# Investigating Neurosymbolic AI for Intent-based Service Management

Lorenzo Colombi*, Sara Cavicchi*, Filippo Poltronieri*,
Mauro Tortonesi*, Cesare Stefanelli*, and Pal Varga†
* University of Ferrara, Ferrara, Italy
Email: {firstname.lastname}@unife.it,
† Budapest University of Technology and Economics, Budapest, Hungary
Email: pvarga@tmit.bme.hu

***Abstract*—The increasing complexity and increasing demands of IT applications, especially in federated multi-cluster environments, pose significant challenges for service orchestration. To address these, Zero-Touch Service Management (ZSM) and intent-based management paradigms are gaining traction, allowing users to specify high-level goals rather than low-level configurations. However, current intent-driven approaches often rely on rigid Domain Specific Languages (DSLs) or graphic user interfaces, limiting expressiveness and usability. In this work, we propose a neurosymbolic intent-based platform that leverages Large Language Models (LLMs) for natural language intent ingestion and Answer Set Programming (ASP), a declarative programming paradigm used for solving complex combinatorial problems. The system translates natural language descriptions of microservice requirements into structured policies, enabling explainable service-to-cluster matching across federated Kubernetes environments. We validate our approach through experiments that evaluate both the syntactic correctness and efficiency of various LLMs in intent translation, as well as the computational time of the symbolic placement algorithm.**

***Index Terms*—Intent-based Service Management, Large Language Model, Neurosymbolic AI**

## I. Introduction

The rising performance demands of IT applications, including Artificial Intelligence (AI)-based ones, which are becoming even more popular also in safety-critical environments, such as Industry 5.0, [1], [2] pose increasing challenges for service orchestration and infrastructure management. These challenges have fueled the shift toward Zero-touch network and Service Management (ZSM), a paradigm aimed at fully automating network and service lifecycle operations. ZSM promotes vendor-neutral, closed-loop automation systems capable of performing tasks such as self-configuration, self-healing, and self-optimization.

An enabler of ZSM is intent-based management, which allows users to specify high-level operational goals rather than detailed configurations. This significantly lowers the cognitive load on operators and better aligns service management with business objectives [3]. Despite progress in intent-based approaches, current systems often rely on rigid domain-specific languages or static templates, which could limit expressiveness or require technical expertise. While some systems support natural language input, this introduces new complexities due to the inherent ambiguity and contextual dependencies of human language. In this regard, the rapid evolution of Large Language Models (LLMs) has opened up new possibilities for bridging the gap between natural language intent expression and machine-actionable policies [4], [5].

At the same time, despite the presence of a unified API-driven infrastructure that spans cloud, edge, and fog layer, the so-called compute continuum, existing orchestration systems, including Kubernetes, are primarily designed for individual cluster management and lack robust mechanisms for automated, scalable orchestration across federated multi-cluster environments, a growing requirement in today's distributed computing landscape [6].

To address these limitations, it is even more common to use AI, including reinforcement learning [7] and neural networks in service management. However, while neural network-based approaches offer adaptability and the ability to learn complex decisions, they also introduce notable limitations. Specifically, they often lack deterministic behavior and function largely as black boxes, making it difficult to interpret or explain the rationale behind their decisions. These drawbacks motivate the exploration of alternative or complementary AI techniques that can offer greater transparency and control, such as symbolic reasoning or hybrid AI approaches. In particular, symbolic AI has demonstrated value for reasoning over structured knowledge bases and enforcing policies in explainable and deterministic ways [8]. Symbolic AI systems have also been used for intent modeling and conflict resolution [9]. Meanwhile, the integration of symbolic AI with neural models offers a promising path forward, combining the reasoning capabilities of symbolic systems with the adaptability of neural networks.

To this end, neurosymbolic AI – the integration of neural learning methods with symbolic reasoning – can be a valuable solution to enable more flexible and explainable decision-making. Therefore, in this work, we present the initial design of a neurosymbolic [10] intent-based service management platform for federated multi-cluster environments. Our goal is to enable operators to specify the constraints, including resource ones and dependencies between each other, for each microservice of an application using natural language and to automatically derive valid, explainable, and optimized service placements across many Kubernetes (K8s) clusters. By combining LLM-powered natural language processing with

978-3-903176-75-1 

symbolic logic reasoning, specifically using the Answer Set Programming (ASP) framework, we aim to support intelligent orchestration across the compute continuum. In our initial implementation, we used an LLM to convert the natural language input into a JSON representation. This JSON is subsequently used to create a symbolic representation of the problem using ASP. Lastly, the Clingo solver is used to extract all the possible service-to-cluster matching solutions.

