Sciences du Nord Nature Science and Technology
Vol. 1, No. 02, December 2024: pp. 139-148
E-ISSN: 3046-6504
P-ISSN: 3047-4922
DOI: https://doi.org/10.71238/snnst.v1i02.237

From Pipelines to Agent Fabrics: A Reference Architecture for Small-Model, Reasoning-Augmented, Organization-Scale Agentic Data Platforms

Jagan Ankam [1]
[1] CGI Inc, Miami, United States
Corresponding Author: jagan.ankam@gmail.com

ARTICLE HISTORY
Received Nov, 2024 **Revised Nov, 2024** **Accepted Dec, 2024**

ABSTRACT

Two shifts are reshaping how enterprises consume data. First, the language-model layer is fragmenting: compact, domain-tuned small language models (SLMs) now handle a large share of routine work, while a separate class of reasoning models spends additional inference-time computation to work through mathematical and logical problems before answering. Second, AI agents are moving from individual assistants to team-level collaborators and, increasingly, to organization-wide fleets that act across departmental boundaries. Most data platforms were designed for human analysts and batch BI, not for thousands of autonomous, heterogeneous model consumers that read, reason and write. This paper argues that these shifts turn the data platform into an agent fabric and proposes a five-layer reference architecture: governed data products, a semantic and context layer, a cost-aware model tier that routes between SLMs and reasoning models, an agent orchestration and interoperability layer, and a cross-cutting observability, lineage and policy plane. We formalize SLM-first routing with verifier-gated escalation as an expected-cost model, derive the break-even condition under which it beats a single large model, and introduce three data-engineering constructs that agents require: context contracts, action lineage and tiered agent memory. A maturity model maps the personal-to-team-to-organization progression onto concrete platform capabilities, and an illustrative supply-chain scenario shows how the layers interact. We close with open problems in evaluation, governance and multi-agent failure.

Keywords: *AI Governance; Data Architecture; Data Lineage; Data Products; Model Routing; Multi-Agent Systems; Reasoning Models; Small Language Models*

INTRODUCTION

For two decades the dominant consumer of an enterprise data platform has been a human: an analyst writing SQL, a dashboard refreshing overnight, a data scientist pulling a training set. Architectures such as the data warehouse, the data lake and their convergence in the lakehouse [1] were optimized for that consumer. Data mesh [2] reorganized ownership around business domains but still assumed that the end user of a data product is a person or a deterministic application.

That assumption is eroding along two axes. The first concerns *which models* consume data. Early scaling work [3] encouraged ever-larger general-purpose models, but compute-optimal training [4] and a wave of capable compact models [5], [6] showed that much smaller networks, trained on curated data and fine-tuned for a domain, deliver adequate quality on narrow tasks at a fraction of the cost and latency. Belcak et al. go further, arguing that SLMs are the natural default for agentic workloads, which are dominated by repetitive, well-scoped sub-tasks [7]. In parallel, *reasoning models* such as OpenAI o1 [8] and DeepSeek-R1 [9] internalize chain-of-thought [10] and trade additional test-time computation [11] for markedly better performance on mathematics, logic

and multi-step planning. The model layer is therefore no longer one model but a portfolio with very different cost and capability profiles.

The second axis concerns *who acts*. Tool-using agents built on ReAct-style loops [12], [13] began as personal assistants. Multi-agent frameworks [14], [15] and interoperability standards, notably the Model Context Protocol (MCP) for tool and data access [16] and the Agent2Agent (A2A) protocol for inter-agent communication [17], are enabling agents that work inside teams and, more recently, fleets that operate across an organization. An organizational agent does not merely read a report; it reconciles ledgers, files purchase orders and negotiates with other agents. Each of these is a read-reason-write cycle against enterprise data.

Together, these trends change the platform's job. Consumers become numerous, non-deterministic and write-capable; the cost of answering a question depends on which model answers it; and every automated action needs the same accountability once reserved for human decisions. We call a platform that meets these demands an **agent fabric**. This paper makes four contributions:

(1) A structured analysis of how the SLM shift, the rise of reasoning models and the personal-to-organizational agent progression each place distinct pressures on data architecture (see *Three Shifts and Their Architectural Pressures*).
(2) A five-layer reference architecture for agentic data platforms with explicit design principles (see *A Reference Architecture for the Agent Fabric*).
(3) An expected-cost model for SLM-first, verifier-gated routing to reasoning models, including a break-even condition and a routing algorithm (see *Cost-Aware Routing Between Small and Reasoning Models*).
(4) Three data-engineering constructs (context contracts, action lineage and tiered agent memory) plus a maturity model that ties agent scope to platform capability (see *Data Engineering Constructs for Organizational Agents*), illustrated through a cross-domain scenario.

