

Original Article

AI-Based Intent Normalization and Service Routing in Distributed Agentic Microservices

Ashok Kumar¹, Dhruv Kumar Seth², Karan Kumar Ratra³, Vikas Kumar Mittal⁴

^{1,2,3,4}Walmart Global Tech, Sunnyvale, CA, USA.

¹Corresponding Author : ashok.mac3@gmail.com

Received: 19 July 2026

Revised: 28 August 2026

Accepted: 11 September 2026

Published: 28 September 2026

Abstract - LLMs are increasingly being embedded into a variety of distributed agentic systems, in the form of agents with APIs, microservices, workflows, and external tools for task completion. Systems need to be able to handle requests that are worded differently but mean the same thing, to make sure routing is consistent, caching can be reused, latency is reduced, reliability is ensured, and governance is supported. Conventional intent representations, beyond simply conveying what users want with their intent and respective slots, do not encode information about user authorization, policies with implications for execution, side effects of actions, locality of execution, and operational implications of intents. Prior work has provided frameworks for normalization, capability discovery and routing in distributed agentic systems, but all this work is done independently and thus does not provide a unified pipeline for handling requests in distributed systems. The main contribution of this paper is to propose a unified framework describing the normalization, discovery, routing, execution control and runtime feedback in distributed agentic systems. The core of this work is the introduction of a canonical representation of intent, called an intent record, which can convey all kinds of semantic information, as well as context, policy, and operational information about the system as a whole, including its uncertainty, to be used by the system's routing layer. We also introduce a new constraint-first multi-criteria routing process. A reference architecture is also given for such sorts of systems. The architecture's main functional components are semantic control and its decoupled, cross-governance-deterministic execution. An evaluation framework for the proposed approach in the future is also presented. This framework is not validated through implementation or benchmarking experiments. Such an empirical evaluation is left for future work.

Keywords - Intent Normalization, Service Routing, Agentic Microservices, Canonical Intent Representation, Distributed Systems, Multi-Agent Orchestration.

1. Introduction

LLM-based agentic systems increasingly act as orchestrators of heterogeneous capabilities rather than as monolithic chat interfaces. The same user request can be served by an LLM-based agent, a predefined workflow, a deterministic microservice, an external API or a human approval step, and the decision is no longer about language understanding. Agentic deployments span multiple tools, registries, service meshes and data stores, so how a request is interpreted before it is routed becomes a systems concern, not a linguistic one.

A major issue is that semantically equivalent queries can often be expressed in many different ways. Without intermediate normalization, two paraphrases of the same underlying request may be interpreted differently by an LLM and routed to different tools, services, or agents. Such instability undermines routing consistency, reduces the effectiveness of semantic caching [1], increases latency via repeated reasoning and duplicated execution paths, and complicates governance, as the same operation may be logged, authorized, or audited inconsistently depending on how it is phrased.

Prior work on intent-classification and slot-filling for task-oriented dialog systems in general does not capture the key aspects of distributed execution environments that a routing system needs to commit to a route. On one hand, while semantic relevance of a request to a tool description is required to select from among candidate tools in production, semantic appropriateness of a tool does not imply that the same tool is the right choice for a given user who may be unauthorized, or a tool that does not reside on a required local area, or a tool with failing dependencies, or even a highly reversible action for which there is not yet sufficient evidence to merit execution. Furthermore, aspects such as authorization, policy, side-effects, data locality, cost, reliability, and infrastructure health are not captured in intent-classification and slot-filling.

We are interested in two stages of processing: after intent recognition and with all of the relevant information visible to the routing stage (instead of as preconditions to invocation of a service by a semi-autonomous function) the following need to be integrated from the vast array of work on distributed systems, human-computer interaction and in task completion by complex systems: intent classification



and slot filling, semantic parsing, query equivalence, tool learning, service discovery, QoS-aware routing, semantic caching, and agent orchestration. This paper provides the first unified conceptual foundation for integrating all of this into a single, distributed framework for intent normalization and routing between distributed, agentic microservices.

This is a conceptual paper, and it builds on representative work in the field. The paper proposes a taxonomy of components of an intent normalization and distributed routing system, introduces a canonical form for an intent record, outlines a constraint-based multi-criterion routing framework, and introduces a reference architecture for a semantic controller/deterministic execution engine system.

