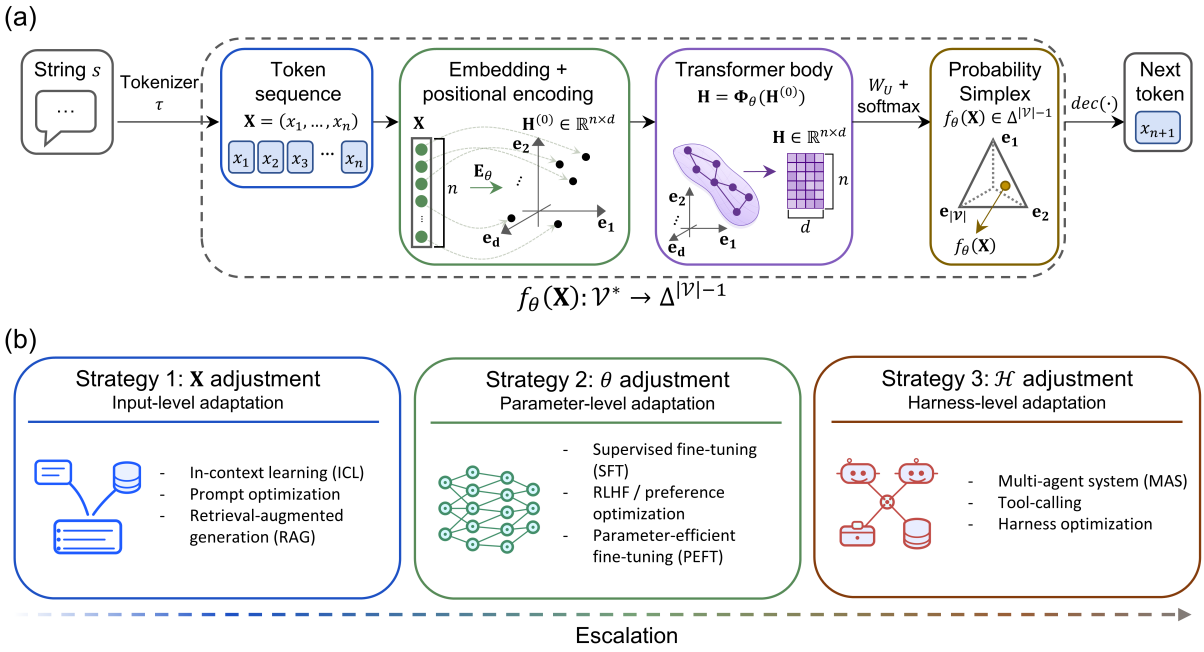

Figure 1: **LLM-as-mapping overview and the escalation framework.** (a) The LLM inference pipeline from input string to next token. (b) The three adaptation strategies ordered by increasing complexity and cost: input-level ($\mathbf{X}$), parameter-level (θ), and harness-level ($\mathcal{H}$).

Methods such as APE [30] and OPRO [31] automate this search, using the LLM itself to propose and score candidate instructions; frameworks such as DSPy [32] and TextGrad [33] extend this optimization to multi-step LLM pipelines.

3.2 Retrieval-augmented generation (RAG)

Prompt optimization shapes *how* $\mathbf{X}$ is composed, but cannot supply knowledge that is absent from θ. A complementary approach is to inject task-relevant knowledge directly into $\mathbf{X}$: rather than relying solely on the model's parametric memory (θ), RAG [10] retrieves passages from an external corpus at query time and supplies them as grounding context, reducing hallucination by anchoring generation in retrieved evidence.

To implement this, RAG relies on two auxiliary components. A *sentence embedding model* ϕ_η [34] maps a token sequence to a d_e-dimensional vector such that semantically similar sequences map to nearby points:

$$\phi_\eta : \mathcal{V}^* \to \mathbb{R}^{d_e} \tag{5}$$

A *retriever* [35] uses ϕ_η to return the k passages from a corpus $\mathcal{K}$ most similar to a query:

$$R(\mathbf{X};\, \eta,\, \mathcal{K}) = \text{top-}\, k_{\mathbf{z} \in \mathcal{K}}\ \text{sim}(\phi_\eta(\mathbf{X}),\, \phi_\eta(\mathbf{z})) \tag{6}$$

Using these components, RAG augments $\mathbf{X}$ with retrieved passages and feeds the augmented input $\mathbf{X}_{\text{aug}} = [\mathbf{I};\, \mathbf{Z};\, \mathbf{X}]$ to the model, where $\mathbf{I}$ is the instruction, $\mathbf{Z}$ the retrieved passages, and $\mathbf{X}$ the original query. In this way, RAG extends the effective knowledge horizon of a frozen model

beyond its parametric memory without any weight update. Subsequent work has advanced RAG in diverse directions: improving retrieval quality and reranking; enabling adaptive or iterative retrieval [36, 37]; and moving toward LLM-driven synthesis over document hierarchies or knowledge graphs [38, 39].