We conducted a series of experiments to validate the solutions, focusing on the intent translation and the ASP-based cluster filtering algorithm. Regarding the intent translation, the evaluation tested several open-source LLMs for translating natural language intents into valid JSON. Models were assessed under one-shot and few-shot settings, using syntax validity and inference time as metrics. On the other hand, for the moment, the validation of the ASP component has been limited to the measurement of the computation time to compute all the possible service-to-cluster matching solutions.

## II. Background and Related Work

The advent of demanding applications, characterized by high dynamicity and diverse Quality of Service (QoS) requirements, has increased the complexity of service management, necessitating a shift toward ZSM paradigms. ZSM is a framework designed to enable fully automated, zero-touch management of networks and services, regardless of vendor dependencies. Its architecture provides flexible, vendor-neutral management services, aligning with industry trends that move away from traditional, rigid management systems [11]. A key feature of ZSM is closed-loop automation, which enables networks to achieve and maintain operational goals without human intervention [3]. The transition to management automation can enhance the flexibility and efficiency of service delivery while lowering Operating Expenses (OPEX) by enabling self-managing capabilities such as self-configuration, self-healing, self-optimization, and self-protection [11].

Intent-driven management is a key enabler of automation, enhancing autonomous network and service operations within the ZSM framework [12]. It simplifies automation by enabling high-level, outcome-focused expressions of management objectives, allowing systems to determine optimal solutions and adapt in real-time [13]. In [12], authors propose, as the formal definition of intent, the "TM Forum IG1230" exploratory report one: *"Intent is the formal specification of the expectations, including requirements, goals, and constraints, given to a technical system"*

Following [14], the first step in intent processing is *profiling* [14], also known as *ingestion* [13], in which users interact with the system to express their intent. Intent statements are typically defined through high-level declarative formats, including JSON and YAML files, specific intent-based languages, Graphic User Interfaces (GUIs), and natural language. In literature, one can find many intent-based languages like Nile or NEtworking MOdeling (NEMO), which function as programming languages designed to reduce the gap between human and machine readability [14]. Although this approach aims to simplify intent declaration and abstract technical details, its structure remains quite rigid and requires technical expertise to understand, define, and modify intents effectively.

Another method for expressing intent statements is through a GUI. However, this approach limits users from specifying high-level goals that require options beyond those provided by the interface. On the other hand, natural language intents—while flexible and user-friendly—face significant challenges due to the ambiguity of human language, including context-dependent meanings, linguistic errors, and subtle phrasing. The diversity of languages, along with their varying grammar and lack of standardization, further complicates the translation of user intents into executable machine actions.

Once intent has been defined, it needs to be translated into low-level policies and machine-executable actions [15]. This phase is commonly referred to as *intent translation* [13]. According to the type and scope of the statement, different translation mechanisms can be employed [14]. The simplest approach consists of associating intent statements with ready-to-use templates. This method is commonly adopted alongside GUI-based intent ingestion, where the selected options can be easily translated into policies. However, since these templates are predefined, they inherently limit the range of possible resulting configurations and require expertise for their definition.

Another recently emerging strategy involves leveraging LLMs to translate intent statements expressed in natural language [16], [17]. The rapid rise of LLMs in the Natural Language Processing (NLP) field is prompting the research community to explore their potential for standardizing intent representation in network and service management [18]. LLMs outperform previous architectures like Long Short-Term Memory networks thanks to the adoption of the self-attention mechanism, which enables them to effectively capture long-range dependencies and process information in parallel, eliminating the bottleneck of sequential processing [19]. LLMs not only outperform previous models in natural language processing but have also demonstrated emergent abilities, including *in-context learning* (also known as few-shot learning), where LLMs learn a new task from a small set of input-output pairs provided in the prompt at inference time; *instruction following*, where instruction-tuned LLMs can perform new tasks without requiring explicit examples; and *multi-step reasoning*, where LLMs solve complex problems by breaking them down into intermediate steps [4]. These remarkable capabilities in knowledge comprehension, generalization, and adaptation could further enhance intent profiling and translation, particularly for sophisticated goals or dynamic environments.

