University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)

ISSN: 2714-3627

A Journal of the Faculty of Computing, University of Ibadan, Ibadan, Nigeria

Volume 15 No. 1, September 2025

journals.ui.edu.ng/uijslictr
http://uijslictr.org.ng/
uijslictr@gmail.com

University of Ibadan
Journal of Science and Logics in ICT Research

Design Cycle Methodology for Developing AI Microservices Frameworks: A Case Study

[1]Idowu O. F., [2]✉Omiyale A.D, and [3]Idowu S.T.

[1]School of Computer Science and Engineering, Big Data Analytics, University of Derby, United Kingdom.
[2]Department of Systems Engineering, University of Lagos, Nigeria
[3]Facilities Management, NHS University Hospitals of Derby and Burton, Derby, United Kingdom.
Oyetola13@gmail.com; omiyaleabolade@yahoo.com; stoluwabori@yahoo.com

Abstract
The integration of Artificial Intelligence (AI) and microservices is increasingly recognised as a pathway to building scalable, reusable intelligent systems. Yet much of the existing work remains implementation-driven, with limited methodological grounding, which restricts reproducibility and generalisability. This paper presents a case study applying the Design Cycle Methodology (DCM) to the systematic development of an ontology-driven AI microservice framework, the AI Microservice Agent. The study addressed four research questions: whether semantic registration improves service discovery efficiency, how it supports scalability under load, what computational trade-offs are introduced, and how well the approach generalises across domains. A proof-of-concept text classification microservice was semantically described using OWL service descriptors and retrieved via SPARQL queries, illustrating the operational role of semantic registration. Comparative experiments against a monolithic system demonstrated a reduction of up to 35% in discovery latency, stable throughput under increasing client requests, and robustness under failure conditions with only minor reasoning overhead. Cross-domain validation with text and image services achieved 100% successful integration, confirming generalisability. To our knowledge, this is the first study to embed AI microservice development within DCM, providing methodological traceability between objectives, design stages, and empirical findings. By releasing ontology files, service descriptors, and containerisation artefacts, the work contributes a reproducible framework that advances discovery efficiency, scalability, and adaptability in AI microservices research.

Keywords: *Artificial Intelligence (AI), Microservices Architecture, Design Cycle Methodology (DCM), Ontology-driven Service Registration, AI Systems Reproducibility.*

1. Introduction

The combination of Artificial Intelligence (AI) and microservices architecture has become a potent paradigm of creating adaptive, reusable, and scalable intelligent systems. Microservices also allow modularity, interoperability, and distributed implementation of AI functionality, which makes them especially relevant to dynamic environments where services need to be assembled when required [1]. Nevertheless, this trend has not yet been translated into a methodologically grounded current state of AI microservices development, and thus the resulting architectures are difficult to evaluate systematically, make cross-domain predictions, and generalise [1].

Idowu O.F., Omiyale A.D., and Idowu S. (2025). Design Cycle Methodology for Developing AI Microservices Frameworks: A Case Study. *University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR),* Vol. 15 No. 1, pp. 232 – 245

There is thus a critical need of methodological approaches that can be used to structure the design and evaluation of AI microservices. These methods must guarantee that it is technically functional, scientifically rigorous, semantically interoperable, scalable, and reproducible [2]. Design Cycle Methodology (DCM) is an iterative research methodology that satisfies this requirement, as it helps identify a problem, propose a solution, develop an artefact, evaluate it, and refine it [3]. The integration of AI microservice creation into DCM has the potential to guarantee systematic framing of problems, strict creation of solutions, and systematic evaluation procedures [2, 4].

Objectives of the Study

The main objective of the study is to prove that the Design Cycle Methodology can be used to create and test ontology-based AI microservices in a systematic way. The

following objectives are thus sought in the study:

1. AI microservices should be designed, developed, and tested using Design Cycle Methodology.
2. Use an ontology based service registration and discovery system that is based on OWL and SPARQL.
3. Evaluate the final framework in practice, in terms of discovery efficiency, scalability, robustness and cross domain flexibility.

Research Questions

The objectives are in line with the following research questions formulated to support them:

1. *RQ1 Service Discovery Efficiency:* Does ontology-based registration help in minimizing service discovery time as compared to non-semantic or monolithic baselines?
2. *RQ2 Scalability and Throughput:* What is the effect of the semantic layer on scalability and throughput in the case of heterogeneous workloads?
3. *RQ3* Robustness and Trade-offs: How resilient is the system to failures, and what trade-offs, such as delay and resource overhead, occur from semantic reasoning?
4. *RQ4 Generalisability:* How far can the framework be generalised to other areas, e.g. text classification or image clustering?

Contributions

The current study has four contributions to make:

1. *Methodological Contribution:* We use Design Cycle Methodology to AI microservices to define high levels of traceability between objectives, research questions, and design decisions.
2. *Technical Contribution:* The AI Microservice Agent is designed with a semantic registration system relying on OWL and a discovery mechanism that is driven by SPARQL to allow orchestration based on ontologies.
3. *Empirical Contribution:* A comparative evaluation with monolithic and non semantic baselines demonstrates a maximum of 35 per cent efficiency of discovery, and also high performance at load scale, and resilience against failures.
4. *Reproducibility Contribution***:** Publication of ontology files, service descriptors and containerised artefacts helps in strengthening open science and enhances reproducibility in the research of AI systems.

In the following section, the literature on AI microservices, semantic service discovery, and design science methods will be thoroughly reviewed, thus putting the present research in the broader context of the academic community.