3.3 Practical Considerations

Input-level adaptation is especially valuable when knowledge is dynamic or proprietary. Because both prompt engineering and RAG operate entirely at inference time and require no parameter updates, they offer a relatively low-cost pathway for incorporating new knowledge.

However, adopting these methods in practice comes with notable concerns. RAG must query an external corpus at inference time, introducing additional latency in production settings. Retrieved documents may contain sensitive or proprietary information, raising the risk of unintended data disclosure [40], and the composition of the retrieved context strongly shapes output quality—distracting or low-relevance passages can degrade the generated answer [41]. When the task demands persistent behavioral change or when the output quality remains insufficient despite careful prompting and retrieval, the practitioner should consider parameter-level adaptation.

4 Parameter-Level Adaptation

Pre-trained models encode broad competence, but many tasks require more specific behavior—domain vocabulary, consistent output format, or alignment with human preferences not seen during pre-training. In addition, in settings where data cannot leave a secure environment—such as defense applications or clinical records—local adaptation may be the only viable option, as it keeps both data and the adaptation process within the secure environment [42]. Parameter-level adaptation addresses this by updating θ directly. Starting from a pre-trained checkpoint θ_0, the goal is to find θ_{opt} that better fits the target distribution; the two dominant families differ in what training signal is available.

4.1 Supervised fine-tuning (SFT)

Given a dataset $\mathcal{D} = \{(\mathbf{X}^{(i)}, \mathbf{Y}^{(i)})\}_{i=1}^{N}$—curated by humans or bootstrapped from a stronger model [43]—SFT [11] typically minimizes the negative log-likelihood over the output tokens, where $p_\theta(y_t \mid y_{<t}, \mathbf{X})$ denotes the model's predicted probability for the t-th token of $\mathbf{Y}$:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\,\mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim\mathcal{D}_{\text{SFT}}}\left[\sum_{t=1}^{|\mathbf{Y}|} \log p_\theta(y_t \mid y_{<t}, \mathbf{X})\right] \tag{7}$$

Minimizing $\mathcal{L}_{\text{SFT}}$ trains θ to reproduce the output distribution $\mathbf{Y}$ given input $\mathbf{X}$: the model learns to imitate expert behavior token by token.

However, full-weight fine-tuning is computationally expensive, motivating parameter-efficient fine-tuning (PEFT) methods [12] that keep θ_0 frozen and train only a small increment $\Delta\theta$. A representative instance is Low-Rank Adaptation (LoRA) [13], which approximates weight up-

dates by retaining only the low-rank components, drastically reducing the number of trainable parameters. QLoRA [44] further quantizes the frozen backbone to low precision, enabling adaptation in resource-constrained settings.

4.2 Preference-based Optimization

SFT presupposes that the right output can be written down; many tasks, however, are easier to evaluate than to label. When feedback takes the form of human preferences or verifier scores rather than labeled sequences, preference-based optimization is the appropriate training framework [45–47].

Several methods have been proposed along this line. Reinforcement learning from human feedback (RLHF) [48] uses a reward model and policy gradient to steer the model toward preferred outputs; representative RL-based methods include Proximal Policy Optimization (PPO) [49] and Group Relative Policy Optimization (GRPO) [50]. Direct Preference Optimization (DPO) [51] offers another preference optimization method that bypasses the reward model.

