A Perspective on LLM Data Generation with Few-shot Examples: from Intent to Kubernetes Manifest

Antonino Angi[1,2], Liubov Nedoshivina[2], Alessio Sacco[1], Stefano Braghin[2], Mark Purcell[2]

[1] Department of Control and Computer Engineering, Politecnico di Torino, Italy
[2] IBM Research, Dublin, Ireland

Correspondence: antonino.angi@polito.it

Abstract

The advent of Large Language Models (LLMs) has transformed how complex tasks across various domains can be automated. One of the industry trends today is Agentic AI, which leverages LLMs to operate multiple tools and provide automatic configuration. In the domain of cloud computing, Agentic AI might be used, for example, with the generation of Kubernetes manifests – structured configuration files that define containerized environments. However, effectively applying LLMs to domain-specific tasks often reveals knowledge gaps that impact the accuracy and reliability of the generated output.

To address these challenges, we propose *KGen*, a pipeline for generating K8s manifests directly from user-described intents expressed in natural language using LLMs. Our approach leverages an extensive *n*-shot learning analysis to choose the appropriate number of examples that can better guide the adopted models in generating the manuscripts while also looking at the computational cost. Our results validate the use of LLM in this task and show that (as expected) increasing the number of *n*-shot examples can improve the quality of the generated configurations when adopting more specialized models, such as `Mixtral-8x7B` (which uses the Mixture of Experts approach) and `Prometheus-8x7B-v2.0`, but (surprisingly) for more general-purpose models like `Llama3-8B` and `Llama3-70B`, it can lead to smaller number of valid K8s manifests. These results underscore the complexities of adapting LLMs for domain-specific structured generation and emphasize the need for an in-depth analysis to determine the most effective setup, also suggesting that smaller models sometimes outperform their larger counterparts for each domain-specific task.

1 Introduction

Traditional cloud computing operations often involve complex manual configurations, particularly in service deployment of containerized environments (*e.g.,* Kubernetes, microservices), where tasks like defining network policies and services require significant expertise and can be challenging for less-experienced users. In response, intent-based networking (IBN), often powered by large language models (LLMs), has emerged as a promising approach (Kratzke and Drews, 2024; Xu et al., 2024). By translating high-level intents expressed in natural language into Kubernetes (K8s) manifests (structured configuration files as exemplified in Figure 1), this approach has the potential to simplify configuration tasks, make them more accessible, and speed up the deployment of network and application configurations.

```yaml
apiVersion: apps/v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:latest
```

Figure 1: Example of a minimal Kubernetes manifest for an `nginx` image Pod deployment.

Recent examples demonstrated the advance of applying LLM-based AI Agents for AIOps (Artificial Intelligence for IT operations) in general (Vitui and Chen, 2025; Chen et al., 2025) and for Kubernetes tasks in particular (Kubiya.ai, 2025; Logz, 2025; kagent, 2025) serving LLMs as core components capable of reasoning. While LLMs have shown versatility across different domains (Ge et al., 2024; Ling et al., 2023), there are techniques, such as few-shot prompting or fine-tuning, that can make LLMs domain-specific and improve their generation accuracy. The first technique leverages customized prompts to guide the model toward more accurate outputs without requiring additional training, making it significantly more computationally efficient and requiring no specialized hardware.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 345–354

Although it may involve prompt refinement and human intervention (Kratzke and Drews, 2024), it remains a faster, more scalable, and automated alternative compared to fine-tuning, which demands extensive GPU resources and training epochs.

In this paper, we introduce *KGen* (**K**ubernetes Manifest **Gen**eration), a pipeline that fine-tunes LLMs to more accurately generate K8s manifests directly from natural language intents, coming, for example, from an end-user or another LLM if in an AI Agent setting. We performed an in-depth *n*-shot learning analysis across multiple LLMs, critically evaluating their effectiveness when dealing with production-like files.

In KGen, we start by generating a dataset of K8s manifests, which were fed into different LLMs (*i.e.,* `Mixtral-8x7B` (MixtralAI, 2025), `Prometheus-8x7B-v2.0` (Prometheus, 2024; Kim et al., 2023), `Llama3-8B` (Meta/Llama, 2025b), and `Llama3-70B` (Meta/Llama, 2025a)) to produce corresponding descriptions (or intents from now on) using an increasing number *n* of few-shot examples. To evaluate the quality of generated intents, we then asked the same adopted LLMs to re-generate the manifests from the intents using the same number of contextual examples. This process resulted in a dataset of reconstructed manifests, which we were able to first validate for structural validness (YAML syntax) and then compare against the original manifests to assess the accuracy of human language translation (intent semantic).