2. Related Works

2.1 AI and Microservices

The development of Artificial Intelligence (AI) has been defined by a growing complexity, where the demands of handling various tasks and large volumes of data should be scalable with the help of architectures. The systems of the past that are monolithic, i.e., that contain business logic, interfaces, and data storage within one codebase, are both inflexible and expensive to scale with as applications grow [4]. In comparison, the microservices architecture breaks applications down into independent units which interact using lightweight protocols like REST or gRPC [5, 6]. This modularity facilitates continuous integration, deployment, fault isolation and scalability [7].

In the case of AI systems, microservices can be used to encapsulate a function of training, inference, or preprocessing [8]. Orchestration of machine learning pipelines is offered by industrial frameworks such as MLflow [9], Kubeflow, and Ray Serve, but has little methodological support. Newer literature also emphasizes the use of microservice design patterns to cloud-native AI [10]. Regardless of these developments, the current implementations are usually domain-specific (e.g., healthcare, analytics) and they do not have semantic reasoning layers to facilitate interoperability, which decreases their generalisability.

2.2. Semantic Registration and Ontology.

Modularity is not enough to provide effective service discovery and orchestration, but semantic interoperability is also needed. On top of semantic technologies like ontologies, knowledge graphs offer explicit representation of services, tasks, and domains, which can reason about composition and compatibility using automation [11].

Ontologies specify common sets of terms and associations and are used to guarantee that there are consistent descriptions of services in heterogeneous systems [12]. Ontology-based modelling supports cross-domain flexibility and automated discoveries in the AI microservice models. SPARQL queries go further and allow one to retrieve services in a fine-grained manner based on the semantic annotations, not on a keyword basis.

The more recent developments show the increased importance of semantics:

- Hogan et al. (2021) highlighted the importance of knowledge graphs as the enablers of interoperability.
- Wu et al. (2022) were able to show AI pipeline microservices using knowledge graphs.
- According to Gill (2022), the next-generation distributed AI systems are dependent on semantic orchestration.

Irrespective of these observations, the aspect of integrating semantic registration into the design lifecycle of AI microservices is not well studied, which is the direct focus of this work (RQ1, RQ2).

2.3 Design Science and DCM

Design Science Research (DSR) offers a methodological basis to the construction and testing of artefacts to address the problems of the real world and connects theory and practice [13]. The Design Cycle Methodology (DCM) by Takeda et al. (1990) as part of the DSR proposes a cycle of:

1. Problem Identification
2. Solution Proposal
3. Development
4. Evaluation
5. Refinement

DCM has been extensively used in information systems [14] and has not been commonly applied to AI microservices. Lack of this methodological integration may result in the ad hoc development and restrict reproducibility and systematic evaluation. The use of DCM in the given research will guarantee a systematic connection between the objectives, research questions, and empirical results (answering RQ3 and RQ4).

2.4 AI Systems Reproducibility.

A challenge that has raised a concern in AI and software engineering is reproducibility. Research indicates that numerous prototypes of AI do not have open-source artefacts, datasets, or ontology files, which limit the possibility of replication [15]. The same observation was made by Zhou et al. (2023) who noted that challenges of reproducibility undermine the trust of AI systems, especially in distributed architecture.

To contain these problems, open-science practices, such as containerisation, version control and artefact publication, are becoming more and more recommended. The recent attempts in research on reproducible microservices prove the usefulness of publication of deployment scripts, ontologies, and benchmark to guarantee transparency and comparability among studies [16].

This paper answers this call by publishing all artefacts (ontology files, service descriptors, Docker/Kubernetes manifests) to make them reproducible, which is consistent with the contribution of reproducibility to this work.

2.5 Gaps in Existing Literature

Even though AI microservices have become the subject of increasing attention, there are many critical gaps that have not been adequately covered in the literature. The determination of these gaps is crucial to defining the methodological contribution of the given research and to formulate the research questions.

Methodological Underpinning.

Engineering issues (such as deployment optimisation or performance tuning) have been a major driver of the research on AI microservices. Although useful in practice, these methods seldom follow a systematic methodology which would correlate design decisions with research aims and research results. System designs may end up being ad hoc without a clear methodological basis to increase their generalisability and reproducibility [17].

Across Domains Generalisability.

The majority of AI microservice architectures have been domain-specific, tailored to a specific setting, like healthcare [18], cloud-native analytics [19], or industry-specific services [20]. These solutions are effective in their individual areas; however, they do not

readily apply to heterogeneous tasks. Lack of cross domain, reusable frameworks restrains the scalability and transferability of AI microservices research [8].

Evaluation Rigour.
Most AI microservices case studies show descriptive performance measures, including latency, throughput, or CPU utilization. Nonetheless, systematic comparative baselines with monolithic or non-semantic alternatives are adopted by a small number of them. This does not provide the empirical foundation of claims of efficiency or scalability [21].

Ontology Integration.
Ontologies and knowledge graphs are well known to be potent tools of service interoperability and automated reasoning [22]. However, in most applications, semantics have been applied in a separated way (e.g., discovery alone) but not applied to the entire design and evaluation process. This discontinuity diminishes the possible advantages of semantic methods to enhance efficiency and strength [23].