BACKGROUND AND RELATED WORK

2.1 Lakehouse and Data Mesh

The lakehouse combines open file and table formats on object storage with warehouse-style transactions, governance and performance [1]. Data mesh complements this with four principles: domain ownership, data as a product, a self-serve platform and federated computational governance [2]. Both remain foundational in our architecture, but neither specifies how unstructured context, model selection or autonomous write-back should be handled. Sculley et al.'s analysis of hidden technical debt in ML systems [18] anticipates many of the integration risks that agentic consumers amplify.

2.2 Small Language Models

Hoffmann et al. showed that many large models were under-trained relative to their parameter count [4], shifting attention to data quality and training efficiency. Models such as Phi-3 [5] and Gemma 2 [6] demonstrated competitive benchmark performance at a few billion parameters. Techniques including knowledge distillation [19] and parameter-efficient fine-tuning such as LoRA [20] make it practical to maintain a pool of domain-specialized SLMs. For data teams, the important property is operational: SLMs can run close to the data, on modest hardware, with predictable latency and lower data-egress exposure.

2.3 Reasoning Models and Test-Time Compute

Chain-of-thought prompting [10] and self-consistency [21] showed that eliciting intermediate steps improves multi-step accuracy. Process supervision [22] and reinforcement learning on verifiable rewards [9] produced models that reason by default. Snell et al. found that allocating test-time compute adaptively can be more effective than scaling parameters [11]. The architectural consequence is that *inference cost becomes elastic*: the same model may spend very different amounts of computation depending on the requested reasoning budget.

2.4 Agents, Multi-Agent Systems and Interoperability

ReAct [12] interleaves reasoning with tool calls; Toolformer [13] learns when to call tools. AutoGen [14] and MetaGPT [15] coordinate multiple role-specialized agents, and generative-agent work [23] highlights the role of long-term memory, a theme MemGPT treats as an operating-system-style memory hierarchy [24]. MCP [16] standardizes how agents discover and call tools and data sources; A2A [17] standardizes agent-to-agent task exchange. Empirical analyses of multi-agent failures identify specification, inter-agent misalignment and verification gaps as dominant causes [25], and indirect prompt injection shows that any data an agent reads can carry adversarial instructions [26].

2.5 Model Routing and Cascades

FrugalGPT [27] demonstrated that cascading cheaper models before expensive ones, with a scorer deciding when to stop, can preserve quality while reducing cost substantially. RouteLLM [28] learns routers from preference data. Our routing model, presented later in this paper, builds on this line, but it treats the *reasoning budget* as a second decision variable and places the router inside the data platform, where it can use data-product metadata and lineage as routing features.

THREE SHIFTS AND THEIR ARCHITECTURAL PRESSURES

Table 1 summarizes how each trend translates into a concrete pressure on the data platform and the design response we propose. We discuss each briefly.

Table 1. Trends, resulting pressures on data architecture, and proposed design responses.

Trend	Pressure on the data platform	Design response
Large → small models	Many specialized models need curated, domain-scoped training and evaluation data; inference moves closer to data.	Data products publish fine-tuning and eval sets as first-class outputs; model registry linked to data lineage.
Rise of reasoning models	Elastic, expensive inference; long reasoning traces become auditable artifacts.	Cost-aware router with reasoning budgets; trace store governed like any other dataset.
Personal → team agents	Shared context and memory; concurrent writes to shared systems.	Semantic/context layer; team-scoped memory with access control; idempotent write APIs.
Team → organizational agents	Cross-domain actions, agent identity, accountability at scale.	Agent identity and least privilege; action lineage; policy-as-code; A2A-mediated hand-offs.

3.1 From One Large Model to a Portfolio

When a single frontier model served all requests, the data platform's concern ended at retrieval. A portfolio of SLMs changes this in three ways. Each SLM needs a reproducible training and evaluation dataset tied to the domain that owns it, which makes model artifacts a natural extension of the data-product concept. Smaller models are more sensitive to context quality, so retrieval precision and schema clarity matter more than they did for large models that could compensate for noisy input. Finally, because SLMs can be deployed inside the domain's own security boundary, they reduce the volume of sensitive data that must leave governed storage.