Our experiments validated the accuracy of the process but also revealed that the number of examples provided for *n*-shot learning has a significant and complex impact on the quality of the generated manifests. On the one hand, a few examples for `Mixtral-8x7B` or `Prometheus-8x7B-v2.0` led to under-performance, as both models lacked sufficient context to generate accurate structured output. On the other hand, for `Llama3` models, a high number of examples can mislead the model and result in worse accuracy while also introducing additional computational overhead and increasing input tokens usage – critical factors in real-world deployment scenarios. This outcome is likely due to the heterogeneity and non-trivial aspects of the structured files as the K8s manifests (Xu et al., 2024) and highlights the necessity of careful model evaluation to determine the optimal number of examples that balances computational efficiency and accuracy while maintaining reliable performance in production-scale applications.

2 Related Work

In the era of Generative AI and Large Language Models (LLMs), many studies have explored the integration of these advanced models to generate network configurations (Zhou et al., 2024a). One of the implementations (Dzeparoska et al., 2023) involves an LLM-based architecture composed of pipelines to translate intents into network policies using a progressive intent decomposition process. Similar work (Fuad et al., 2024) demonstrates a framework to translate intents, specified in natural language, to network configurations adapted for a Border Gateway Protocol (BGP) routing protocol using different LLMs. However, the authors do not investigate the hallucination problem that is common when working with LLMs and could impact the overall model's performance.

While LLMs are powerful and versatile, their adaptability across domains can result in decreased performance when applied to specific tasks (Xiao et al., 2024; Zhang et al., 2024; Huang et al., 2024). For this reason, researchers have begun to integrate techniques, known as *prompting*, into their solutions to better guide the model and produce more adapted responses. An example (Lin et al., 2023) presents Appleseed, an intent-based system to train an LLM using few-shot examples with the goal of generating a set of executable Python programs that can be adapted to different use cases. Similarly, an intent extraction solution focused on 5G networks employs a customized LLM with prompting techniques (Manias et al., 2024).

Moving towards an intersection of LLMs with Intent-based Networking for cloud-native scenarios, researchers have also focused on generating structured cloud configurations (*e.g.,* in YAML and JSON) used to automate service deployment across distributed infrastructures. The adoption of these configurations has shown flawless integration in containerized environments, microservices, and Kubernetes-based clusters or Ansible-based automation tools (Pujar et al., 2023). An example of this integration presents a benchmark (Xu et al., 2024) that was tested on a hand-crafted dataset using different LLMs. In another work (Mekrache et al., 2024) the authors propose an architecture for decomposing the intents into their Cloud/Edge and RAN elements. A competing approach for generating Kubernetes manifests (Kratzke and Drews, 2024) employs custom prompts and various LLMs. However, this work shows that manual interven-

tion might be needed for some LLMs to refine the generated manifests, which in the long term could slow the generation process, especially in long-structured manifests.

3 KGen Overview and Components

In this section, we explore the steps that compose KGen (see Figure 2 for an overview). As mentioned previously, the primary goal of KGen is to adapt the few-shot learning strategy to enable an LLM, either standalone or as a core of an AI Agent, to generate Kubernetes manifest from natural language intent. As the first step to perform the few-shot (or *n*-shot), we begin by describing the process of collecting the Kubernetes Manifests Dataset from a subset of industry examples (see Section 3.1). This dataset is then used by different LLMs (*i.e.,* `Llama3-70B`, `Llama3-8B`, `Prometheus-8x7B-v2.0`, and `Mixtral-8x7B`) as source of examples for *n*-shot to generate descriptive summaries of the manifests, *i.e.*, intents (Section 3.2). Then, the LLMs are tasked with regenerating the original manifests based on the descriptions (Section 3.3).

To evaluate whether the generated intents are sufficiently descriptive to provide adequate context for generating valid manifests, the generated manifests are compared with the initial manifests. As part of the extensive evaluation of KGen's consistency, we also conducted a cross-check: for each LLM from the list above, we applied few-shot learning to one model to generate manifest-to-intent pairs and prompted another model to re-generate the manifest from the intent.

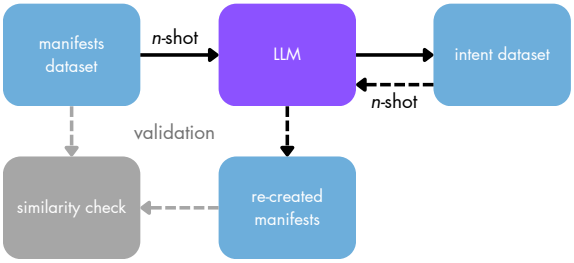


Figure 2: Overview of KGen's principle: manifests will be fed into LLMs to generate intents, which will then be reintroduced into the same LLMs to regenerate the manifests. The recreated manifests will be compared to the originals to evaluate their similarity.