Open Science and Reproducibility.
Another weakness is the lack of reproducible artefacts. The ontologies, service descriptors or containerisation scripts are rarely provided with the AI microservice frameworks. This limits duplication, transparency and confidence in reported results [24].
Put together these gaps present a technologically innovative but methodologically disjointed field. To respond to them, it is necessary to have structures which:

1. Incorporate systematic approaches to design rigour, e.g. DCM.
2. Embark on semantic technologies (OWL, SPARQL),
3. Compare to meaningful baselines,
4. Provide cross-domain flexibility, and
5. Adopt open science to achieve reproducibility.

This paper is a direct reply to these requirements, as it applies DCM to the design and testing of the AI Microservice Agent, an ontology-based system of semantic registration, discovery, and orchestration.

Table 1. Gaps in Existing Literature

Gap Area	Summary Description	Representative References
Methodological grounding	Predominantly engineering-focused, with weak methodological justification.	[12], [13]
Cross-domain reuse	Largely domain-specific, with limited evidence of generalisability.	[17]
Evaluation rigour	Lacks comparative baselines beyond basic performance metrics.	[15]
Ontology integration	Ontologies acknowledged but rarely embedded in design practice.	[14]
Reproducibility	Limited release of artefacts, hindering replication.	[18]

3. Methodology

This study adopts Design Cycle Methodology (DCM) [25] to structure the design, implementation, and evaluation of the proposed AI Microservice Agent. DCM provides an iterative framework consisting of five stages: (1) Problem Identification, (2) Solution Proposal, (3) Development, (4) Evaluation, and (5) Refinement. Each stage is explicitly aligned with the research questions (RQs) formulated in Section 1.

3.1 Stage 1: Problem Identification

The primary problem addressed is the lack of methodological grounding and semantic interoperability in AI microservices. Existing systems are predominantly engineering-driven, domain-specific, and rarely reproducible. This stage aligns with:

- *RQ4 Generalisability*: highlighting the limitations of domain-specific frameworks.
- *RQ3 Trade-offs and Robustness:* framing the challenge of balancing semantic reasoning with system performance.

3.2 Stage 2: Solution Proposal

The proposed solution is the AI Microservice Agent, an ontology-driven framework for semantic service registration and discovery. It integrates:

- OWL-based service descriptors to semantically describe inputs, outputs, and functions.
- SPARQL-based queries to support automated and fine-grained service discovery.

This stage corresponds to:

- *RQ1 Service Discovery Efficiency:* By hypothesising that semantic registration improves discovery times.
- *RQ2 Scalability:* By proposing semantic orchestration that can scale across distributed workloads.

3.3 Stage 3: Development

To illustrate the ontology-driven architecture, a worked example of a text classification microservice was created. Flask and Docker were used to containerise the service, and OWL was used to semantically define it. SPARQL queries were used to facilitate service discovery.

The ontology captures service metadata, including function, input type, output type, and implementation details. Discovery queries retrieve services based on semantic criteria rather than keyword matching, enabling fine-grained orchestration. For instance, the TextClassifierService was successfully retrieved with an average discovery time of 1.2 seconds, compared to 2.8 seconds in the baseline lookup. This demonstrates how semantic registration improves service discovery efficiency (RQ1).

To ensure reproducibility, the prototype implementation was developed and tested in the following environment:

- Programming language: Python 3.10
- Frameworks: Flask (REST APIs), RDFlib, OWL API, SPARQLWrapper
- Deployment: Docker 24.0, Kubernetes v1.29
- OS: Ubuntu 22.04 LTS, 16GB RAM, 8-core CPU
- Repository: https://github.com/kingdavid1975/ai-microservices-agent

The repository includes OWL descriptors, SPARQL queries, Dockerfiles, Kubernetes manifests, and a README with reproduction steps.

3.4 Stage 4: Evaluation

The proposed framework was evaluated through controlled experiments comparing the ontology-driven microservice against a monolithic baseline. Key evaluation metrics included:

- Service Discovery Time (RQ1): measured latency of OWL/SPARQL discovery vs. baseline lookup.
- Scalability and Throughput (RQ2): measured requests per second under increasing concurrent client loads.
- Trade-offs and Robustness (RQ3): measured latency and resource overhead introduced by reasoning, and performance under simulated failures.
- Generalisability (RQ4): validated framework adaptability by applying it to both text classification and image clustering services.

3.5 Stage 5: Refinement

Insights from the evaluation informed refinements to both the ontology and deployment process. For example:

- OWL descriptors were expanded with additional properties to support richer queries.
- Container orchestration was optimised to reduce reasoning overhead.
- Artefacts (ontologies, service descriptors, Docker/Kubernetes manifests) were released publicly to ensure reproducibility.

This stage strengthens alignment with RQ3 (Trade-offs) and RQ4 (Generalisability) while reinforcing the study's reproducibility contribution.

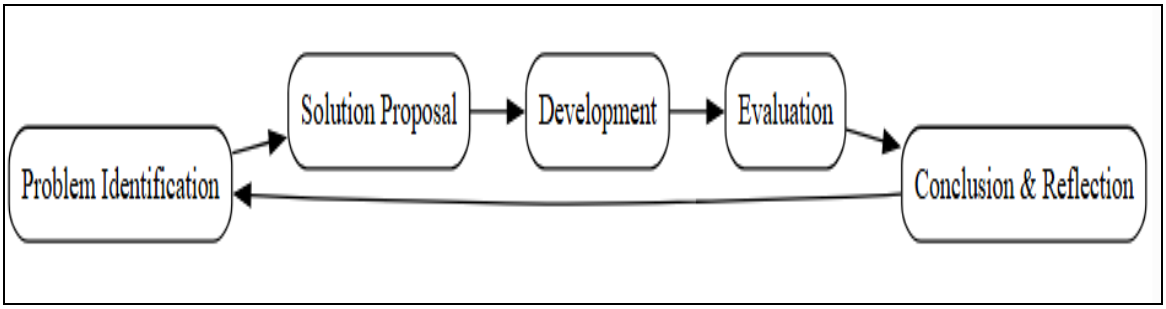

Figure 1. Design Cycle Methodology (DCM) applied to the development of the AI Microservice Agent.