As a representative example of preference-based optimization, DPO optimizes the following objective given preference pairs $(\mathbf{Y}_w, \mathbf{Y}_l)$ where $\mathbf{Y}_w$ is preferred over $\mathbf{Y}_l$:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\,\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(\mathbf{Y}_w|\mathbf{X})}{\pi_{\text{ref}}(\mathbf{Y}_w|\mathbf{X})} - \beta \log \frac{\pi_\theta(\mathbf{Y}_l|\mathbf{X})}{\pi_{\text{ref}}(\mathbf{Y}_l|\mathbf{X})}\right)\right] \tag{8}$$

where σ is the logistic sigmoid function, β is a temperature parameter that controls how far π_θ is allowed to deviate from π_{ref}, $\pi_\theta(\mathbf{Y}|\mathbf{X})$ is the probability of generating the complete output sequence $\mathbf{Y}$ given $\mathbf{X}$, obtained by multiplying the per-token probabilities from f_θ at each decoding step, and π_{ref} is a frozen copy of that distribution before training. Across these methods, the shared principle is to update θ by designing objectives that reward preferred outputs—steering the model toward better responses without requiring labeled target sequences.

4.3 Practical Considerations

Modifying θ is appropriate when the behavioral gap between the base model and the target task is too large to close through prompting alone, or when consistent behavioral change is required across all deployments. For tasks that are not overly complex, or when working with small language models (SLMs) [52], this approach can be effective even with modest computational resources.

However, updating θ is not without drawbacks. Even with PEFT methods, fine-tuning large models remains computationally demanding, often requiring multiple high-end GPUs and substantial memory—hardware accessible only in well-resourced settings. A further limitation is inflexibility to new information: once θ is fixed, incorporating updated knowledge requires another training run, making real-time or frequent data updates difficult to accommodate [53]. When the task is structurally complex—requiring decomposition, parallelism, or live tool access that a single model call cannot provide—parameter-level adaptation alone is insufficient, and harness-based orchestration should be considered.

5 Harness-Level Adaptation: Multi-Agent Systems

Long-horizon or multi-step tasks cannot be addressed by controlling a single LLM through θ or $\mathbf{X}$ alone. The solution is a *harness*: an orchestrator that coordinates model calls, tools, and retrievers under a controller policy π:

$$\mathcal{H} = (\{f_{\theta_i}\},\ \{T_j\},\ \{R_k\},\ \pi) \tag{9}$$

where $\{f_{\theta_i}\}$ is a set of sub-LLMs, $\{T_j\}$ a set of tools, and $\{R_k\}$ a set of retrievers; we write $\mathcal{C} = \{f_{\theta_i}\} \cup \{T_j\} \cup \{R_k\}$ for the full set of available components. A *tool* performs external computation (e.g. a calculator, code interpreter, or simulator), while a *retriever* returns relevant passages from an external corpus. Whatever a component produces internally—a number, a file, or a passage—is serialized back into a token sequence before re-entering the context, so at the model-facing interface both share the signature $\mathcal{V}^* \to \mathcal{V}^*$ and are handled uniformly by the controller. The controller π selects which components to invoke at each step:

$$\pi : 2^{\mathcal{C}} \times \mathcal{V}^* \to 2^{\mathcal{C}} \tag{10}$$

π may be a hard-coded script, a rule-based router, or an LLM that dynamically decides which component to invoke next [54], implemented through frameworks such as LangGraph [55] or AutoGen [56].

5.1 Multi-agent systems

A multi-agent system (MAS) is a specialization of $\mathcal{H}$ in which each model-call slot f_{θ_i} is replaced by a full agent A_i—a harness instance in its own right, with its own backbone LLM, role prompt, tool subset, and memory buffer:

$$\mathcal{M} = (\mathcal{A},\ \mathcal{T},\ \mathcal{P},\ \pi_{\text{coord}}) \tag{11}$$

where $\mathcal{A} = \{A_i\}$ is the set of agents, $\mathcal{T}$ their collective tools, $\mathcal{P}$ their role prompts, and π_{coord} the policy coordinating interactions among agents. Common patterns include planner–executor pairs, multi-agent debate [57], and hierarchical supervisor–worker teams [16, 58]. Frameworks such as ReAct [54], AutoGen [56], and LangGraph [55] provide practical scaffolding for these patterns.

5.2 Practical Considerations

Harness-level adaptation is appropriate when a task genuinely requires decomposition, parallelism, or live tool access that a single model call cannot provide—long-horizon research, multi-step code generation, or pipelines where independent verification by a separate agent improves reliability.