Subsequently, once the intent has been translated into machine-readable policies, which could be single rules or rule sets, the next phase, known as *intent resolution* [14], verifies that the expressed intent does not conflict with statements specified either previously or simultaneously, attempting to solve any inconsistencies that may arise.

Next comes *intent activation* [14], which allocates, schedules, and deploys the required resources to satisfy the intent across the underlying infrastructure. However, while robust

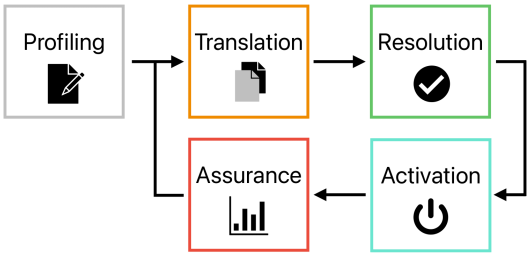


Fig. 1: Intent processing steps

scheduling and autoscaling mechanisms are well-established at the individual cluster level, reliable and fully automated approaches for scheduling, scaling, and service migration across large-scale federated cluster environments have yet to achieve production-grade maturity [6]. Noteworthy, these multi-cluster settings are becoming more popular; therefore, the need for a multi-cluster management solution arises.

A possible solution could be using an external decision-making tool with access to the information of all the federated clusters. This external tool could employ neurosymbolic AI, combining the adaptability and complex decision-making capabilities of neural network-based approaches with a symbolic reasoner to mitigate their limitations. One of the main drawbacks of neural approaches lies in their lack of deterministic behavior: given the same input, outputs may vary depending on internal states or minor perturbations, which complicates system predictability and reproducibility. In addition, neural networks function largely as "black boxes," making it difficult to interpret or explain the rationale behind their decisions.

Symbolic AI can address this issue by reasoning and performing decision-making based on user-defined policies and a domain-specific knowledge base, which includes the system's current state—for instance, resource utilization, hardware capabilities, cost, Service Level Agreement (SLA) level, and latency. Symbolic AI offers key advantages such as explainability, efficient handling of structured knowledge, low data dependency, consistency, and ease of debugging. These strengths also make it well-suited for critical environments.

Despite not being specifically designed for a multi-cluster setting, in literature, several works apply symbolic AI to service management. For instance, in [8], authors used the open-source Prolog language to model intent statements in an intent-based networking setting in a declarative manner, while in [9], Prolog is used to perform intent conflict management.

The final phase is *intent assurance*, which is concerned with ensuring that the network continues to comply with the fulfilled intent throughout its operational lifetime.

The five steps of intent processing are shown in Fig. 1. This sequence constitutes a closed loop, which is essential for enabling dynamic and autonomous network and service orchestration [14].

## III. NeSy intent-based service placement solution

This work presents an initial design of an intent-based service management platform. The proposed solution aims to facilitate natural language interactions between human operators and service infrastructures in a multi-cluster environment. However, it is conceived as a work in progress, open to future enhancements and adaptations in response to evolving requirements and technological advancements. Our goal is to design a neurosymbolic solution to support multi-cluster service management, enabling intelligent, dynamic orchestration across the compute continuum, integrating neural network-based and symbolic AI techniques, such as LLMs and logic programming.

For now, our focus is limited to the application deployment in a federated multi-cluster scenario, where each application is composed of a set of microservices. We modelled each microservice with an ID and a set of requirements and dependencies. Requirements include available resources (e.g., CPU, RAM) and hardware capabilities (e.g., GPU), and cluster characteristics (e.g., SLA level, costs). At the same time, dependencies represent a list of other microservices that are needed for the full functionality of the application. For simplicity, we assume that intra-cluster latency and connectivity are not an issue, whereas inter-cluster communication may experience disconnections and limited bandwidth. For this reason, strongly coupled microservices have to be deployed in the same cluster. These applications are represented as a Directed Acyclical Graph (DAG), where each node, with its properties, represents a microservice and the edges represent the "depends on" relation.