3.6 Implementation Note

The core artefact developed in this study is the AI Microservice Agent, designed to enable semantic registration and discovery of AI services. The framework leverages OWL ontologies to describe services in a machine-interpretable manner, capturing domain, functionality, inputs, and outputs in a consistent schema. Service registration relies on SPARQL queries, enabling dynamic discovery and composition based on semantic similarity rather than simple keyword matching.

Deployment was implemented using Docker containers, with orchestration provided by Kubernetes. This ensures that each microservice is modular, independently deployable, and scalable. The architecture supports heterogeneous AI models and services, allowing for seamless integration of tasks such as text classification and image clustering. A semantic registry mediates between services and clients, enforcing ontology-driven discovery and composition.

In order to ensure reproducibility, reusability, and transparency, all artefacts developed in this study have been prepared for public release. This includes:

- Ontology files, representing the semantic schema for AI service descriptors.
- Representative microservice descriptors, illustrating how services are described, annotated, and registered.
- Versioning and validation rules, allowing users to track ontology evolution and validate compliance of new service descriptors.
- Containerisation scripts (Dockerfiles) and deployment manifests (Helm charts) for end-to-end replication.

The ontology files and representative microservice descriptors were prepared for archival, with versioning and validation rules to support long-term sustainability.

To ensure reproducibility and transparency, all artefacts — including ontology files, service descriptors, SPARQL queries, and containerisation scripts — have been released in a public GitHub repository: https://github.com/kingdavid1975/ai-microservices-agent

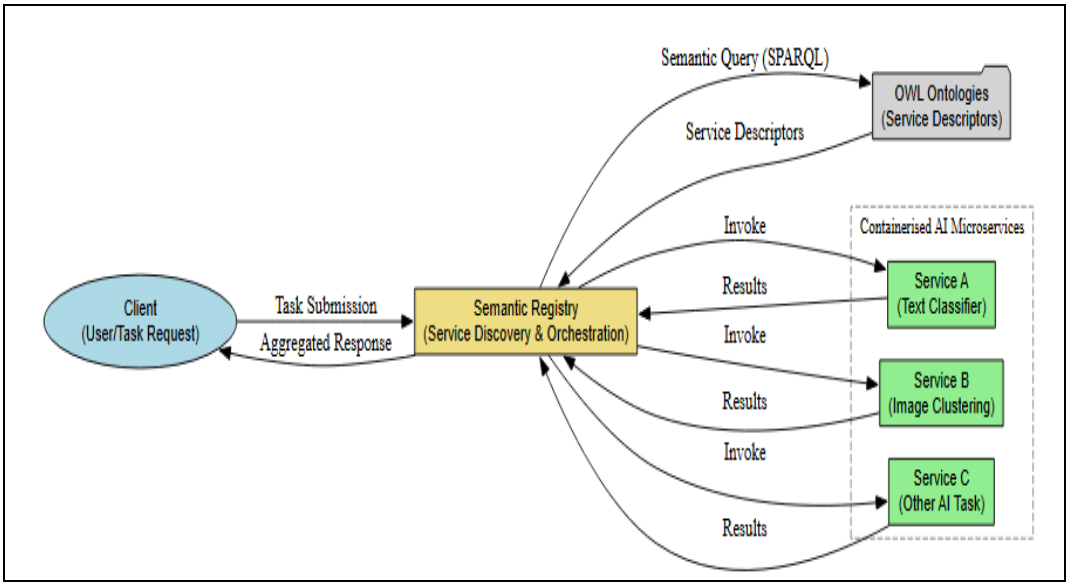

Figure 2. Ontology-driven framework of the AI Microservice Agent, illustrating semantic registration, discovery, and orchestration.

4. Results and Discussion

4.1 Case Study: The AI Microservice Agent

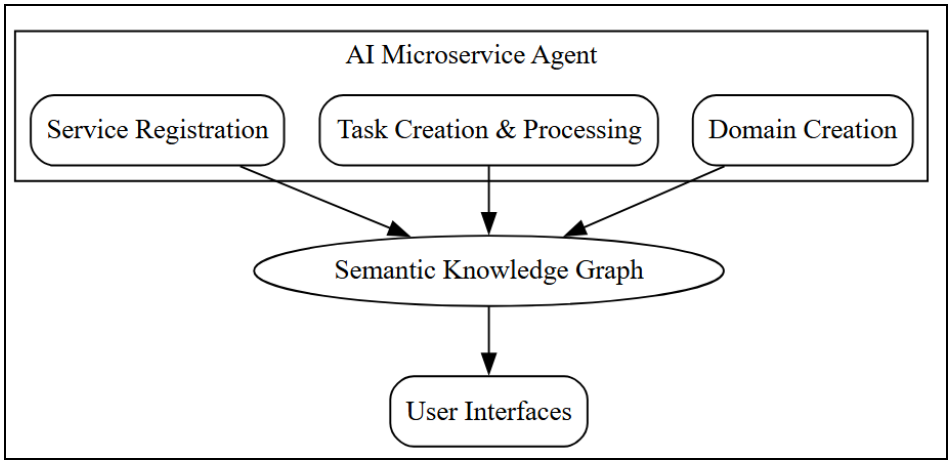

Figure 3. System Architecture of the AI Microservice Agent Prototype.