The work provides five major conceptual contributions. First, it provides a structured taxonomy that systematically characterizes the main stages of intent normalization, capability discovery, route evaluation, execution control, and runtime feedback. Second, it defines a canonical representation of user intents. This intent representation is semantic, context-dependent, policy-aware, operational-aware, and uncertainty-aware. Third, it provides a general constraint-driven multi-criteria routing approach that is applicable to a variety of degrees of programmatic control, such as agents, tools, APIs, workflows, and even fully deterministic microservices. Fourth, the work outlines a general reference architecture for such semantic-based distributed execution environments that segregates control from execution. It identifies a number of key cross-cutting concerns and provides examples for how they can be governed. Fifth, it presents a concrete case study and a future evaluation framework for this general class of systems, along with several research questions concerning routing consistency, safety, efficiency, and recoverability.

2. Background and Related Work

This section positions the proposed framework relative to five bodies of related work: intent understanding, agent orchestration, tool-use benchmarking, service discovery and QoS-aware routing, and observability.

2.1. Intent Understanding: Classification, Parsing, and Canonicalization

Joint intent classification and slot filling, for example, BERT-based joint models, propose that intent detection and entity extraction should be solved jointly instead of independently, improving generalization for natural-language understanding in task-oriented systems [2]. This line of work directly motivates treating normalization as a structured, unified process rather than a sequence of independent classifiers, but it yields a fixed label-and-slot output rather than the richer operational representation that distributed routing requires, which the canonical intent record proposed in Section IV is intended to supply. Work on semantic parsing complementarily frames the same underlying problem of mapping paraphrased natural-language requests onto a shared canonical or logical form,

directly motivating the equivalence-preserving role assigned to normalization in the framework of this paper [3].

2.2. Agent Orchestration and Tool-Use Frameworks

MRKL systems are a modular, neuro-symbolic architecture where a controller routes between large language models and discrete external modules, making routing an architectural, not purely linguistic concern [4]. ReAct interleaves reasoning traces with actions so a model can plan, act, and revise its plan based on observations, which underpins the interpretation and exception-handling role assigned to agents in Section V [5]. Toolformer shows that a model can learn when and how to call external tools in a self-supervised way [6].

HuggingGPT shows that a controller model can break down a request and assign subtasks to specialized models [7]. AutoGen generalizes this pattern into a multi-agent conversation framework where agents with different roles and tools coordinate to solve a task [8]. Collectively, these works establish that modern agentic systems are orchestration layers over heterogeneous capabilities, which is the premise motivating a dedicated normalization stage prior to routing.

2.3. Tool and Capability Discovery Benchmarks

API-Bank evaluates tool-augmented LLMs for their ability to plan, retrieve, and call APIs in a runnable evaluation system [9]. ToolLLM builds on this line of work by providing a large-scale instruction-tuning dataset over more than sixteen thousand real-world APIs, and a depth-first search-based decision-tree algorithm for multi-step tool-use planning [10]. Gorilla couples a fine-tuned model with document retrieval, so that API calls remain accurate as documentation changes, directly demonstrating the stale-metadata problem discussed in Section V-D [11].

The Berkeley Function-Calling Leaderboard (BFCL) is a live leaderboard that measures function-calling accuracy across models and tasks [12]. StableToolBench aims at addressing the instability of live-API benchmarking and provides a stable evaluation setting for tool learning [13]. More recent benchmarks that use the Model Context Protocol (MCP) evaluate tool use at a larger scale: LiveMCPBench evaluates agents across dozens of live MCP servers and hundreds of tools [14]; MCP-Bench connects models to real MCP servers across finance, travel, and scientific-computing domains to test cross-tool coordination [15]; MCP-RADAR introduces a multi-dimensional benchmark that scores both answer correctness and operational accuracy [16]; and MCP-Atlas evaluates tool-use competency against production MCP servers using claims-based, rather than holistic, scoring [17].

Together, these benchmarks show that realistic tool-use evaluation now extends well beyond isolated function call accuracy to include multi-step coordination, discovery under distractors, and operational efficiency; the future evaluation framework in Section VII builds on this.

2.4. Service Discovery, QoS-Aware Routing, and Infrastructure Standards

Outside the LLM literature, service discovery and QoS-aware routing have long combined functional compatibility with operational signals such as latency, cost and reliability when choosing among candidate services. The Model Context Protocol (MCP) standardizes the way AI applications discover and invoke tools, prompts, and resources exposed by external servers [18], with its specification defining the schema and lifecycle that a capability registry can rely upon [19]; its authorization guidance treats access control and delegated authorization as transport-level concerns that a policy filter must still evaluate before a route is committed [20]. In addition to MCP's agent-to-tool interoperability, Google's Agent2Agent (A2A) protocol standardizes agent-to-agent discovery, authentication, and task delegation across frameworks and organizational boundaries, implying that the routing layer envisioned in this paper must ultimately interoperate with both tool-facing and agent-facing infrastructure standards [21]. The Kubernetes Gateway API offers a standardized routing layer for ingress and service-mesh traffic, with route semantics for east-west communication between services [22]. These standards imply that the control plane for agentic routing increasingly needs to interoperate with general infrastructure routing mechanisms, rather than being hidden inside prompts. This paper treats interoperability as a design constraint that the reference architecture in Section V is meant to satisfy, not as an integration this paper has built or tested against live MCP, A2A, or Gateway API deployments; validating actual compatibility with these standards is left to the future prototype work described in Section VII.