3.2 Reasoning as an Elastic, Auditable Resource

Reasoning models produce extended intermediate traces before answering. Two consequences follow for data engineers. Cost is no longer a function of the question alone but of the reasoning budget granted, which calls for budget allocation as a platform service rather than an application-level afterthought. And where a reasoning model's output drives a business action, its trace is evidence: it should be retained, versioned, access-controlled and linked to the data it consulted, subject to the same retention policies as other records. Traces may also contain sensitive intermediate values, so redaction belongs in the pipeline.

3.3 From Personal to Organizational Agents

A personal agent operates with one user's permissions over that user's data. A team agent shares memory and context with several people and may act on shared systems, introducing concurrency and the need for idempotent writes. An organizational agent fleet acts across domain boundaries, delegating tasks to agents owned by other teams. At this stage, questions that data governance already answers for humans (who may see what, who changed this record and why) must be answered for non-human actors at machine speed. The progression is cumulative: each stage inherits the requirements of the previous one.

A REFERENCE ARCHITECTURE FOR THE AGENT FABRIC

Figure 1 presents the proposed architecture. It is layered so that each layer can evolve independently, and it reuses established data-platform patterns wherever possible rather than introducing parallel infrastructure for AI.

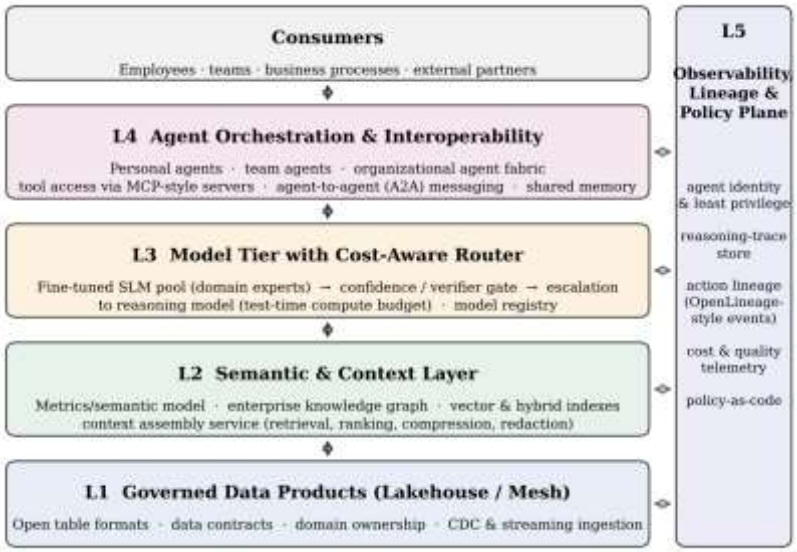

Figure 1. Five-layer reference architecture for an agentic data platform. L1–L4 form the request path; L5 spans all layers.

L1: Governed data products. Domain-owned data products on open table formats, with explicit data contracts covering schema, freshness, quality thresholds and semantics. Change data capture and streaming ingestion keep products current enough for agents that act in near real time. New output ports expose curated fine-tuning and evaluation sets alongside tables, so SLMs inherit the product's ownership and quality guarantees.

L2: Semantic and context layer. A semantic model of metrics and entities, an enterprise knowledge graph, and vector or hybrid indexes over unstructured content [29]. A *context assembly service* retrieves, ranks, compresses and redacts content per request according to the caller's identity and the target model's context window. This service is where the context contracts described later are enforced.

L3: Model tier with cost-aware router. A registry-backed pool of fine-tuned SLMs, one or more reasoning models, and lightweight verifiers. The router sends each request to the cheapest model expected to meet the quality target and escalates when a verifier rejects the answer or confidence falls below a threshold.

L4: Agent orchestration and interoperability. Runtime for personal, team and organizational agents. Tools and data are exposed through MCP-style servers [16] that wrap data products and operational systems behind typed, permissioned interfaces; inter-agent delegation uses A2A-style messaging [17]. Shared memory follows the tiered memory design described later.

L5: Observability, lineage and policy plane. Cross-cutting services for agent identity and least-privilege authorization, a reasoning-trace store, action lineage emitted as open lineage events [30], cost and quality telemetry, and policy-as-code evaluated at tool-call time.