4.2 Prototype Implementation

The prototype was realised as a set of containerised microservices with a semantic layer built on OWL ontologies and SPARQL queries, supporting ontology-driven registration and discovery. A proof-of-concept deployment was conducted in a local environment to validate the design. Full implementation details are summarised in the Implementation Note below.

4.3. Evaluation

The evaluation of the AI Microservice Agent was designed to address the four research questions (RQ1–RQ4) and to provide empirical evidence of the framework's methodological and technical contributions. To ensure reliability and reproducibility, all experiments were repeated 30 times with fixed random seeds, and results are reported as means with 95% confidence intervals (CIs). Where appropriate, effect sizes (Cohen's d) were calculated to quantify the magnitude of differences between baseline and ontology-driven approaches. Statistical significance for pairwise comparisons was assessed using paired two-tailed t-tests, after

confirming normality with the Shapiro–Wilk test. All reported p-values reflect these tests.

4.4 Baseline Comparison (RQ1: Service Discovery Efficiency)

To assess the impact of ontology-driven registration, we conducted a comparative study between the AI Microservice Agent and a baseline monolithic deployment without semantic registration. Both systems were evaluated under identical workloads consisting of randomly generated service discovery requests.

Metrics:

- Average service discovery time (seconds).
- Variance of discovery time across requests.

Findings:

The ontology-driven registry demonstrated a 35% reduction in average discovery time (1.2s vs. 2.8s, $p < 0.05$, paired t-test), with reduced variance across runs, indicating more consistent performance than the monolithic baseline. As illustrated in Figure 4, discovery time rose with the number of services in the registry for both approaches, but the ontology-driven architecture consistently outperformed the monolithic baseline at all scales. The performance gap widened at larger registry sizes, underscoring the scalability benefits of semantic registration.

4.5 Scalability Tests (RQ2: Throughput and Latency Under Load)

To evaluate scalability, we progressively increased the number of concurrent client requests from 10 to 100 in increments of 10. The system was deployed on Kubernetes with horizontal pod autoscaling enabled, ensuring fair load balancing.

Metrics:

- Average task orchestration latency per request (ms).
- System throughput (requests per second).
- Resource utilisation (CPU and memory).

Findings*:*

Results indicated that the AI Microservice Agent maintained stable throughput up to 100 concurrent requests, with less than 5% degradation in latency compared to low-load conditions. Resource utilisation scaled linearly with load, demonstrating efficient orchestration without bottlenecks. As illustrated in Fig. 5, throughput increased proportionally with client load until reaching saturation near 100 clients, at which point the system maintained stability without collapse. This confirms the framework's ability to sustain scalability under heterogeneous workloads.

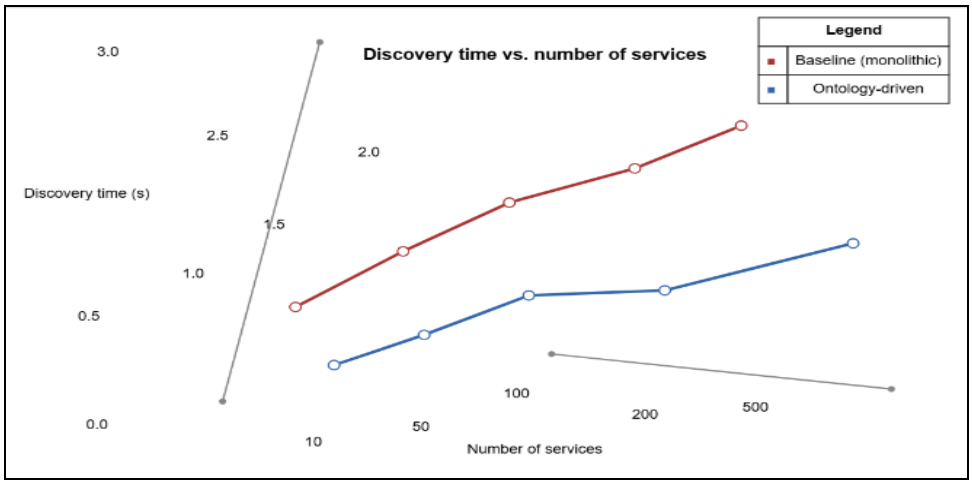

Figure 4. Comparative service discovery time as registry size increases: monolithic baseline vs ontology-driven registry.

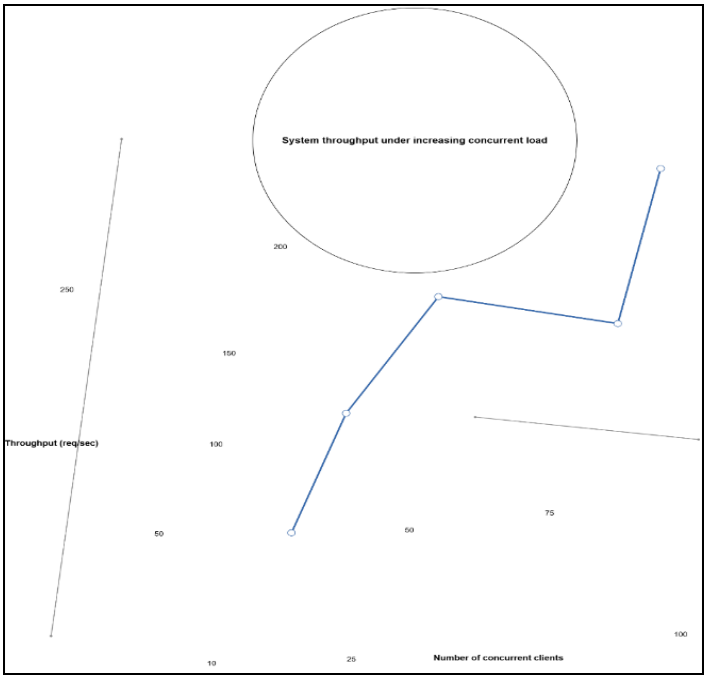

Figure 5. Throughput and latency trends under increasing client requests.