3.1 Kubernetes Manifests Dataset

To build the Kubernetes Manifests Dataset, we propose a pipeline (see Figure 3) to extract values from a sample of workload manifests (*i.e.,* Pod, Deployment, Job, and CronJob), which have been collected from a set of production clusters, and group them into relevant categories (*e.g.,* authentication, certification).

Template generation. First, we generate a template for each Kubernetes manifest category. These templates have the structure of actual Kubernetes manifests (*e.g.,* `apiVersion`, `kind`, `specs`), but instead of real values, we insert placeholders formatted in HELM (HELM, 2025). We chose HELM due to its structured notation, which simplifies the representation of complex Kubernetes configurations (Zerouali et al., 2023).

Next, KGen recursively traverses each manifest using a Depth-First Search (DFS) approach. It explores each object's structure as deeply as possible before filling in values based on their hierarchical position following the HELM notation (*e.g.,* {{spec.containers.image}}). In the final step, we remove unnecessary elements, such as `status` and `annotations`, which typically store system-specific details or metadata not needed for defining cluster resources.

Value extrapolation. For each category in the example dataset, once the corresponding template was generated, we focused on extracting the distinct values associated with each object. To achieve this, we applied a recursive traversal method using the DFS strategy, ensuring that each object's structure was systematically explored. At every step of the recursion, we appended the parent object's name to the current field name. For instance, when going through the *spec* object, the traversal continues into *containers*, forming the identifier *spec.containers*, and then proceeds to *name*, ultimately constructing *spec.containers.name*. This hierarchical labeling approach ensures seamless alignment between the extracted values and the previously generated HELM-formatted template. Once the traversal reaches a terminal node in the structure, the algorithm records the corresponding value in a dictionary before resuming exploration along alternative paths.

Manifest generation. Once the templates were created and the distinct values for each manifest category were extracted, the next step was generating the final Kubernetes manifests. For each category, we began with its corresponding template and systematically replaced each placeholder with a randomly selected value from its associated list. For instance, the placeholder {{ spec.containers.image }} was substituted with a random entry from the

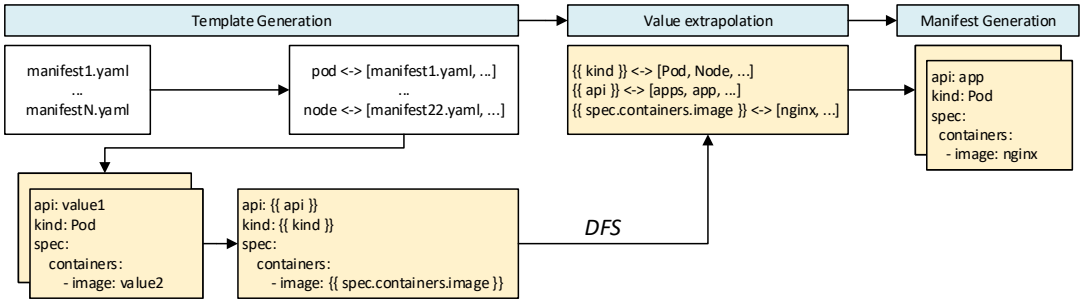


Figure 3: Pipeline for generation of Kubernetes Manifest Dataset from a subset of industry examples.

spec.containers.image list. Instead of performing a one-to-one substitution, we handled each placeholder individually to ensure diversity within the generated manifests, preventing excessive repetition of the same values. To further maintain uniqueness, each generated manifest was hashed, and any duplicates identified by matching hash values were removed from the output folder. As a final step, we assigned a unique metadata name to each manifest using a Universally Unique Identifier (UUID), ensuring that every generated file remained distinct.

3.2 Few-shot Learning for Intent Generation

After building the Kubernetes Manifest Dataset, the next step was to generate the descriptive intents. Although LLMs have demonstrated exceptional accuracy in various fields, their effectiveness heavily relies on well-structured prompting techniques, which can significantly enhance both the relevance and quality of their outputs, especially when applied to specialized domains (Zamfirescu-Pereira et al., 2023). To address this, in KGen we provided each selected LLM with two strategies to enhance the intent generation process: a structured context template utilizing the role field (*e.g.,* assistant, user) and a set of few-shot examples, ranging from 0 to 10, which helps the model understand the expected input-output pattern and improve response accuracy.

When the total input length, including the context template and examples, exceeds the model's token limit, we implemented a chunking method to divide the input into smaller segments. After processing, these segments were merged to ensure logical consistency and preserve YAML formatting. This structured approach improved coherence and accuracy while preventing errors in longer prompts (Zhou et al., 2024b).