2.5. Observability and Trace Propagation

OpenTelemetry offers a vendor-neutral framework for gathering traces, metrics, and logs from distributed services [23]. The W3C Trace Context recommendation standardizes the propagation of trace identifiers across services via the traceparent header [24]. The corresponding

standards allow for the normalized intent identifier to be propagated through services and related to the process of candidate selection, service routing and end service execution. The intent is compared to the literature in the field, and novel aspects are defined. The novel aspects are not within individual components, such as, for example, intent classification and semantic parsing, which end at the stage of label and slot output. Other areas, such as tool use and service discovery, assess the quality of candidate selection but do not include information regarding authorization, policy and effects within candidate selection decisions. Lastly, standards for QoS-aware routing and for the associated cloud and enterprise infrastructure are in place to support the operational selection process, provided that the incoming request is well-formed in the first place. None of the reviewed approaches integrates equivalence-preserving normalization with hard eligibility checking and, most importantly, with operation-aware candidate comparison. This gap is closed by the canonical intent record and by the constraint-first routing process.

3. Structured Literature Synthesis and Taxonomy

In Section II, we synthesize representative work in the form of peer-reviewed studies, preprints, and technical documentation for natural-language understanding, agent orchestration, tool and capability discovery, service routing, and observability. This is a structured integrative review rather than a systematic literature review (cf. Section III). We do not describe a search with correspondent reproducible search strings, databases, screening numbers and quality assessment for the studies that we review in Section II. The taxonomy in Table I has been derived from clustering the reviewed literature along the dimensions of request normalization, the types of capabilities to be routed to, constraints on eligible requests, distinguishing signals for eligible requests, where the routing decision is made, and how execution is controlled after a decision has been made.

Table 1. Taxonomy of Intent Normalization and Routing

Dimension	Categories	Routing significance
Normalization method	Intent-slot; semantic parsing; schema grounding; query equivalence; context-aware normalization	Shapes the structure and stability of the canonical representation
Routing target	Microservice; API; tool; workflow; agent; human approval	Defines the execution class selected
Eligibility constraints	Authorization; residency; modality; schema; policy; side effects	Removes invalid candidates before comparison
Comparative signals	Semantic fit; reliability; latency; cost; locality; service health	Differentiates eligible candidates
Architectural placement	Gateway; agent runtime; orchestration service; service mesh	Determines where the decision is made
Runtime control	Verification; fallback; approval; refusal; telemetry	Governs risk and recovery

Table 1 reviews the critical route dimensions that cannot individually determine a route: normalization methods to determine which information to forward to the routing layer; eligibility constraints and comparative signals

to select the candidate of choice; architectural placement where the above decisions are taken; and runtime control how the system evolves after having made a choice and committed to a route. Sections IV and V elaborate on these

points, the first providing a canonical representation, the second outlining a constraint-first routing approach.

4. Canonical Intent Representation

The central conceptual abstraction proposed in this paper is the canonical intent record, a structured representation produced by the intent normalizer before any routing decision is made. It is denoted:

$$I_c = \langle o, e, c, x, p, s, q, u \rangle \quad (1)$$

where o is the normalized operation; e is the entities, objects and parameters involved; c is business and

execution constraints; x is conversational, user, tenant and runtime context; p is policy, authorization, privacy and data-residency requirements; s is the side-effect, reversibility and consequence classification of the requested operation; q is quality-of-service expectations such as latency, reliability and cost; and u is uncertainty and confidence information associated with the normalization itself. This notation is a conceptual representation whose aim is to make the architectural idea precise. It is not a standardized schema, nor a trained model, nor an empirically validated mathematical formulation. The purpose and routing effect of each field are shown in Table 2.