However, the operational costs are real and frequently underestimated. Each sub-call adds latency and token cost [6], and errors propagate: a mistake in an early call can corrupt all downstream reasoning. Indeed, for narrow, well-defined tasks, a carefully fine-tuned single model frequently outperforms a multi-agent pipeline [7]. Beyond deployment costs, optimizing

the harness composition is non-trivial: the search space grows combinatorially with the number of agents and tools, and automated topology search and multi-step pipeline debugging remain open problems [59, 60].

6 An Escalation Framework

The preceding sections introduced three handles through which practitioners can adapt LLMs to a target task: the input context ($\mathbf{X}$), the model parameters (θ), and the harness ($\mathcal{H}$). These handles differ not only in mechanism but in the operational overhead they impose. This overhead is not a single scalar: parameter-level adaptation front-loads a one-time training cost but leaves inference simple, whereas harness-based orchestration adds little training cost yet imposes recurring inference latency and engineering complexity on every deployed call.

This section synthesizes them into a practical escalation framework that orders the handles by the operational complexity they add to a deployed system and by deployment context. Rather than prescribing a fixed sequence, we illustrate through concrete scenarios how the appropriate escalation path shifts with the deployment context, to help practitioners build intuition for the choice; Table 1 summarizes the key considerations for each strategy.

Table 1: Escalation framework for LLM adaptation.

Strategy	Handle	Try when	Escalate when	Key methods
1	$\mathbf{X}$ (Input)	Task specifiable by instruction; knowledge is dynamic or proprietary	Behavioral gap persists; output quality insufficient after prompting	Prompt eng., RAG
2	θ (Parameters)	Persistent behavioral adaptation needed; consistent output format or style required	Task requires decomposition, parallelism, or live tool access	SFT, LoRA, DPO, GRPO
3	$\mathcal{H}$ (Harness)	Task genuinely requires multi-step reasoning, tool use, or independent verification	—	MAS, RAG pipelines, harness optimization

For example, the appropriate escalation path depends on the deployment context, particularly the sensitivity of the data involved. When data sensitivity is low—such as in cloud API deployments—the full range of strategies is available. Input-level adaptation ($\mathbf{X}$) is the natural starting point, as it requires no training and can leverage external retrieval services for dynamic knowledge. If the behavioral gap persists after prompting and retrieval, parameter-level adaptation (θ) provides persistent behavioral change without orchestration overhead. The choice between θ and $\mathcal{H}$, however, is also a cost comparison: when the target model is too large to fine-tune under the available compute, a harness that leaves θ frozen can be the more practical route even absent a strong task-structure reason. Where adapting θ is feasible, harness-based orchestration ($\mathcal{H}$) becomes appropriate when the practitioner is willing to accept greater latency and cost in exchange for quality gains that require decomposition, parallel execution, or live tool access.

Conversely, when data sensitivity is high—such as in on-premise or regulated deployments—external retrieval may be restricted. This constrains the RAG component of input-level adaptation, but prompt optimization requires no external corpus and remains the natural first step. When prompting alone proves insufficient, parameter-level adaptation (θ) becomes the primary lever for behavioral alignment, and harness-based orchestration ($\mathcal{H}$) is confined to internally hosted infrastructure.

Across these contexts a broad tendency holds, but the escalation ordering is best treated as

a guideline, not a dogma—the appropriate path shifts with the deployment situation. When a task's structure makes decomposition and heterogeneous tool use intrinsic from the outset—an assistant that must browse the web, execute code, and file tickets within a single request—beginning directly at $\mathcal{H}$ is the correct call rather than a premature escalation. The point is that MAS should be treated as a considered choice justified by task structure—not a default architectural pattern adopted for its generality.

Beyond these deployment paths, the three handles are not mutually exclusive: they combine in recurring patterns that reveal cross-strategy synergies. For instance, θ can absorb harness capabilities through tool-aware fine-tuning, eliminating the need for an external orchestrator (Toolformer [61]). Similarly, $\mathbf{X}$ and θ can reinforce each other: fine-tuning a model on retrieval-augmented inputs teaches it to use retrieved context more effectively (RAFT [62]). Furthermore, the handles can co-evolve to form a *self-evolving system*—a closed loop in which each iteration's outputs become the next iteration's training signal: STaR [63] and DeepSeek-R1 [47] refine θ through self-generated rationales or reinforcement, while AlphaEvolve [64] holds θ fixed and evolves the programs that form $\mathcal{H}$'s tools and orchestration. As these examples suggest, the levels can also be composed rather than used one at a time, combining their complementary strengths.