Our proposed solution is illustrated in Fig. 2, which leverages distinct components to manage the full life cycle of intent statements—from their initial definition to the deployment of the specified requirements over the infrastructure. This modular architecture allows for better maintainability, scalability, and the potential reuse of individual components. In the first stage of the proposed pipeline, the user interacts with the system to declare their intent. The statement, expressed in natural language for greater flexibility and ease of use, is processed by the intent ingestion component, powered by an LLM, which collaborates with the intent translator to assess whether the intent is sufficiently clear and well-defined or requires further refinement. In case of ambiguity, the user is asked to provide the missing details necessary for a correct translation. The decision to implement the ingestion component using LLMs is motivated by their strong capabilities in natural language understanding and semantic information extraction. Moreover, these models can engage in real dialogues with the user, effectively guiding them through the formulation of precise and unambiguous requirements. Once the statement is considered well-specified, the intent translator converts it into a machine-readable format, such as a JSON file, for further processing. JSON or YAML format could be chosen due to their ease of validation and their suitability for both human readability and machine processing. Moreover, leveraging this intermediate representation decouples the translation process from the following stages, allowing eventual substitutions of single components. It is worth noting that both the ingestion and the translation components could be implemented using

```
resources(C_id, T_cpu, T_ram) :- cluster(C_id,
    C_cpu, C_ram), T_cpu = #sum{S_cpu, C_id:
    match(S_id, C_id), service(S_id, S_cpu, _)
    }, T_ram = #sum{S_ram, Cluster_id: match(
    S_id, C_id), service(S_id, _, S_ram)}.
```

Listing 1: Total resources used per cluster in ASP

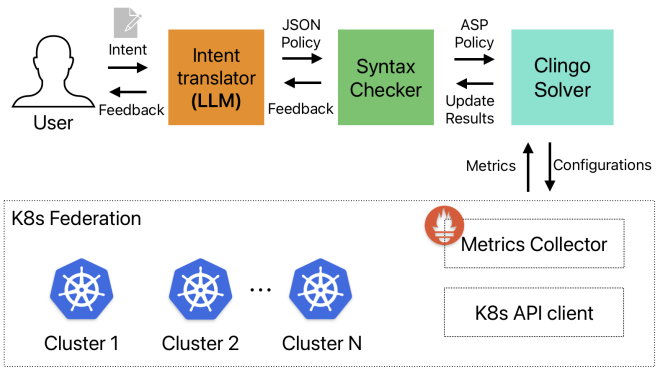


Fig. 2: NeuroSymbolic Intent-based Service Management System Architecture

a single LLM. This LLM first receives and parses the intent, and if the statement is clear, proceeds with the translation.

Subsequently, a Syntax Checker module, illustrated in Fig. 2 analyzes the generated JSON to ensure that its structure is syntactically correct. If a validation error is detected, the translation component—the LLM—is notified and prompted to retry the conversion. The prompt is augmented with the specific syntax error, effectively enabling a few-shot learning setup to improve correction. If the output remains invalid after a predefined number of retries, the user is alerted to manually intervene.

Once the JSON or YAML file has been successfully validated, it is passed to a conversion module that translates it into a set of executable low-level policies. This translation is done by mapping the structured representation into a corresponding set of ASP, a rule-based language for knowledge representation and nonmonotonic reasoning, developed within the field of logic programming, rules [20], [21]. ASP solvers compute all the stable models (also known as answer sets), which represent the set of literals that are true according to the program's rules. In our case, the stable models represent all the possible service-to-cluster matching solutions.

ASP was chosen due to its expressive power and because the symbolic reasoning with ASP is deterministic and inherently explainable, providing transparency and reliability in the decision-making process. A comprehensive tool for solving ASP programs is Clingo [1], which combines Gringo, a grounder that assigns concrete domain values to variables, and Clasp, the solver that generates the final answer sets. Moreover, a Python module is available, providing functions and classes to control and execute the grounding and solving processes, facilitating the integration of neural network–based decision-making tools with symbolic reasoning. However, at present, the use of ASP and Clingo is limited to cluster filtering; thus, the output consists of a list of service placement solutions that satisfy the required constraints.

An example of a simplified version of the "resources" clause, which is used to calculate for each cluster the total resources consumed by the services deployed there, could be written in ASP as:

This rule states that a service, identified by its *service_id*, can be matched with a cluster *cluster_id* if the cluster's available CPU, RAM, and storage are greater than or equal to the corresponding requirements of the service. In other words, the rule checks that the cluster has enough resources to host the service, ensuring that all constraints are satisfied before a match is made. However, more complex rules and logical reasoning could be implemented.

Cluster metrics, on which the decision-making process is based, are collected using Prometheus, which is configured to scrape data from all clusters. Each cluster runs its own Prometheus instance in federation mode, enabling it to gather metrics from all nodes within the cluster and expose them for external scraping.