Table 2. Canonical Intent Record Fields

Field	Purpose/example	Routing effect
o	Normalized operation, e.g. cancel_order	Determines the candidate's capability class
e	Entities and parameters, e.g. order identifier, customer identifier	Bounds schema and input compatibility
c	Business and execution constraints, e.g. exclude the linked subscription	Filters ineligible candidates
x	Conversational, tenant, and runtime context	Disambiguates otherwise equivalent requests
p	Authorization, privacy, and residency requirements	Enforces hard eligibility filtering
s	Side-effect and reversibility classification	Sets the confidence-consequence policy
q	Latency, reliability, and cost expectations	Differentiates eligible candidates
u	Ambiguity and normalization confidence	Triggers verification or fallback

Note that we conceive of constructing the full record from the raw request as more of a multi-stage process than a single-stage classification. An intent normalizer for the operation and for entities within it will serve as the basis for a joint intent-slot model. The conversational and tenant context can then be merged to deal with references to previous requests (e.g. "my most recent order"). Authorization and policy metadata for the caller's session and for tenant-wide configuration can then be attached. A side-effect and reversibility classifier can then label the resulting operation against a taxonomy of known operations (governance-maintained). Moreover, finally, an uncertainty estimator can record where alternative parses and referring expressions are still ambiguous. Note that every field of the record is then populated from a different information source: i.e. the raw request, the normalized intent, the session, the tenant configuration, etc. — as opposed to just the language model's output. This is what allows the record to carry information that a purely linguistic normalizer would not. The canonical intent record for an operation contains the same information that downstream routing and execution-control components would need to know about the user's intent. Thus, while a traditional intent-and-slot representation for an operation would merely contain information about what the user seems to want to have happen, the canonical intent record for that same operation also contains information about the operational implications, the governance implications, the risk implications, the side-effects, and the uncertainty associated with the user's request. The additional information in the

record above and beyond that in a typical intent-and-slot representation for an operation allows for a number of things, including: (1) equivalence comparison between paraphrases of a request; (2) schema and policy validation for proposed actions; (3) capability retrieval for proposed actions; (4) semantic caching of requests and their normalized forms; (5) trace correlation across services that handle a single request; (6) fallback handling when a service fails to handle a request; and (7) longitudinal analysis of route stability.

5. Reference Architecture and Constraint-First Routing Process

5.1. Reference Architecture

Figure 1 shows a conceptual reference architecture, structured in three planes: a semantic control plane, an execution plane and a governance and observability plane. The semantic control plane handles request intake, context assembly, intent normalization, uncertainty estimation, policy and authorization filtering, capability retrieval, route evaluation, and planning. The execution plane runs the committed route through an agent runtime, a workflow engine, a deterministic execution gateway, and the underlying tools, APIs, microservices, data stores, and human approval endpoint. The governance and observability plane captures policy decisions, propagates trace context, maintains the normalized intent identifier, and collects route decisions and execution outcomes into metrics, logs, and evaluation feedback that flows back into the control plane.