Initially, since no descriptions or intents were available, we extracted a few samples from the Kubernetes Manifest Dataset and asked the LLMs to describe and create intent for them. This allowed us to take advantage of the models' few-shot learning ability, using their own outputs as a basis for following generations. Next, we manually reviewed the responses to ensure they met our expectations and began compiling a database of few-shot examples. Given the possibility of LLMs to produce hallucinated outputs (Ji et al., 2023; Yao et al., 2023), we made sure that the intents had different structures, some being more concise or schematic, others more elaborate, which helps the model generalize better and minimizes the risk of overfitting to a specific structure.

When using advanced prompting methods like few-shot examples, the goal is to have the model generate responses that closely align with the provided patterns and structures. To achieve this in KGen, we fine-tuned key hyperparameters that influence text generation. One of the most critical parameters we adjusted was temperature, which determines the randomness of the model's output. Research has shown that temperature settings significantly impact response accuracy and consistency (Renze and Guven, 2024; Saha et al., 2024; Shen et al., 2024). Lower values (*e.g.,* 0.2–0.5) make the model more deterministic, ensuring it follows the given structure more strictly, whereas higher values (*e.g.,* above 0.6) introduce more variability and creativity. Since studies indicate that lower temperatures tend to yield higher accuracy (Saha et al., 2024; Shen et al., 2024; Ifland et al., 2024), in KGen we set the temperature to 0.3. Additionally, we fine-tuned two other key parameters: *top_k* and *top_p*. The *top_k* parameter restricts the model to select from only the k most probable next tokens, while *top_p* ensures that the model only chooses tokens whose cumulative probability exceeds a specified threshold. In KGen, we configured *top_k* to 20 and *top_p* to 0.8 to strike

a balance between controlled generation and response diversity.

3.3 Few-shot Learning for Manifests Generation

For each intent, obtained with the KGen pipeline, we instructed the same LLMs to generate Kubernetes manifests using n-shot examples ranging n from 0 to 10, which allowed us to test all combinations of LLMs and examples. Before saving, each manifest was processed using YAML's *safe_load* function in Python to check for syntax errors or invalid formatting. Valid manifests were stored with a ".yaml" extension, while those failing validation were saved as "_error.yaml", allowing us to distinguish between correct and faulty outputs.

After generation, we checked whether the manifests were valid Kubernetes configurations. This was done using the *kubectl* command-line tool, enhanced by GNU Parallel to speed up execution (Tange, 2025). By ensuring that the generated Kubernetes manifests are correct and valid for both YAML and Kubernetes standards, we can significantly improve the quality of LLM outputs: fewer errors and more accurate automation in future applications. The results shown in Figure 4 indicate that all tested LLMs (*i.e.,* `Llama3-70B`, `Llama3-8B`, `Prometheus-8x7B-v2.0`, and `Mixtral-8x7B`) performed well in generating accurate manifests. Notably, `Mixtral-8x7B` and `Prometheus-8x7B-v2.0` showed increased validity as more examples were provided, suggesting that additional examples improve its accuracy. Opposite considerations can be reached with the `Llama3` family. These models showed higher accuracy with fewer examples, which implies that adding more examples might reduce precision.

Interestingly, some manifests were Kubernetes-valid, but not YAML-valid, possibly due to Kubernetes' more flexible structure compared to strict YAML formatting.

4 Evaluation

In this section, we present the results obtained from prototyping KGen, which played a key role in shaping our conclusions. First, we begin by analyzing the settings in which the experiment was conducted. Next, we discuss the evaluation process and show the similarity score achieved during manifest regeneration. This strategy allowed us to solve any possible issue with the intent generation (*e.g.,* hyperparameters' settings, model prompts, and template). Finally, we consider the economic aspect of employing each model.

4.1 Experimental Settings

Our analysis was performed in a production datacenter supported by computing clusters running four chosen LLMs: `Llama3-70B`, `Llama3-8B`, `Prometheus-8x7B-v2.0` and `Mixtral-8x7B`, and providing an increasing number of few-shot learning examples from 0 to 10. The cluster consists of 174 computing nodes running Intel and AMD processors with a range of 56 to 128 cores, between 768 and 2048 GB RAM, and each equipped with the 8 to 16 GPUs, mostly NVIDIA A100 and V100.

4.2 Similarity Check

After the initial validation at the Manifest Generation stage, we compared the generated manifests with the original ones used to create the intents. While it is still considered challenging to evaluate LLMs and there is no unique way of evaluating each model's responses, in KGen, we adopted four main evaluation metrics already known in the state of the art for tokens similarity (Hu and Zhou, 2024; Chen et al., 2024; Banerjee et al., 2023): (i) edit-distance (Levenshtein) score, that measures the minimum number of edits (*e.g.,* insertions, deletions, substitutions) needed to transform one string into another; (ii) Cosine similarity, which assesses semantic closeness by comparing word embedding vectors; (iii) BLEU (Bilingual Evaluation Understudy) algorithm, which calculates precision based on the ratio of matching token sequences; (iv) METEOR (Metric for Evaluation of Translation with Explicit Ordering), which uses a weighted average of various factors (*e.g.,* unigram precision, bigram overlap) to compare generated and reference text. It is important to note that we normalized similarity scores between 0 (completely different) and 1 (identical).