Despite being beyond the scope of this work, once the possible solution has been computed, the optimal one has to be found. This is also possible by implementing a sorting algorithm in ASP, but a more advanced solution could be more suitable for this task. These include, for example, the use of Multi-Criteria Decision Making (MCDM) algorithm or AI-based methods, such as reinforcement learning [7].

Lastly, once the more suitable cluster for each microservice has been found, the K8s Application Programming Interface (API) client interfaces with the clusters deploying the service a leaving the node choice to the Kube scheduler of each cluster.

## IV. Experimental Evaluation

We conducted a series of experiments to validate the proposed approach, separately testing the two main components of the neurosymbolic architecture: the LLM used to translate user intent and the ASP, which implements the symbolic part, solver used to compute all the possible service placement solutions. The experimentation setup is summarized in Table I.

TABLE I: Summary of Experimental Setups

| Specification | LLM Intent Translation | ASP Experiement |
|---|---|---|
| Operating System | Linux (CentOS) | Linux (Ubuntu) |
| RAM | 32GB DDR4 | 32GB DDR4 |
| CPU | AMD EPYC 9224 | Intel Core i7-9750H |
| GPU | Nvidia L40S | Nvidia RTX 3060 |

### A. LLM Intent Translation

The goal of this experiment was to measure the capabilities of different open source LLMs in generating a JSON with

[1] https://potassco.org/clingo/

correct syntax and complying with a predefined schema when prompted with an intent expressed in natural language.

Deepseek-R1 with 14 billion parameters, Gemma3 with 12 and 27 billion parameters, qwen2.5-coder with 7 billion parameters, and qwen3 with 32 and 8 billion parameters. These models were selected due to their performance despite having relatively fewer parameters compared to other LLMs. This makes them particularly well-suited for deployment in real-world industrial scenarios, where computational resources may be limited and heavier models are impractical.

To simulate operator interactions, we prompted the LLMs with natural language inputs alongside a predefined JSON template and a series of illustrative examples. We formatted the prompt using a Markdown syntax, using unordered lists and headings to separate the different parts. However, other formats, including YAML, could and have been used in the literature. The expected JSON output was required to include a set of properties necessary for policy creation, such as resource constraints and a target objective (e.g., minimizing latency).

Our evaluation focused on two primary metrics: inference time and response accuracy. However, we limited our validation to syntactic correctness due to the challenges in assessing the semantic alignment between the operator's intent and the generated JSON policy. Specifically, an output was deemed valid if it could be successfully deserialized into a JSON object, contained all the required keys (e.g., serviceID, depends_on, constraints), and assigned appropriate data types and legal values (e.g., the CPU constraint has to be a positive float) to each field.

We tested each LLM in two different conditions. In the One-Shot (1-S) setting, the model was prompted once with at least one example of intent translation at the end of the prompt, and we measured the percentage of outputs containing syntactically valid JSON. In the Few-Shot (F-S) setting, the model was allowed multiple attempts: if the initial output was invalid, the prompt was augmented with the corresponding error message and resubmitted by the check component, measuring the success rate across these corrected iterations. Specifically, as presented in Table II, we limited the number of requests, or shots, to the LLM to 3. In this condition, we also measured the average minimum number of prompts to obtain a valid JSON.

Experiment results are presented in Table II and shows how there is a great variability between different models. In detail, DeepSeek-R1-Distill-Qwen-14B-Q_K and qwen2.5-coder-7b-instruct-q6_k show the highest performance in the 1-S accuracy, which also improves in the successive shots. However, the highest accuracy has been obtained from the second shot using google_gemma-3-27b-it-Q6_K, which also presents one of the lowest mean inference times, despite its number of parameters and mean prompts to get a correct response, making it the most suitable model for this kind of experiment. Notably, the "reasoning" model, such as Qwen, presents a notable increase in inference time but without a corresponding increase in accuracy.

### B. ASP experiments

In this experiment, we measured the execution time of Clingo, an ASP solver, to compute all possible stable models, or, in other words, the service-to-cluster matching solutions that satisfy the specified constraints. The goal was to assess whether the Clingo execution time was suitable for a real-world scenario.