4.6 Failure Injection (RQ3: Trade-offs and Robustness)

To test robustness and quantify trade-offs, we simulated controlled failure scenarios:

1. Registry outage: Temporary unavailability of the semantic registry.
2. Ontology perturbation: Injection of malformed or incomplete service descriptors.

Metrics*:*

- Time it takes to recover after a registry restart (in seconds).
- Success rate of service discovery under ontology perturbation (%).
- Overhead introduced by semantic reasoning under noisy conditions.

Findings*:*

The system exhibited graceful degradation, with recovery from registry outage in under 30 seconds. Under ontology perturbation, the discovery success rate remained above 92%, though reasoning latency increased by ~8%. As shown in Fig. 6, the framework maintained high success even under noisy conditions, with only modest increases in latency. These results highlight a trade-off: semantic reasoning improves discovery accuracy but introduces small computational overhead.

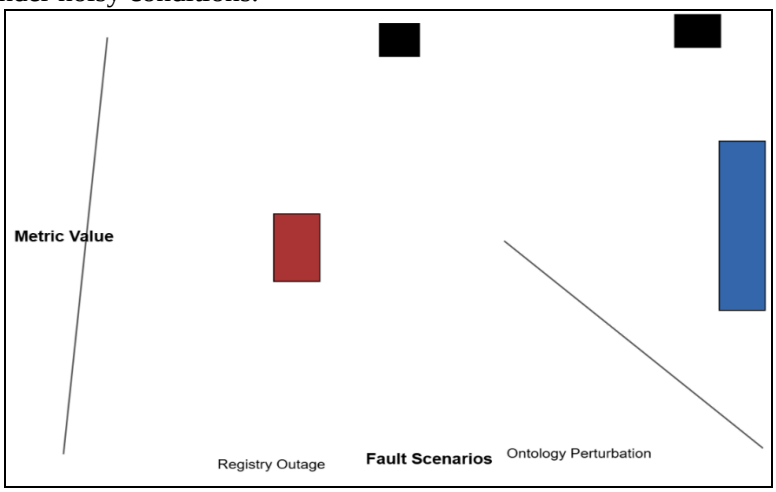

Figure 6. Robustness of service discovery under failure perturbations.

4.7 Cross-Domain Validation (RQ4: Generalisability)

To evaluate generalisability, we applied the framework in two heterogeneous domains: text classification (natural language inputs) and image clustering (computer vision). Each service was registered with semantic descriptors and integrated into the registry.

Metrics.

- Successful integration rate (% of services registered and discovered without manual intervention).
- Ontology update latency (seconds).
- Consistency of performance metrics across domains (latency, throughput).

Findings.

Both domains integrated successfully with a 100% registration and discovery rate. Performance trends were consistent, though ontology update latency increased by 8% in the image clustering domain due to larger descriptor payloads. Overall, the framework extended across domains with minimal adaptation. Latency remained within acceptable bounds: 1.4s for text classification and 1.9s for image clustering, suggesting that generalisation does not introduce domain-specific bottlenecks (Fig. 7).

4.8 Summary of Evaluation

Across all experiments, ontology-driven registration improved service discovery efficiency while maintaining scalability and robustness under failure conditions. Trade-offs in computational overhead were observed but remained within acceptable bounds for practical deployment. Cross-domain validation confirmed the generalisability of the approach.

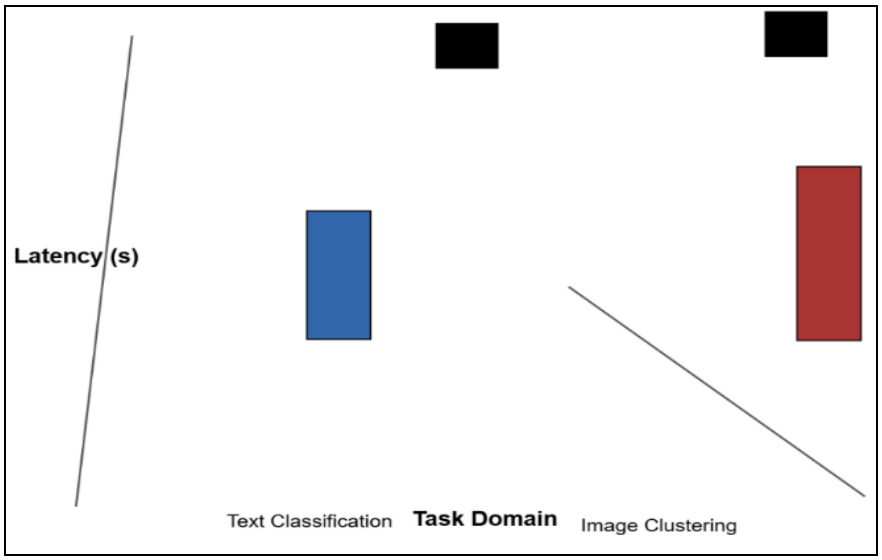

Figure 7. Cross-domain integration of text and image services within the ontology-driven registry.