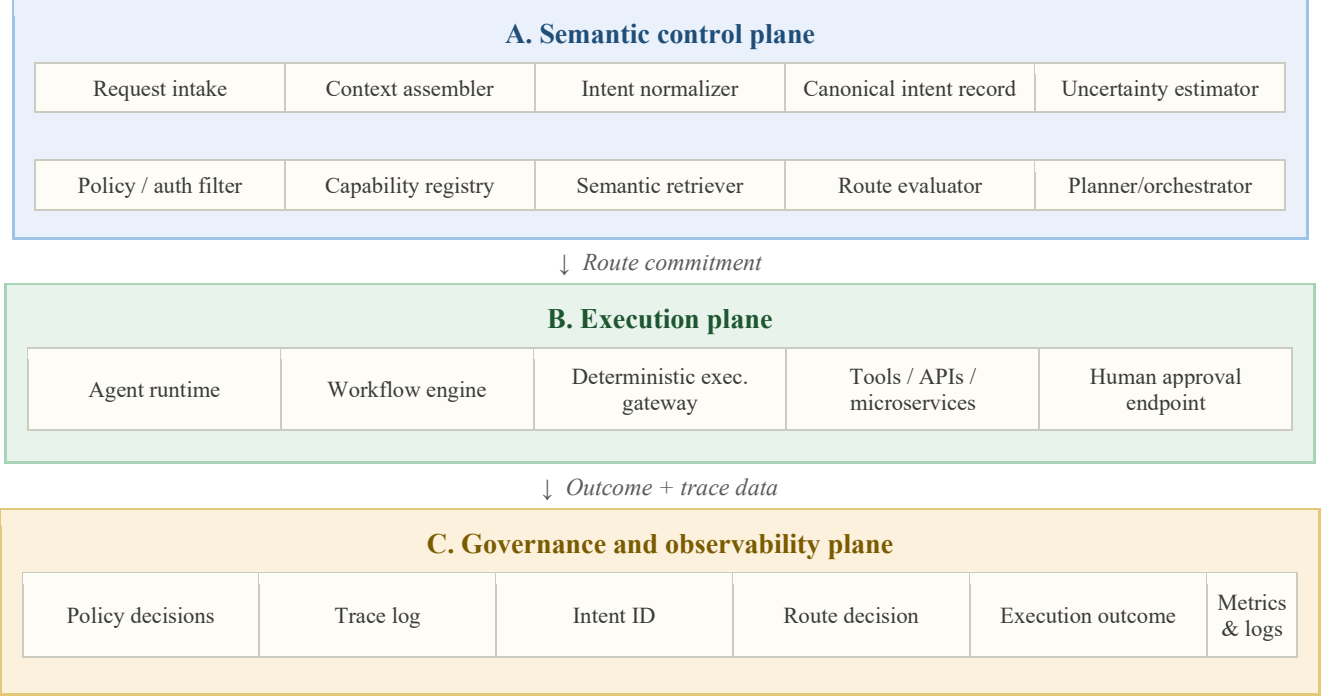


Fig. 1 Conceptual reference architecture separating the semantic control plane, the execution plane, and the governance and observability plane. Solid arrows indicate the request, normalization, capability-discovery, route-commitment, and outcome-collection flow; the dashed arrow indicates the feedback path from observability back into the control plane

In this architecture, LLMs and chat agents are specialized for interpretation, decomposition, semantic search and planning and handling exceptions. Normalization and orchestration are done by deterministic microservices (e.g. for transactions, payments, pricing, identity, stock, for authoritative read and writes, etc., including all compliance-critical cases). The architecture does not aim to replace microservices with chat agents and LLMs, but it is based on a deterministic execution substrate to which normalization and orchestration are added on top in parallel and with their own guarantees.

5.2. Constraint-First, Multi-Criteria Routing Process

No implemented and empirically validated weighting scheme exists, so routing is described here as a conceptual process rather than a scoring formula. The process is built of 10 conceptual stages: (1) request and context intake, (2) canonical intent construction, (3) capability-class identification, (4) hard eligibility filtering, (5) semantic candidate retrieval, (6) operational comparison, (7) confidence and consequence assessment, (8) route commitment, (9) controlled execution, and (10) fallback, recovery and telemetry.

First, all candidate functions for any given action are filtered by a set of hard eligibility rules, corresponding to a variety of factors, including but not limited to: authorization, tenant-specific policies for certain actions, service level objectives for certain actions, and risk classification of actions. Once a candidate function has made it through the eligibility rules for an action, it is ranked against a set of signals for that action, with no a priori specified weights for any of the signals. The set of signals for ranking functions to be called for a given action

includes the following: semantic and schema compatibility; reliability; past route success; latency; cost; geographical locality; service health; availability of dependent services and functions; stability of routes; and risk vs. consequence of failure for the action under consideration. As was stated previously, the above framework fixes the order of evaluation for any given action (i.e. first eligibility, then ranking). In other words, an ineligible function can never be chosen because it happens to score better on some operational criterion for some function than another function would have scored.

Conversely, a function that scores very well on some operational criterion can never be authorized to perform an action of very high irrevocability under low confidence. However, within the boundaries fixed by the above for the order of evaluation for any given action, there is considerable variability allowed. Specifically, within the boundaries fixed by the above for the order of evaluation for any given action, there are no fixed weights for any of the ranking signals for that action.

Instead, the weights for the ranking signals for any given action will vary based on a variety of factors, including but not limited to the above-mentioned domain at hand, tenant-specific policies for certain actions, service-level objectives for certain actions, and risk classifications for actions under consideration. For these reasons, the paper is implementation-neutral. In other words, a concrete implementation of the above framework could be based on ordered rules, a policy engine, a simple deterministic decision tree, learned ranking such as decision trees or support vector machines, or even a type of constrained optimization.

5.3. Execution Targets and Two-Stage Routing

The framework can distinguish between six execution targets for natural language commands, which are often mixed up when talking about them informally. A microservice is the smallest software unit that can be executed on a network to accomplish a given task and is characterized by being bounded, deterministic and provides a business or technical function. An API is a means to expose services, platforms, external systems or data in the form of an interface. A tool is a model-callable entity (function, method, API operation, data resource) with associated metadata for invocation as well as input and output.

A workflow describes a set of services, tools, approval steps and processing steps, which can be defined statically or dynamically. An agent is a reasoning entity that can accomplish goals by creating plans, choosing tools and coordinating and even adapting in case of need while executing tasks. A human approval endpoint is a point of governance where humans can approve, reject or even modify actions proposed for execution in the system. This implies that routing in such a system consists of two steps: first, selecting the appropriate capability class for a given input, and then selecting the correct endpoint or path to execute that capability.