We randomly generated a series of clusters and services with their respective constraints using Python and the Clingo Python module. Each service was identified by a unique ID and characterized by its required amounts of CPU, RAM, and storage, as well as whether it needed a node with a GPU. Similarly, for each cluster, in addition to its ID, we specified the available amounts of CPU, RAM, and storage, along with the number of nodes equipped with a GPU. It should be noted that these features can be modified or extended depending on the specific use case, individual considerations, or system objectives. We varied the number of clusters from 2 to 30, and for each cluster count, we generated 10 scenarios. In these scenarios, the constraints for each service were randomly generated, allowing us to simulate the deployment of a simple web application composed of 4 microservices. Subsequently, we calculated the average computation time required to solve each scenario. Exploiting Clingo's parallelization capabilities, we compared the ASP execution time using a single thread and four threads.

The results, reported in Fig. 3, show that the computational time increases as the number of clusters grows, with some oscillations due to the randomness in the characteristics of the added services and clusters. This is due to the increasing number of potential service-to-cluster combinations, which are at maximum, in non-constrained conditions $\text{number\_of\_clusters}^{\text{number\_of\_services}}$. The required time to compute the matchings significantly depends on both the characteristics of the clusters and the resource demands of the services: the more resources available in the clusters and the fewer resources required by the services, the greater the number of feasible combinations. We can notice how the parallel version is notably faster. However, the execution time is below an acceptable threshold, especially considering the intent translation time, even with a large number of clusters (e.g., 20).

## V. Conclusions

We demonstrated the feasibility of using neuro-symbolic AI [10], combining open-source LLMs with logic programming via ASP. In this approach, LLMs are employed for intent profiling and translation into structured JSON representations, while ASP is used to compute all feasible service-to-cluster matching solutions. Experimental evaluations revealed that even lightweight LLMs provide a strong balance between performance and computational efficiency, making them suitable for real-world applications.

However, our solution is still in an initial phase of development, and many real-world challenges remain to be addressed, such as handling ambiguous or conflicting user intents from

| Model | 1-s Acc. | 2-s Acc. | 3-s Acc. | Mean #Prompts | Inference Time (s) |
|---|---|---|---|---|---|
| DeepSeek-R1-Distill-Qwen-14B-Q6_K | 0.48 | 0.83 | 0.83 | 1.42 | 16.74 |
| google_gemma-3-27b-it-Q6_K | 0.06 | 1.00 | 1.00 | 1.94 | 2.99 |
| google_gemma-3-12b-it-Q6_K | 0.00 | 0.67 | 0.67 | 2.00 | 1.50 |
| Meta-Llama-3.1-8B-Instruct-Q6_K_L | 0.16 | 0.66 | 0.67 | 1.77 | 0.95 |
| qwen3-8b-q6_k | 0.03 | 0.56 | 0.66 | 2.12 | 63.84 |
| Qwen3-32B-Q6_K | 0.00 | 0.00 | 0.72 | 2.21 | 87.50 |
| qwen2.5-coder-7b-instruct-q6_k | 0.39 | 0.83 | 0.83 | 1.53 | 0.97 |

TABLE II: Model accuracy at different shot levels, average prompts to get valid JSON (**Mean #Prompts**), and **Inference Time**.

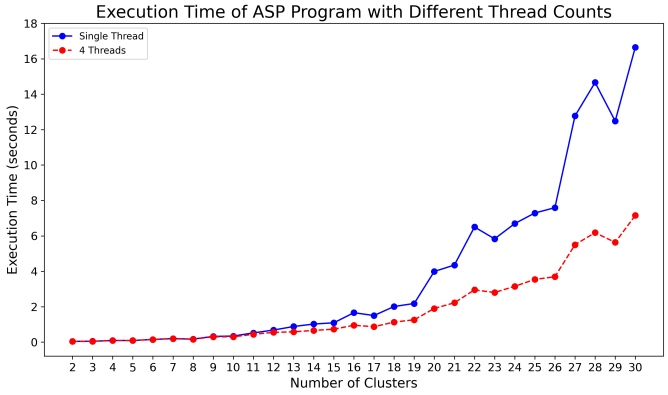


Fig. 3: Service-to-Cluster ASP-based filtering execution time with one and four threads.

different users. To solve these challenges, the Scallop-based filtering algorithm could also be extended to node selection decisions and, above all, used in combination with a solution sorting algorithm.