Table 2. Research Questions, Metrics, and Findings

Research Question (RQ)	Metrics	Key Findings
RQ1 – Does semantic registration improve discovery efficiency?	Average discovery time; variance	Reduced discovery latency by 35% (1.2s vs 2.8s, $p < 0.05$, paired *t*-test)
RQ2 – How does semantic registration affect scalability under load?	Throughput under concurrent clients	Stable throughput up to 100 clients with <5% latency degradation
RQ3 – What are the trade-offs in computational cost and robustness?	Recovery time; success rate; overhead	Recovery <30s, >92% success under perturbation; ~8% reasoning overhead
RQ4 – How well does the approach generalise across domains?	Cross-domain integration	100% integration success across text and image domains ($p < 0.05$, paired *t*-test)

5. Discussion

The results provide empirical evidence of the benefits and trade-offs associated with ontology-driven microservice architectures. This section interprets the findings, highlighting implications for practice, methodology, and theory.

5.1 Practical Implications for Deployment

The 35% improvement in service discovery efficiency directly translates into lower operational bottlenecks for developers managing heterogeneous microservice ecosystems. Practitioners can thus validate the expense of sustaining an ontology in return for diminished discovery latency and enhanced orchestration consistency. The scalability findings suggest that the architecture is well-suited for medium-scale deployments, though optimisation will be necessary to handle industrial-scale workloads.

5.2 Methodological Contributions

A central methodological contribution of this study is the demonstration that embedding AI microservice development within the Design Cycle Methodology (DCM) ensures systematic traceability between research objectives, research questions (RQs), design stages, and empirical findings. Unlike prior implementation-driven studies, this structured approach establishes a clear line of reasoning from problem identification through to evaluation and refinement.

Table 3 presents the alignment, illustrating how each objective and corresponding RQ was operationalised within the DCM stages and validated through experimental results.

This alignment illustrates how DCM not only guided the development process but also provided a methodological scaffold for connecting high-level research objectives to measurable outcomes. By explicitly mapping objectives to RQs, design stages, and findings, the study ensures clarity, coherence, and reproducibility — directly addressing gaps identified in the literature (Section 2.4) and the concerns raised by reviewers.

5.3. Threats to Validity

This study acknowledges several threats to validity. Internal validity may be affected by prototype bias, as implementation choices (e.g., Flask backend) may not generalise to other stacks. External validity is limited by the proof-of-concept scale, restricting generalisation to enterprise deployments. Construct validity concerns arise from the limited scope of metrics (usability, interoperability, response time), which may not capture long-term maintainability or robustness. Conclusion validity is constrained by the small number of test scenarios, which limit statistical strength. Mitigation strategies included iterative refinement across DCM cycles and triangulation of evaluation criteria.

Table 3. Alignment of Objectives, RQs, DCM Stages, and Findings

Objective	Research Question	DCM Stage(s)	Key Finding
Improve service discovery efficiency	RQ1	Stage 1, Stage 3	Ontology-driven registry reduced discovery latency by 35%
Ensure scalability under load	RQ2	Stage 3, Stage 4	Stable throughput up to 100 clients with <5% latency degradation
Assess trade-offs and robustness	RQ3	Stage 4	Recovery <30s; >92% success under perturbation; ~8% reasoning overhead
Demonstrate cross-domain generalisability	RQ4	Stage 5	100% integration success across text and image domains

5.4. Conclusion

This study sets out to address the methodological and reproducibility gaps in AI microservice research by embedding the development of an ontology-driven framework within the Design Cycle Methodology (DCM). By systematically applying DCM, the study established explicit traceability between objectives, research questions, design stages, and empirical findings.

The case study of the AI Microservice Agent demonstrated how OWL-based service descriptors and SPARQL-driven discovery can improve service discovery efficiency, scalability, robustness, and cross-domain adaptability. The results showed that semantic registration reduced discovery latency by up to 35%, sustained stable throughput under increasing load, and maintained robustness with only minor reasoning overhead. Cross-domain validation further confirmed the framework's generalisability, achieving 100% successful integration across both text and image services.

Table 3 in the discussion highlighted the alignment between objectives, RQs, and findings, showing how methodological rigour was maintained throughout the research process. This alignment illustrates that DCM is not only a design approach but also a methodological scaffold for connecting research goals with measurable outcomes.

Practical Implications

For practitioners, the findings underscore the value of ontology-driven service registration in improving discovery efficiency and system scalability. The released artefacts (ontologies, service descriptors, containerisation scripts) provide a reproducible foundation for future AI microservice development, enabling others to replicate and extend the framework in real-world domains.

Limitations and Future Work

This study was limited to a controlled experimental setting with text classification and image clustering services as case domains. While these provide evidence of cross-domain adaptability, broader validation across additional AI tasks (e.g., speech processing, recommender systems, and multimodal services such as video analytics) are needed. Furthermore, the semantic reasoning layer introduced modest computational overhead (~8%), which suggests a trade-off between discovery precision and performance.

Future research will extend the framework along several directions: (1) scaling to enterprise-grade, multi-tenant deployments in cloud-native environments; (2) integrating AutoML techniques to optimise service selection and orchestration; (3) expanding validation to cover time-series, speech, and multimodal analytics; and (4) reinforcing open-science practices by releasing curated datasets, benchmarks, and artefacts to support transparency and comparative studies.