Conclusion

This perspective has proposed an escalation framework for LLM adaptation, grounded in treating LLMs as parametric mappings. This view makes explicit three adaptation strategies—input-level adaptation, parameter-level adaptation, and harness-based orchestration—ordered by increasing complexity and cost. The central argument is that input- and parameter-level methods, often sufficient yet underestimated in practice, deserve to be exhausted before more complex orchestration. This does not condemn harness-based orchestration, including multi-agent systems. Rather, because it carries real costs in latency, error propagation, and engineering overhead, it should be justified by task structure rather than adopted by default for its generality.

The present framework is not without limitations. The scope is restricted to autoregressive, decoder-only language models [17]; encoder-only [65] and encoder-decoder [66] architectures fall outside the next-token prediction assumed here. Multimodal models—vision-language, audio-language, and scientific foundation models—require extending $\mathcal{V}^*$ beyond text token sequences, which the current formalism does not accommodate [67, 68]. Extending the framework to these settings, and systematically characterizing which strategy is most appropriate for which class of task, remain natural directions for future work.

We hope this framework offers practitioners a useful reference: selecting the right level of adaptation, rather than jumping straight to a multi-agent system, remains central to principled LLM deployment. We further hope that extensions along the directions above will sharpen it over time.

Author Contributions

J.L. contributed to conceptualization, formal analysis, and methodology, and wrote the original draft. J.K. contributed to writing – review & editing. S.R. contributed to conceptualization, acquired funding, administered the project, and supervised the work, and contributed to writing – review & editing.

Acknowledgements

This work was supported by the InnoCORE program of the Ministry of Science and ICT (N10260002).

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

No datasets were generated or analysed during the current study.

References

[1] Zheng, Y. *et al.* Large language models in drug discovery and development: From disease mechanisms to clinical trials. *arXiv preprint arXiv:2409.04481* (2024).

[2] Zhang, L., Liu, Z., Ni, B. & Wang, Q. Large language models (llms) for materials design. *Advanced Functional Materials* **36**, e2525897 (2026).

[3] Fan, A. *et al.* Large language models for software engineering: Survey and open problems. In *IEEE/ACM Int. Conf. on Software Engineering: Future of Software Engineering (ICSE-FoSE)* (2023).

[4] Lai, J. *et al.* Large language models in law: A survey. *arXiv preprint arXiv:2312.03718* (2023).

[5] Luo, Z. *et al.* LLM4SR: A survey on large language models for scientific research. *arXiv preprint arXiv:2501.04306* (2025).

[6] Chen, L. *et al.* Are more LLM calls all you need? towards the scaling properties of compound AI systems. In *Advances in Neural Information Processing Systems (NeurIPS)* (2024).

[7] Cemri, M. *et al.* Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657* (2025).

[8] Brown, T. B. *et al.* Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, vol. 33, 1877–1901 (Curran Associates, Inc., 2020).

[9] Wei, J. *et al.* Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, vol. 35, 24824–24837 (Curran Associates, Inc., 2022).

[10] Lewis, P. *et al.* Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, vol. 33, 9459–9474 (Curran Associates, Inc., 2020).

[11] Wei, J., Bosma, M., Zhao, V. Y. *et al.* Finetuned language models are zero-shot learners. In *International Conference on Learning Representations (ICLR)* (2022).

[12] Houlsby, N., Giurgiu, A., Jastrzebski, S. *et al.* Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning (ICML)*, 2790–2799 (2019).

[13] Hu, E. J. *et al.* LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations* (2022).

[14] Ding, N. *et al.* Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904* (2023).

[15] Gao, Y. *et al.* Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2024).

[16] Wang, L. *et al.* A survey on large language model based autonomous agents. *Frontiers of Computer Science* **18**, 186345 (2024).

[17] Zhao, W. X. *et al.* A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[18] Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016). `http://www.deeplearningbook.org`.

[19] Radford, A. *et al.* Language models are unsupervised multitask learners. Tech. Rep., OpenAI (2019). OpenAI Blog, 1(8):9.

[20] OpenAI. GPT-4 technical report. Tech. Rep., OpenAI (2023). ArXiv:2303.08774.

[21] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1715–1725 (2016).