5.4. Distributed-Systems Considerations

This framework is intended for distributed agentic microservices; it must support distributed execution semantics such as retries, timeouts, duplicate invocations, idempotency keys, at-least-once delivery, circuit breakers, partial workflow failure, compensation, route failover, stale registry metadata, schema drift, dependency failure, trace propagation, and eventual consistency with respect to classification of side effects and irreversible actions. None of these are not afterthoughts of the infrastructure. For example, if an operation is marked as irreversible in the canonical intent record, then that operation will require an idempotency key and a compensation path when it is routed to a service that does not offer transactional guarantees.

A semantic equivalence does not necessarily translate to execution equivalence; two similar requests (e.g., in terms of semantic meaning) could be directed to different routes for various reasons (e.g., user-level authorization, tenant-specific configuration, geographical restrictions, business-object state, service dependencies and health, service availability, transactional boundaries, potential side effects, compliance-related restrictions). The contextual information, policies, and side effects are all recorded in the canonical intent record so that they can be inspected and

used to reject semantically similar but wrong operation-execution routes even before they get executed.

5.5. Confidence and Consequence Assessment

Most papers discussed in this survey treat routing confidence and the consequence of an action for execution as two independent dimensions rather than mapping both to a single confidence value. Therefore, the low-confidence branch of Table III does not necessarily map to the low-consequence branch.

The confidence for routing decisions could be based on a variety of factors including: (1) the request ambiguity and the number of alternative interpretations considered by the model, (2) the confidence in the normalization process (i.e. the confidence that the input record accurately and completely describes the request), (3) the agreement between models for parsing different representations of the request, (4) the result of schema validation on the request, (5) the semantic retrieval separation between the top-ranked models for the request, (6) the freshness of the capability metadata for the invoked service, (7) the historical success rate for the invoked service for similar requests, and (8) the risk of the corresponding policy (i.e. potential impact on other related services and policies).

For the low-confidence branch of Table III, we distinguish between several categories of low confidence, since their causes are distinct and require different kinds of recovery. We note that normalization errors, found in the record itself, mean that the record is faulty and must be re-parsed or even re-generated. Ambiguity in a request is by definition an underspecified request that must be disambiguated against a context, possibly by asking a clarifying question.

Metadata for a capability to be invoked as a tool may be stale, meaning that the candidate registry for tools has gone stale and must be re-generated or even fall back to another candidate. Service or dependency failures are generally found during the operational comparison of services; thus must be able to fail over to another healthy service, or even return a safe error message to the user that invoked the failed service.

Lastly, a policy violation that was not discovered until late in the process of selecting a candidate for some purpose (e.g., a tenant) must always result in a hard rejection of that candidate, regardless of the confidence values. The tracing of all of these cases, including the functional scenarios of Section VI, is supported by the layers of the observability plane sketched in Section V-A.

Table 3. Confidence, Consequence, and Execution action

Confidence	Consequence	Action
High	Low	Direct execution
Moderate	Low	Verification or reranking
High	High	Deterministic validation or approval
Low	Any	Clarification, safe fallback, human approval, or refusal

6. Illustrative Conceptual Case Study

This section is developed to help understand Sections IV and V by means of a particular example enterprise e-commerce request: “Cancel my most recent order if it has not shipped, but do not cancel the subscription associated with it.” Note that this is only an example for explaining the framework, not a validated case on empirical grounds.

The request contains at least two ambiguities, namely, “most recent” needs to be disambiguated against the order history of the authenticating customer for this enterprise e-commerce application and “not shipped” needs to be checked against live fulfillment information as opposed to being assumed. Available information at the intake point in time includes the identity of the authenticated customer, the tenant or market of that customer, the recent order history of that customer, the conversation history with that customer, available credentials for accessing information about that customer and the service(s) availability information. Table IV outlines the flow of information from this information at intake through to observability in support of the diagnostic workflow outlined in Section V.