Each LLM received the previously generated manifest's descriptions (intents) with the request of generating the manifest back with an increasing number of few-shot examples (from 0 to 10). As shown in Figure 5, `Mixtral-8x7B` and `Prometheus-8x7B-v2.0` achieved better accuracy as more examples were provided, aligning with the increase in valid manifests seen in Figure 4a. The same coherency appeared in the `Llama3` models (`Llama3-8B` and `Llama3-70B`), where similarity scores peaked with only a few examples (0–3). A

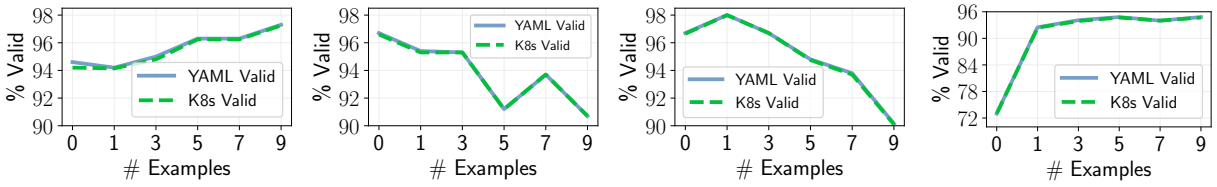


Figure 4: Number of valid K8s manifests in (a) `Mixtral-8x7B`, (b) `Llama3-8B`, (c) `Llama3-70B` and (d) `Prometheus-8x7B-v2.0`.

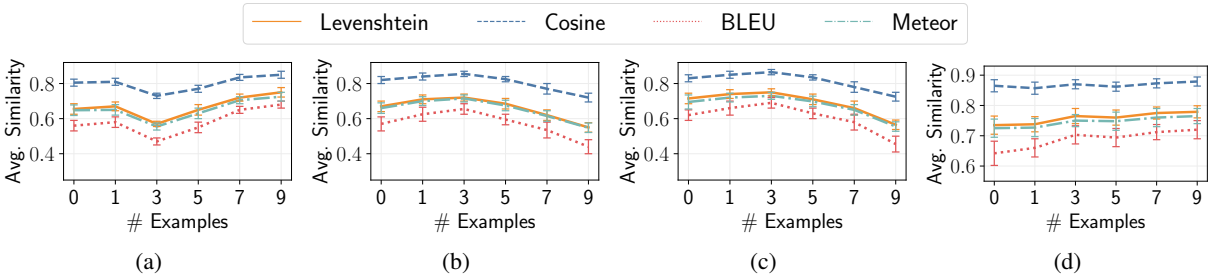


Figure 5: Similarity scores for (a) `Mixtral-8x7B`, (b) `Llama3-8B`, (c) `Llama3-70B` and (d) `Prometheus-8x7B-v2.0`.

more detailed analysis of the few-shot examples experiment is provided in Appendix 6, reporting the similarity scores between the initial manifest and the one generated from the LLM-based intent.

4.3 Economic considerations

Based on these evaluations, we also examined the cost implications of using different LLMs using publicly available information in (Artificial Analysis, 2025) and reporting results in Table 1.

First, we can point out that `Mixtral-8x7B` and `Prometheus-8x7B-v2.0` are associated with the same costs, since `Prometheus-8x7B-v2.0` is trained using `Mixtral-instruct` as a base model (Prometheus, 2024). Our analysis highlights that this class of models tends to be more expensive due to their need for more examples, whereas `Llama3-8B` keeps costs lower while maintaining strong performance. A similar trend is observed with `Llama3-70B`, which achieves high similarity scores without requiring additional examples. Despite a slightly lower percentage of valid manifests (96.68% vs. 98%), `Llama3-8B` remains a more cost-effective option, as both `Llama3` models achieved similar similarity scores. For large-scale applications, such as building datasets for LLM fine-tuning, `Llama3-8B` is preferred due to its price. However, when precision is the top priority, `Llama3-70B` might be the better choice despite its higher cost. As Table 1 illustrates, fewer provided examples result in fewer tokens, directly reducing overall costs.