[22] Su, J. *et al.* RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864* (2021). Published in Neurocomputing, 568:127063, 2024.

[23] Vaswani, A. *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems*, vol. 30 (Curran Associates, Inc., 2017).

[24] Fan, A., Lewis, M. & Dauphin, Y. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 889–898 (2018).

[25] Holtzman, A., Buys, J., Du, L., Forbes, M. & Choi, Y. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)* (2020).

[26] Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27 (2014).

[27] Dong, Q. *et al.* A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2024).

[28] Schulhoff, S. *et al.* The prompt report: A systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608* (2024).

[29] Zhou, D. *et al.* Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations (ICLR)* (2023).

[30] Zhou, Y. *et al.* Large language models are human-level prompt engineers. In *International Conference on Learning Representations* (2023).

[31] Yang, C. *et al.* Large language models as optimizers. In *International Conference on Learning Representations* (2024).

[32] Khattab, O. *et al.* DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations* (2024).

[33] Yuksekgonul, M. *et al.* TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496* (2024).

[34] Reimers, N. & Gurevych, I. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, 3982–3992 (2019).

[35] Karpukhin, V. *et al.* Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6769–6781 (2020).

[36] Asai, A., Wu, Z., Wang, Y., Sil, A. & Hajishirzi, H. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations (ICLR)* (2024).

[37] Jiang, Z. *et al.* Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 7969–7992 (2023).

[38] Sarthi, P. *et al.* RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)* (2024).

[39] Edge, D. *et al.* From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[40] Zeng, S. *et al.* The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG). In *Findings of the Association for Computational Linguistics: ACL 2024* (2024).

[41] Cuconasu, F. *et al.* The power of noise: Redefining retrieval for RAG systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 719–729 (2024).

[42] Wang, H. *et al.* Towards adapting open-source large language models for expert-level clinical note generation. *arXiv preprint arXiv:2405.00715* (2024).

[43] Wang, Y. *et al.* Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 13484–13508 (Association for Computational Linguistics, Toronto, Canada, 2023).

[44] Dettmers, T., Pagnoni, A., Holtzman, A. & Zettlemoyer, L. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, vol. 36 (Curran Associates, Inc., 2023).

[45] Ouyang, L. *et al.* Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, vol. 35, 27730–27744 (Curran Associates, Inc., 2022).

[46] Stiennon, N. *et al.* Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[47] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[48] Christiano, P. F. *et al.* Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, vol. 30 (Curran Associates, Inc., 2017).

[49] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[50] Shao, Z. *et al.* DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).

[51] Rafailov, R. *et al.* Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, vol. 36 (Curran Associates, Inc., 2023).

[52] Lu, Z. *et al.* Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790* (2024).

[53] Ovadia, O., Brief, M., Mishaeli, M. & Elisha, O. Fine-tuning or retrieval? comparing knowledge injection in LLMs. *arXiv preprint arXiv:2312.05934* (2023).

[54] Yao, S. *et al.* ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations* (2023).

[55] Chase, H. LangGraph: Build resilient language agents as graphs. `https://github.com/langchain-ai/langgraph` (2024).

[56] Wu, Q. *et al.* AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155* (2023).

[57] Du, Y., Li, S., Torralba, A., Tenenbaum, J. B. & Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)* (2024).

[58] Hong, S. *et al.* MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)* (2024).

[59] Zhang, J. *et al.* AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)* (2025).

[60] Hu, S., Lu, C. & Clune, J. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435* (2024).

[61] Schick, T. *et al.* Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, vol. 36 (Curran Associates, Inc., 2023).

[62] Zhang, T. *et al.* RAFT: Adapting language model to domain specific RAG. *arXiv preprint arXiv:2403.10131* (2024).

[63] Zelikman, E., Wu, Y., Mu, J. & Goodman, N. D. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, vol. 35, 15476–15488 (Curran Associates, Inc., 2022).

[64] Novikov, A. *et al.* AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131* (2025).

[65] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. 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 (Association for Computational Linguistics, Minneapolis, Minnesota, 2019).

[66] Raffel, C. *et al.* Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* **21**, 1–67 (2020).

[67] Yin, S. *et al.* A survey on multimodal large language models. *National Science Review* **11**, nwae403 (2024).

[68] Li, C. *et al.* Multimodal foundation models: From specialists to general-purpose assistants. *arXiv preprint arXiv:2309.10020* (2023).