Table 4. Conceptual Case-Study Trace

Stage	Summary
Canonical intent	o = cancel_order; e = {order id, customer id}; c = must not cancel linked subscription; s = reversible while unshipped, else irreversible
Candidate classes	Order-inquiry service; cancellation workflow; fulfillment-status API; subscription-management service; support agent; human approval
Hard filtering	Excludes candidates that cannot verify ownership, do not support the customer’s market, risk cancelling the subscription, cannot perform idempotent cancellation, or cannot confirm fulfillment state
Operational comparison	Remaining candidates are compared on correctness, current availability, reliability, latency, transactional guarantees, dependency health, and consequences of failure.
Selected path	The agent interprets and decomposes the request; the deterministic cancellation workflow verifies identity, ownership, fulfillment state, cancellation eligibility, subscription exclusion, refund policy, and idempotency before committing.
Fallback	If fulfillment state is ambiguous or changes mid-execution: revalidate, stop, request clarification, route to a human, or return a safe refusal.
Observability	Records the normalized intent identifier, candidate set, eligibility decisions, selected route, validation evidence, execution outcome, fallback reason (if any), and final response

The example illustrates that an agent may interpret and decompose the request, but a deterministic service must still verify customer identity, order ownership, current fulfillment state, cancellation eligibility, subscription exclusion, refund policy, idempotency, and the final transactional outcome before the action is considered complete.

7. Future Evaluation Framework

Since this paper is conceptual, evaluation is proposed as future work rather than reported as a consequence. Future work could compare four conditions: (1) direct LLM capability selection; (2) semantic retrieval to LLM selection; (3) canonical intent normalization followed by retrieval; and (4) canonical normalization with policy-aware, operationally informed routing. The candidate metrics are route-selection accuracy, consistency over paraphrases, top-k capability recall, invalid-invocation rate, schema-mismatch rate, policy-violation rate, end-to-end latency, token consumption, execution cost, cache-reuse rate, duplicate-execution rate, fallback success, recovery success, route stability, trace completeness, and human-intervention rate.

A possible future evaluation dataset may contain similar paraphrases, ambiguous requests, context-dependent requests, unauthorized requests, high-consequence actions, unavailable services, stale schemas, conflicting operational constraints, and multi-step workflows. No such dataset is used in this paper, and no experimental table or figure in this paper should be taken to contain measured values.

8. Challenges, Research Agenda, and Limitations

8.1. Research Challenges

The most intractable problem is semantic ambiguity. Even the best joint intent and slot models and semantic parsers are plagued by under-specification and competing interpretations. In distributed agentic systems, two locally plausible candidates may be globally incompatible with policy or dependency constraints, motivating uncertainty representations that survive normalization and are visible to the routing layer.

Normalization overhead trades route consistency for higher latency and token cost; a promising approach is staged normalization, where a fast canonicalization pass handles common cases and deeper parsing or verification is invoked only when ambiguity or risk is high.

Capability metadata gets stale as tool schemas, service descriptions, and external APIs are ever changing, which is addressed by retrieval-augmented approaches such as Gorilla but not eliminated [11], motivating live capability refresh and schema-drift detection.

Routing decisions are increasingly based on where data might legally or organizationally travel, not only on semantic relevance, complicating privacy and trust-boundary management. MCP authorization guidance treats

access control as a transport concern that a policy filter must still evaluate [20], spurring research into federated intent normalization that preserves local data privacy while sharing canonicalization or routing policy across domains.

Benchmark fragmentation continues because existing benchmarks measure overlapping, partial slices of capability, such as API correctness, function selection, or large-scale tool discovery [9], [10], [12]–[17]; a benchmark that jointly evaluates semantic correctness, routing stability, latency, cost, fallback behavior, and trace completeness for normalized service routing is still missing.

Other future work includes uncertainty-preserving canonical representations, adaptive routing policies, cross-layer routing with agents and infrastructure, route stability testing on paraphrased requests, and causal analysis of routing failures.

8.2. Limitations

This is a conceptual framework for a contribution. There is no implementation of the prototype, and thus no new benchmark results are reported. The canonical intent record is a conceptual abstraction, and thus no standard schema is proposed. The routing criteria and their importance vary by domain, by tenant, and by risk classification, and thus, this paper does not attempt to prescribe any universal weights. The illustrative case study in Section VI is not an empirical validation. The evidence in Section II includes a number of very different research areas, and thus the reported tool-routing and MCP sources are mostly recent preprints and are not yet peer-reviewed publications. It remains to be seen whether the individual components of this new framework can be validated, but this will require a future prototype and comparative evaluation along the lines proposed in Section VII. This is a deliberate scope rather than an oversight, and the prototyping of the proposed taxonomy, representation and routing process against a suitable benchmark (as surveyed in Section II) is a large undertaking that will build on this paper as conceptual groundwork. Reporting a small, uncontrolled implementation here would be to misleadingly

present such evidence of validation when, in fact, it would not meet the criteria for a valid evaluation as proposed in Section VII.