Model	Input Price	Output Price	Dataset Cost
Mixtral-8x7B	0.70	0.70	1400
Llama3-8B	**0.07**	**0.20**	**270**
Llama3-70B	0.80	0.88	1680
Prometheus-8x7B-v2.0	0.70	0.70	1400

Table 1: Cost analysis of the tested LLMs: average input and output prices in $ per 1M tokens (or approx. 500 manifests of 1000 tokens length) and total cost of generation of $500k$ samples dataset similar to large scale cluster industry examples (Verma et al., 2015; Cortez et al., 2017).

5 Conclusion

In this paper, we presented *KGen*, a pipeline that translates natural language descriptions (intents) into Kubernetes (K8s) manifests for automatic cloud-native deployments. By analyzing different LLMs, our method strategically selects the optimal number of examples through a *n*-shot learning evaluation, balancing accuracy and computational efficiency. Our findings reveal that while increasing *n*-shot examples can enhance output quality for specialized models like `Mixtral-8x7B` and `Prometheus-8x7B-v2.0`, it may degrade the validity of K8s manifests for more general models such as `Llama3-8B` and `Llama3-70B`. This performance underscores the importance of tailored LLM selection for structured data generation, where smaller models can sometimes outperform larger ones. These insights emphasize the necessity of performing an in-depth LLM analysis to identify the most effective configurations to achieve higher generation accuracy at lower costs for DevOps pipelines.

Acknowledgements

This work has received funding from the EU Horizon Europe R&I Programme under Grant Agreement no. 101070473 (FLUIDOS).

References

Artificial Analysis. 2025. Artificial analysis. Accessed: 27-02-2025.

Debarag Banerjee, Pooja Singh, Arjun Avadhanam, and Saksham Srivastava. 2023. Benchmarking llm powered chatbots: methods and metrics. *arXiv preprint arXiv:2308.04624*.

Lekai Chen, Ashutosh Trivedi, and Alvaro Velasquez. 2024. Llms as probabilistic minimally adequate teachers for dfa learning. *arXiv preprint arXiv:2408.02999*.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds. *arXiv preprint arXiv:2501.06706*.

Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167.

Kristina Dzeparoska, Jieyu Lin, Ali Tizghadam, and Alberto Leon-Garcia. 2023. Llm-based policy generation for intent-based management of applications. In *2023 19th International Conference on Network and Service Management (CNSM)*, pages 1–7. IEEE.

Ahlam Fuad, Azza H Ahmed, Michael A Riegler, and Tarik Čičić. 2024. An intent-based networks framework based on large language models. In *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, pages 7–12. IEEE.

Yingqiang Ge, Wenyue Hua, Kai Mei, Juntao Tan, Shuyuan Xu, Zelong Li, Yongfeng Zhang, et al. 2024. Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems*, 36.

HELM. 2025. HELM format. Accessed: 19-03-2025.

Taojun Hu and Xiao-Hua Zhou. 2024. Unveiling llm evaluation focused on metrics: Challenges and solutions. *arXiv preprint arXiv:2404.09135*.

Yudong Huang, Hongyang Du, Xinyuan Zhang, Dusit Niyato, Jiawen Kang, Zehui Xiong, Shuo Wang, and Tao Huang. 2024. Large language models for networking: Applications, enabling techniques, and challenges. *IEEE Network*.

Beni Ifland, Elad Duani, Rubin Krief, Miro Ohana, Aviram Zilberman, Andres Murillo, Ofir Manor, Ortal Lavi, Hikichi Kenji, Asaf Shabtai, et al. 2024. Genet: A multimodal llm-based co-pilot for network topology and configuration. *arXiv preprint arXiv:2407.08249*.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843.

kagent. 2025. kagent.dev. Accessed: 19-03-2025.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2023. Prometheus: Inducing fine-grained evaluation capability in language models. *Preprint*, arXiv:2310.08491.

Nane Kratzke and André Drews. 2024. Don't train, just prompt: Towards a prompt engineering approach for a more generative container orchestration management. In *CLOSER*, pages 248–256.

Kubiya.ai. 2025. AI Agents for Kubernetes. Accessed: 2025-03-14.

Jieyu Lin, Kristina Dzeparoska, Ali Tizghadam, and Alberto Leon-Garcia. 2023. Appleseed: Intent-based multi-domain infrastructure management via few-shot learning. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 539–544. IEEE.

Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, et al. 2023. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *arXiv preprint arXiv:2305.18703*.

Logz. 2025. Demystifying K8S observability with Generative AI and LLMs. Accessed: 2025-03-14.

Dimitrios Michael Manias, Ali Chouman, and Abdallah Shami. 2024. Towards intent-based network management: Large language models for intent extraction in 5g core networks. In *2024 20th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 1–6. IEEE.

Abdelkader Mekrache, Adlen Ksentini, and Christos Verikoukis. 2024. Intent-based management of next-generation networks: an llm-centric approach. *IEEE Network*.