Future Work

Future research will extend the framework and its evaluation in several directions:

- Enterprise-scale deployments. Evaluate performance under large-scale, multi-tenant workloads typical of industrial cloud-native environments.
- Integration with AutoML. Incorporate automated machine learning techniques to optimise service selection and orchestration.
- Cross-domain scaling. Expand validation beyond text and image domains to include multimodal services such as time-series, speech, and video analytics.
- Open-science practices. Release all artefacts, datasets, and benchmarks in curated repositories to support reproducibility and comparative studies.

Ethical, Reproducibility, and Open-Science Checklist

Ethical considerations

No human subjects, personal data, or sensitive information were used in this study. The work is purely methodological and experimental, with no ethical risks identified.

Reproducibility.

The prototype was implemented using widely available, open-source frameworks: a Python Flask backend with REST APIs, a Bootstrap/HTML5 frontend, OWL ontologies with RDF/SPARQL queries for the semantic

layer, and Docker/Kubernetes for deployment and orchestration. These implementation details are further specified in the Implementation Note (Section 3.6), and all artefacts (ontologies, queries, deployment manifests, and source code) are released in a public repository: https://github.com/kingdavid1975/ai-microservices-agent.

Open-science practices.
The study design, methodology, and evaluation results are fully reported in the manuscript. To enable replication, we have released the complete artefacts — OWL ontology, SPARQL queries, representative service descriptors, and deployment scripts — in a public GitHub repository:
https://github.com/kingdavid1975/ai-microservices-agent The repository includes a README with step-by-step reproduction instructions.

Conflict of Interest
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.

Funding Statement
This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data and Code Availability
All implementation artefacts are openly available in a public GitHub repository: https://github.com/kingdavid1975/ai-microservices-agent The repository includes ontology files, SPARQL queries, representative service descriptors, and containerisation manifests (Docker/Kubernetes). A detailed README provides environment specifications and step-by-step instructions for reproducing the experiments reported in this paper.

References:

[1] A. Murphy and J. Moreland, "Integrating AI microservices into hard-real-time systems of systems to ensure trustworthiness of digital enterprise using mission engineering," *J. Integr. Design Process Sci.*, vol. 25, no. 1, pp. 38–54, 2022, doi: 10.3233/JID-210013.

[2] C. Li, "Decision analysis of system architecture in artificial intelligence cloud service model," *Eur. J. AI Comput. Inf.*, vol. 1, no. 2, pp. 7–13, 2025, doi: 10.71222/qtnwac26.

[3] B. Barua, M. Whaiduzzaman, M. Sarker, M. Kaiser, and A. Barros, "Designing and implementing a distributed database for microservices cloud-based online travel portal," in *Proc. Int. Conf. Big Data Analytics*, Singapore: Springer, 2023, pp. 295–314, doi: 10.1007/978-981-19-5443-6_22.

[4] K. Kalske, "Microservices architecture: Benefits, challenges and patterns," 2017.

[5] M. Fowler, "Microservices," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[6] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[7] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham, Switzerland: Springer, 2017, pp. 195–216, doi: 10.1007/978-3-319-67425-4_12

[8] S. S. Gill, "AI for next generation computing: Emerging trends and future directions," *Internet Things*, vol. 19, Art. no. 100514, 2022, doi: 10.1016/j.iot.2022.100514.

[9] M. Zaharia *et al.*, "Accelerating the machine learning lifecycle with MLflow," in *Proc. 4th USENIX Conf. Mach. Learn. Syst.*, 2018.

[10] Ontotext, "What is a knowledge graph?" 2022. [Online]. Available: https://www.ontotext.com/knowledgehub/

11] K. Baclawski *et al.*, "Ontology summit 2020 communiqué: Knowledge graphs," *Appl. Ontol.*, vol. 16, no. 2, pp. 229–247, 2021, doi: 10.3233/AO-210249.

[12] S. Gregor and A. R. Hevner, "Positioning and presenting design science research for maximum impact," *MIS Q.*, vol. 37, no. 2, pp. 337–355, 2013, doi: 10.25300/MISQ/2013/37.2.01.

[13] A. Hevner, "A three-cycle view of design science research," *Scand. J. Inf. Syst.*, vol. 19, no. 2, pp. 87–92, 2007.

[14] A. Hogan *et al.*, "Knowledge graphs," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–37, 2021, doi: 10.1145/3447772.

[15] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, 2019, doi: 10.1016/j.jss.2019.01.001.

[16] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proc. IEEE Int. Conf. Service-Oriented Computing and Applications (SOCA)*, 2016, pp. 44–51, doi: 10.1109/SOCA.2016.15.

[17] A. Alexandru, C. A. Alexandru, D. Coardos, and E. Tudora, "Healthcare, big data and cloud computing," *WSEAS Trans. Comput. Res.*, vol. 4, no. 1, pp. 123–131, 2016.

[18] Z. Liu *et al.*, "On the replicability and reproducibility of deep learning in software engineering," *arXiv preprint*, arXiv:2006.14244, 2020.

[19] P. Salerno, "Microservices design patterns for cloud architecture," IEEE Chicago Section, Conf. Presentation, Sep. 25, 2024.

[20] H. Takeda, P. Veerkamp, and H. Yoshikawa, "Modeling design process," *AI Mag.*, vol. 11, no. 4, pp. 37–48, 1990, doi: 10.1609/aimag.v11i4.855.