4.1 Design Principles

Five principles guide the architecture:

- **Agents are first-class principals.** Every agent has an identity, an owner, scoped credentials and an audit trail; no agent borrows a human's session.
- **Smallest sufficient model.** Default to SLMs; reserve reasoning models for requests that demonstrably need them, and make that decision observable.
- **Context is a product.** Context delivered to models is versioned, contracted and quality-monitored, just as tables are.
- **Every action has lineage.** Writes performed by agents are linked to the inputs, model versions, reasoning traces and policies that produced them.
- **Treat retrieved content as untrusted.** Data read by agents may contain injected instructions [26]; tool permissions, not model obedience, are the security boundary.

COST-AWARE ROUTING BETWEEN SMALL AND REASONING MODELS

5.1 Expected-Cost Model

Consider a request class k with a quality target. Let c_s be the mean cost of answering with an SLM, c_v the cost of verifying that answer, and $c_r(b)$ the cost of a reasoning model with reasoning

budget b (in thinking tokens). Let α be the fraction of SLM answers accepted by the verifier. Under an SLM-first policy with verifier-gated escalation, the expected cost per request is

$$E[C_{route}] = c_s + c_v + (1 - \alpha) \cdot c_r(b) \quad (1)$$

whereas sending every request directly to the reasoning model costs $c_r(b)$. Routing is cheaper whenever

$$c_s + c_v < \alpha \cdot c_r(b), \quad \text{i.e.} \quad \alpha > \alpha^\star = (c_s + c_v) / c_r(b) \quad (2)$$

The break-even acceptance rate $\alpha^\star$ is small whenever SLM and verification costs are small relative to the reasoning model, which is typically the case. Quality must be considered alongside cost. If the verifier has false-acceptance rate ε (accepting an incorrect SLM answer), the residual error introduced by routing is approximately $\varepsilon \cdot \alpha$. The verifier therefore bounds quality, and its calibration on each request class is a data-engineering task: it needs labeled evaluation sets drawn from the owning data product (L1).

Because c_r depends on b, the router also chooses a budget. Following the observation that optimal test-time compute varies with difficulty [11], we assign budgets by request class and adjust them using telemetry from L5, increasing b only where the marginal accuracy gain justifies the cost.

5.2 Worked Example

Table 2 applies Eq. (1) to three hypothetical request classes. Costs are expressed in normalized units relative to a single SLM call; they are illustrative parameters chosen to show the shape of the trade-off, not measurements.

Table 2. Illustrative routing economics (normalized cost units; hypothetical parameters).

Request class	c_s	c_v	$c_r(b)$	α	$E[C_{route}]$	Saving vs. direct
Metric lookup / SQL	1	0.5	30	0.92	3.9	87%
Contract clause extraction	1	1	40	0.75	12.0	70%
Multi-constraint planning	1	2	60	0.20	51.0	15%

For routine, well-structured classes, SLM-first routing yields large savings. For hard planning tasks, where the SLM rarely passes verification, the saving is marginal and the added latency of a failed first attempt may not be worth it; Eq. (2) tells the platform to route such classes straight to the reasoning model. The decision is thus made per class, from data, and revisited as SLMs improve.

5.3 Routing Algorithm

Algorithm 1 sketches the router. Request classification uses lightweight features available in the platform: the target data products, their sensitivity labels, the presence of numeric or multi-hop constraints, and the agent's historical escalation rate.

Algorithm 1. SLM-first routing with verifier-gated escalation

```
input: request q, caller identity u, policy P
k  <- classify(q, metadata(q.targets))
if not P.permits(u, q.targets): return DENY
ctx <- assemble_context(q, u, contract(k))      # L2
if alpha_hat[k] <= (c_s[k] + c_v[k]) / c_r(b[k]):
    return reasoning(q, ctx, budget=b[k])         # skip SLM
a  <- slm[k](q, ctx)
if verify[k](q, ctx, a) >= tau[k]:
    log(k, model=slm[k], accepted=true); return a
r  <- reasoning(q, ctx ++ [a], budget=b[k])      # draft as hint
log(k, model=reasoning, accepted=false, trace=r.trace)
update(alpha_hat[k], b[k])                       # from L5 telemetry
return r.answer
```

Two details matter in practice. Passing the rejected SLM draft to the reasoning model as a hint can reduce reasoning tokens, though it may also anchor the model; this should be validated per class. And all routing decisions are logged to L5, which makes the platform's model usage auditable and lets $\hat{\alpha}$ and b be re-estimated continuously.

DATA ENGINEERING CONSTRUCTS FOR ORGANIZATIONAL AGENTS

6.1 Context Contracts

A data contract specifies what a table promises. A **context contract** specifies what a model will receive for a given request class: the permitted sources, maximum token size, required freshness, redaction rules, mandatory provenance metadata, and a minimum retrieval-quality threshold measured on an evaluation set. Context contracts make prompt inputs testable in continuous integration, so a schema change in an upstream product that would silently degrade an agent's context fails a check before deployment. SLMs benefit disproportionately because they tolerate noisy context less well than large models.