Meta/Llama. 2025a. Llama3-70B. Accessed: 25-02-2025.

Meta/Llama. 2025b. Llama3-8B. Accessed: 25-02-2025.

MixtralAI. 2025. Mixtral-8x7b. Accessed: 13-01-2025.

Prometheus. 2024. Prometheus-8x7B-v2.0. Accessed: 27-02-2025.

Saurabh Pujar, Luca Buratti, Xiaojie Guo, Nicolas Dupuis, Burn Lewis, Sahil Suneja, Atin Sood, Ganesh Nalawade, Matt Jones, Alessandro Morari, et al. 2023. Automated code generation for information technology tasks in yaml through large language models. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4. IEEE.

Matthew Renze and Erhan Guven. 2024. The effect of sampling temperature on problem solving in large language models. *arXiv preprint arXiv:2402.05201*.

Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. 2024. Llm for soc security: A paradigm shift. *IEEE Access*.

Maohao Shen, Subhro Das, Kristjan Greenewald, Prasanna Sattigeri, Gregory Wornell, and Soumya Ghosh. 2024. Thermometer: Towards universal calibration for large language models. *arXiv preprint arXiv:2403.08819*.

Ole Tange. 2025. GNU parallel 20240822 ('southport'). Accessed: 19-03-2025.

Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France.

Arthur Vitui and Tse-Hsun Chen. 2025. Empowering aiops: Leveraging large language models for it operations management. *arXiv preprint arXiv:2501.12461*.

Bin Xiao, Burak Kantarci, Jiawen Kang, Dusit Niyato, and Mohsen Guizani. 2024. Efficient prompting for llm-based generative internet of things. *arXiv preprint arXiv:2406.10382*.

Yifei Xu, Yuning Chen, Xumiao Zhang, Xianshang Lin, Pan Hu, Yunfei Ma, Songwu Lu, Wan Du, Zhuoqing Mao, Ennan Zhai, et al. 2024. Cloudeval-yaml: A practical benchmark for cloud configuration generation. *Proceedings of Machine Learning and Systems*, 6:173–195.

Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Mu-Nan Ning, and Li Yuan. 2023. Llm lies: Hallucinations are not bugs, but features as adversarial examples. *arXiv preprint arXiv:2310.01469*.

JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why johnny can't prompt: how non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–21.

Ahmed Zerouali, Ruben Opdebeeck, and Coen De Roover. 2023. Helm charts for kubernetes applications: Evolution, outdatedness and security risks. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 523–533. IEEE.

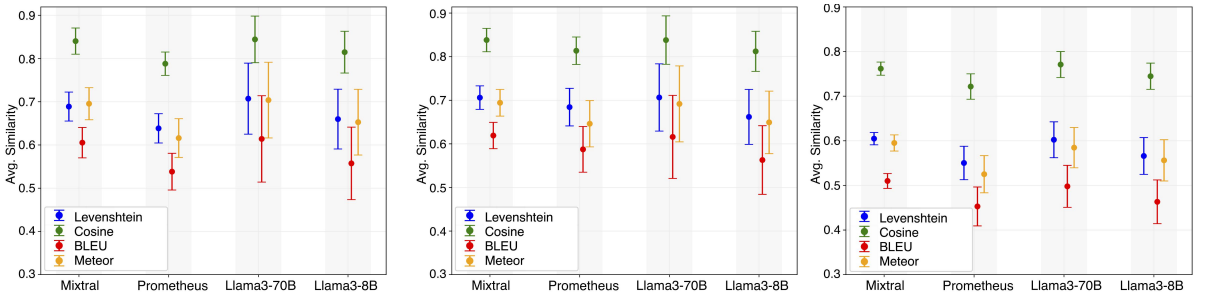
Yuechen Zhang, Shengju Qian, Bohao Peng, Shu Liu, and Jiaya Jia. 2024. Prompt highlighter: Interactive control for multi-modal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13215–13224.

Hao Zhou, Chengming Hu, Ye Yuan, Yufei Cui, Yili Jin, Can Chen, Haolun Wu, Dun Yuan, Li Jiang, Di Wu, et al. 2024a. Large language model (llm) for telecommunications: A comprehensive survey on principles, key techniques, and opportunities. *arXiv preprint arXiv:2405.10825*.