9. Conclusion

Our literature review indicates that distributed agentic systems, including intent normalization and service routing, are best viewed as a single integrated problem. This is a problem at the edge of Natural-Language Understanding (NLU) and distributed-systems control. We described in Section IV an intent record that, if structured semantically, can serve as the basis for a constraint-first, multi-criteria filtering of services to consider, followed by a comparison of candidates as described in Section V.

This synthesis is anchored by a fundamental design principle that seeks to separate the space of semantic control from that of purely deterministic execution, with a cross-cutting governance and observability plane that spans both spaces. The natural language interpretation and decomposition agents, as well as the language models, are focused on the space of managing interpretation, decomposition, and exception handling. The deterministic, user-space-based, transactional microservices manage the space of transactions, policy enforcement points, etc., that must all operate with absolute determinism and that are otherwise largely irreversible. The space of routing and execution decisions is made observable, and thus able to form the basis for future normalization and routing policy, by having that space's decisions and resulting execution's effects written to and afterwards consulted from probes in that space.

In this paper, we presented a starting point for the design of distributed multi-agent service systems, in the form of a taxonomy, a canonical intent record, a reference architecture and a simple case study. The future work needed to empirically evaluate the components of this framework within a variety of benchmarks and deployments is outlined in the future evaluation framework in Section VII and the research agenda in Section VIII.

References

- [1] Fu Bang, "GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings," *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pp. 212-218, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [2] Qian Chen, Zhu Zhuo, and Wen Wang, "BERT for Joint Intent Classification and Slot Filling," *arXiv*, pp. 1-6, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [3] Jonathan Berant, and Percy Liang, "Semantic Parsing via Paraphrasing," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1415-1425, 2014. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [4] Ehud Karpas et al., "MRKL Systems: A Modular, Neuro-Symbolic Architecture that Combines Large Language Models, External Knowledge Sources and Discrete Reasoning," *arXiv*, pp. 1-19, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [5] Shunyu Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," *International Conference on Learning Representations (ICLR)*, pp. 1-33, 2023. [[Google Scholar](#)] [[Publisher Link](#)]
- [6] Timo Schick et al., "Toolformer: Language Models Can Teach Themselves to use Tools," *Advances in Neural Information Processing Systems*, Neural Information Processing Systems Foundation, Inc. (NeurIPS), vol. 36, pp. 1-13, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [7] Yongliang Shen et al., “HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 1-27, 2023. [[Google Scholar](#)] [[Publisher Link](#)]
- [8] Qingyun Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,” *arXiv*, pp. 1-43, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [9] Minghao Li et al., “API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs,” *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, pp. 3102-3116, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [10] Yujia Qin et al., “ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs,” *arXiv*, pp. 1-24, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [11] Shishir G. Patil et al., “Gorilla: Large Language Model Connected with Massive APIs,” *arXiv*, pp. 1-18, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [12] Fanjia Yan et al., Berkeley Function-Calling Leaderboard, Berkeley Sky Computing Lab, 2024. [Online]. Available: https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html
- [13] Zhicheng Guo et al., “StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of LLMs,” *arXiv*, pp. 1-14, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [14] Guozhao Mo et al., “LiveMCPBench: Can Agents Navigate an Ocean of MCP Tools?,” *arXiv*, pp. 1-18, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [15] Zhenting Wang et al., “MCP-Bench: Benchmarking Tool-using LLM Agents with Complex Real-World Tasks via MCP Servers,” *arXiv*, pp. 1-52, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [16] Xuanqi Gao et al., “MCP-RADAR: A Multi-Dimensional Benchmark for Evaluating Tool use Capabilities in Large Language Models,” *arXiv*, pp. 1-20, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [17] Chaithanya Bandi et al., “MCP-Atlas: A Large-Scale Benchmark for Tool-Use Competency with Real MCP Servers,” *arXiv*, pp. 1-25, 2026. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [18] Anthropic, Introducing the Model Context Protocol, 2024. [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>
- [19] Model Context Protocol, Specification, 2025. [Online]. Available: <https://modelcontextprotocol.io/specification/2025-03-26>
- [20] Model Context Protocol, Authorization, 2026. [Online]. Available: <https://modelcontextprotocol.io/specification/2025-03-26/basic/authorization>
- [21] Google, Announcing the Agent2Agent Protocol (A2A), Google Developers Blog, 2025. [Online]. Available: <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/>
- [22] Kubernetes SIG Network, Gateway API: Introduction. [Online]. Available: <https://gateway-api.sigs.k8s.io/docs/introduction/>
- [23] OpenTelemetry, Documentation. [Online]. Available: <https://opentelemetry.io/docs/>
- [24] World Wide Web Consortium, Trace Context, W3C Recommendation. [Online]. Available: <https://www.w3.org/TR/trace-context>