6.2 Action Lineage

Conventional lineage records which jobs read and wrote which datasets. **Action lineage** extends this to agent behaviour: each write performed by an agent emits an event linking the action to the agent identity, the delegating principal (human or agent), the model and version, the reasoning-trace identifier, the context bundle, the policy decision, and the downstream records changed. Representing these as open lineage events [30] lets existing catalog and impact-analysis tooling answer questions such as *which purchase orders were influenced by model version X* or *which agent decisions relied on a data product later found to be stale*. This is the mechanism by which an organization can roll back, explain or audit automated decisions.

6.3 Tiered Agent Memory

Agents need memory, but unmanaged memory becomes an ungoverned shadow data store. We propose three tiers aligned with agent scope: *working memory* scoped to a single task and discarded at completion; *team memory* shared among a team's agents and members, stored as a governed data product with access control and retention; and *organizational memory*, consisting of curated, validated facts promoted from team memory through a review step and published to the knowledge graph (L2). Promotion between tiers is a data-quality pipeline, not an implicit side effect, which limits the spread of hallucinated or injected content across the organization.

6.4 A Maturity Model

Table 3 connects the personal-team-organizational progression to platform capabilities. Organizations can use it to sequence investment: attempting stage-3 agents on a stage-1 platform is a common source of the specification and verification failures documented in [25].

Table 3. Maturity model linking agent scope to required platform capabilities.

Stage	Agent scope	Minimum platform capabilities	Typical risks if missing
1 Personal	Single user, read-mostly, user's own permissions	Catalog search, RAG over documents, basic logging	Data leakage via over-broad retrieval
2 Team	Shared context, writes to team systems, several agents	Semantic layer, team memory, agent identities, idempotent tools, routing	Conflicting writes, unreviewed shared memory
3 Organization	Cross-domain delegation, autonomous workflows	Action lineage, policy-as-code, A2A hand-offs, trace store, cost governance	Unattributable actions, cascading multi-agent failures, cost overruns

ILLUSTRATIVE SCENARIO: CROSS-DOMAIN SUPPLY-CHAIN EXCEPTIONS

To show how the layers interact, consider a manufacturer whose supplier signals a delayed shipment. The scenario is illustrative; it describes intended behaviour rather than reporting a deployment.

(1) A procurement team agent receives the supplier's delay event via streaming ingestion (L1). An SLM fine-tuned on procurement data classifies the event and extracts affected part numbers; the verifier accepts the result (L3).

(2) The agent queries the inventory and production-plan data products through MCP-style tools (L4). The context assembly service applies the procurement context contract, redacting supplier pricing not needed for the task (L2).

(3) Determining whether to expedite, substitute or reschedule is a multi-constraint planning problem. Its request class has a low historical α, so the router sends it directly to a reasoning model with a moderate budget, as described in the routing model.

(4) The recommended substitution requires a budget variance approval, which belongs to finance. The procurement agent delegates via A2A to a finance agent, which evaluates the request against policy-as-code and, because the amount exceeds an autonomy threshold, routes it to a human approver (L4, L5).

(5) On approval, the procurement agent issues a revised purchase order. An action-lineage event links the order to both agents, the reasoning trace, the context bundle, the policy decision and the approving human (L5). The validated finding that the substitute part is compatible is promoted to organizational memory after review.

If the inventory data product is later found to have been stale, action lineage identifies every affected decision, and the stage-3 capabilities in Table 3 make remediation tractable rather than forensic.

DISCUSSION AND OPEN CHALLENGES

Evaluation. Benchmarks for individual models do not measure end-to-end agent-fabric quality. Platforms need task-level evaluation suites, owned by domains, that capture correctness of actions, not just answers. Building and maintaining these datasets is squarely a data-engineering responsibility.

Verifier reliability. The routing model depends on calibrated verifiers. Verifiers can drift as data and SLMs change, and an overconfident verifier silently lowers quality. Continuous recalibration from human-reviewed samples is required.

Trace governance. Reasoning traces are valuable evidence but can be voluminous and may contain sensitive intermediate data. Retention, summarization and access policies for traces remain largely unaddressed in practice.