Future work will focus on extending the symbolic reasoning capabilities to support, for instance, intent assurance [22] and more complex deployment policies, including multi-objective optimization and temporal constraints. Additionally, we plan to integrate advanced decision-making approaches, such as reinforcement learning [7] or MCDM algorithms. Further development will include real-time orchestration support in environments with intermittent connectivity or limited resources, as well as deeper integration of neural and symbolic components for a more adaptive and intelligent management system.

## References


[1] L. Colombi *et al.*, "Embedding models for multivariate time series anomaly detection in industry 5.0," *Data Science and Engineering*, pp. 1–17, 2025.

[2] S. Dahdal *et al.*, "An mlops framework for gan-based fault detection in bonfiglioli's evo plant," *Infocommunications Journal*, vol. 16, no. 2, pp. 2–10, 2024. doi: 10.36244/ICJ.2024.2.1

[3] M. Liyanage *et al.*, "A survey on zero touch network and service management (zsm) for 5g and beyond networks," *Journal of Network and Computer Applications*, vol. 203, p. 103362, 2022. doi: https://doi.org/10.1016/j.jnca.2022.103362. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522000297

[4] S. Minaee *et al.*, "Large language models: A survey," *arXiv preprint*, vol. arXiv:2402.06196, 2024. [Online]. Available: https://arxiv.org/abs/2402.06196

[5] G. Hollósi *et al.*, "Generative ai for low-level netconf configuration in network management based on yang models," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–7.

[6] M. Tortonesi, "The compute continuum: Trends and challenges," *Computer*, vol. 58, no. 3, pp. 105–108, 2025. doi: 10.1109/MC.2024.3520255

[7] J. Santos *et al.*, "Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*. IEEE, 2024, pp. 1–9.

[8] J. Massa *et al.*, "Declarative provisioning of virtual network function chains in intent-based networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023. doi: 10.1109/NetSoft57336.2023.10175449 pp. 522–527.

[9] ——, "A declarative reasoning approach to conflict management in intent-based networking," in *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2024. doi: 10.1109/ICIN60470.2024.10494474 pp. 228–233.

[10] L. De Smet *et al.*, "Defining neurosymbolic ai," *arXiv preprint arXiv:2507.11127*, 2025.

[11] C. Benzaid *et al.*, "Ai-driven zero touch network and service management in 5g and beyond: Challenges and research directions," *Ieee Network*, vol. 34, no. 2, pp. 186–194, 2020.

[12] ETSI, "Zero-touch network and Service Management (ZSM); Intent-driven autonomous networks;Generic aspects," European Telecommunications Standards Institute (ETSI), Tech. Rep. ETSI GR ZSM 011 V1.1.1, December 2024. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/018/01.01.01_60/gs_ZSM018v010101p.pdf

[13] A. Clemm *et al.*, "Intent-Based Networking - Concepts and Definitions," RFC 9315, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9315

[14] A. Leivadeas *et al.*, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2023. doi: 10.1109/COMST.2022.3215919

[15] K. Dzeparoska *et al.*, "Towards a self-driving management system for the automated realization of intents," *IEEE Access*, vol. 9, pp. 159 882–159 907, 2021. doi: 10.1109/ACCESS.2021.3129990

[16] ——, "Llm-based policy generation for intent-based management of applications," in *2023 19th International Conference on Network and Service Management (CNSM)*. IEEE, Oct. 2023. doi: 10.23919/cnsm59352.2023.10327837 p. 1–7. [Online]. Available: http://dx.doi.org/10.23919/CNSM59352.2023.10327837

[17] A. Mekrache *et al.*, "Intent-based management of next-generation networks: an llm-centric approach," *IEEE Network*, vol. 38, no. 5, pp. 29–36, 2024. doi: 10.1109/MNET.2024.3420120

[18] E. Karlsen *et al.*, "Benchmarking large language models for log analysis, security, and interpretation," *Journal of Network and Systems Management*, vol. 32, no. 3, p. 59, 2024.

[19] A. Vaswani *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[20] V. Lifschitz, *Answer set programming*. Springer Cham, 2019, vol. 3.

[21] A. Bertagnon *et al.*, "Fine-grained timing analysis of digital integrated circuits in answer set programming," *arXiv preprint arXiv:2507.11150*, 2025.

[22] K. Dzeparoska *et al.*, "Intent assurance using llms guided by intent drift," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024. doi: 10.1109/NOMS59830.2024.10575429 pp. 1–7.