Security. Indirect prompt injection [26] means that any data product can become an attack vector once agents read it with write privileges elsewhere. Least-privilege tool design and policy enforcement outside the model are necessary but not sufficient; provenance-aware context assembly is a promising direction.

Organizational design. Agent fleets blur domain boundaries. Federated governance [2] must decide who owns an agent, who answers for its actions across domains, and how autonomy thresholds are set and reviewed.

Limitations. This paper proposes an architecture and an analytical cost model; it does not report an empirical deployment. The parameters in Table 2 are illustrative. Validating the break-even analysis and the maturity model across real workloads is the primary direction for future work.

CONCLUSION

The move from large to small language models, the emergence of reasoning models with elastic inference cost, and the progression of agents from personal assistants to organization-wide fleets together change what a data platform is for. It must now serve many heterogeneous, autonomous, write-capable consumers while keeping cost, quality and accountability under control. We proposed the agent fabric, a five-layer reference architecture that extends lakehouse and mesh principles with a context layer, a cost-aware model tier and a cross-cutting lineage and policy plane. We formalized SLM-first, verifier-gated routing and its break-even condition, and introduced context contracts, action lineage and tiered agent memory as concrete constructs. Our central claim is that the hard problems of enterprise agents are, to a large degree, data-architecture problems, and that data engineers are well placed to solve them.

REFERENCES

[1] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. Conf. Innovative Data Systems Research (CIDR)*, 2021.

[2] Z. Dehghani, *Data Mesh: Delivering Data-Driven Value at Scale*. Sebastopol, CA, USA: O'Reilly Media, 2022.

[3] J. Kaplan, "Scaling laws for neural language models," 2020. doi: 10.48550/arXiv.2001.08361.

[4] J. Hoffmann, "Training compute-optimal large language models," in *Advances in Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2203.15556.

[5] M. Abdin, "Phi-3 technical report: A highly capable language model locally on your phone," 2024. doi: 10.48550/arXiv.2404.14219.

[6] Gemma Team, "Gemma 2: Improving open language models at a practical size," 2024. doi: 10.48550/arXiv.2408.00118.

[7] P. Belcak, "Small language models are the future of agentic AI," 2025. doi: 10.48550/arXiv.2506.02153.

[8] OpenAI, "OpenAI o1 system card," 2024. doi: 10.48550/arXiv.2412.16720.

[9] D. Guo, "DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning," *Nature*, vol. 645, pp. 633–638, 2025, doi: 10.1038/s41586-025-09422-z.

[10] J. Wei, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2201.11903.

[11] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM test-time compute optimally can be more effective than scaling model parameters," 2024. doi: 10.48550/arXiv.2408.03314.

[12] S. Yao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023. doi: 10.48550/arXiv.2210.03629.

[13] T. Schick, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, 2023. doi: 10.48550/arXiv.2302.04761.

[14] Q. Wu, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," 2023. doi: 10.48550/arXiv.2308.08155.

[15] S. Hong, "MetaGPT: Meta programming for a multi-agent collaborative framework," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2024. doi: 10.48550/arXiv.2308.00352.

[16] Anthropic, "Introducing the Model Context Protocol."

[17] Google, "Announcing the Agent2Agent Protocol (A2A)," Google Developers Blog.

[18] D. Sculley, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems*, 2015.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. doi: 10.48550/arXiv.1503.02531.

[20] E. J. Hu, "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2022. doi: 10.48550/arXiv.2106.09685.

[21] X. Wang, "Self-consistency improves chain of thought reasoning in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023. doi: 10.48550/arXiv.2203.11171.

[22] H. Lightman, "Let's verify step by step," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2024. doi: 10.48550/arXiv.2305.20050.

[23] J. S. Park, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th ACM Symp. User Interface Software and Technology (UIST)*, 2023. doi: 10.1145/3586183.3606763.

[24] C. Packer, "MemGPT: Towards LLMs as operating systems," 2023. doi: 10.48550/arXiv.2310.08560.

[25] M. Cemri, "Why do multi-agent LLM systems fail?," 2025. doi: 10.48550/arXiv.2503.13657.

[26] K. Greshake, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. doi: 10.1145/3605764.3623985.

[27] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," 2023. doi: 10.48550/arXiv.2305.05176.

[28] I. Ong, "RouteLLM: Learning to route LLMs with preference data," 2024. doi: 10.48550/arXiv.2406.18665.

[29] P. Lewis, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, 2020. doi: 10.48550/arXiv.2005.11401.

[30] OpenLineage, "An open framework for data lineage collection and analysis."