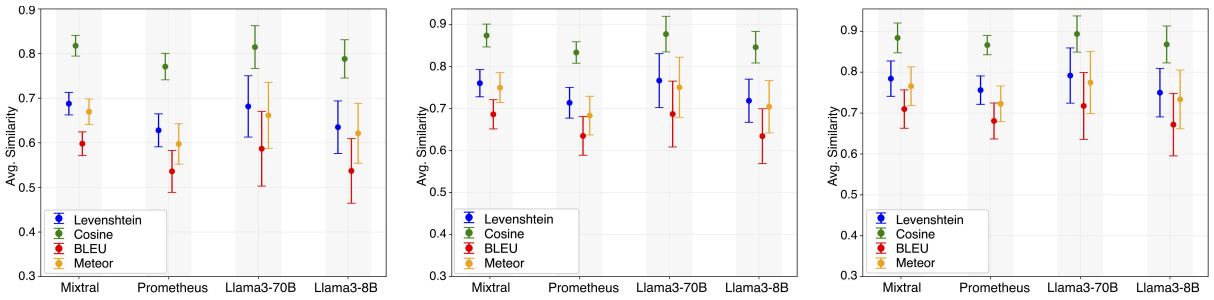
Zihan Zhou, Chong Li, Xinyi Chen, Shuo Wang, Yu Chao, Zhili Li, Haoyu Wang, Rongqiao An, Qi Shi, Zhixing Tan, et al. 2024b. Llm x mapreduce: Simplified long-sequence processing using large language models. *arXiv preprint arXiv:2410.09342*.

6 Appendix

We report here the detailed analysis of the *n*-shot examples experiments that demonstrate the impact of different numbers of example *n* and also highlight the difference between the performance of studied LLMs (see Figure 6 for `Mixtral-8x7B`, Figure 7 for `Llama3-8B`, Figure 8 for `Llama3-70B` and Figure 9 for `Prometheus-8x7B`). The analysis shows similarity scores between the initial manifest and the one generated from the LLM-based intent, plotted on the X-axis. For instance, Figure 6a illustrates the average similarity scores between the initial manifests and the generated ones, based on the intents produced by the LLM (shown on the X-axis).

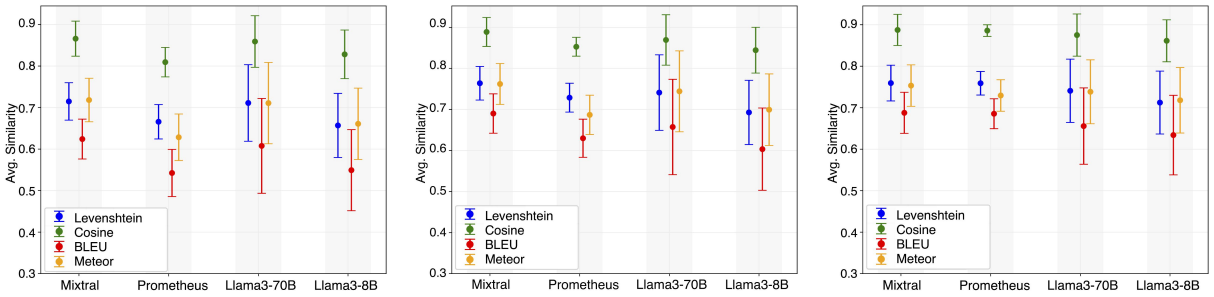


(a) `Mixtral-8x7B` 0-shot learning (b) `Mixtral-8x7B` 1-shot learning (c) `Mixtral-8x7B` 3-shot learning

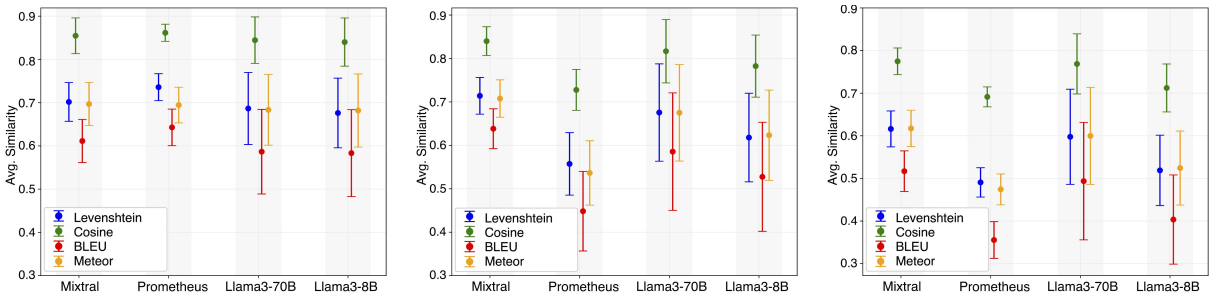


(d) `Mixtral-8x7B` 5-shot learning (e) `Mixtral-8x7B` 7-shot learning (f) `Mixtral-8x7B` 9-shot learning

Figure 6: Deep analysis for `Mixtral-8x7B` at increasing number of provided examples when the intents were generated from the LLMs on the x-axis.

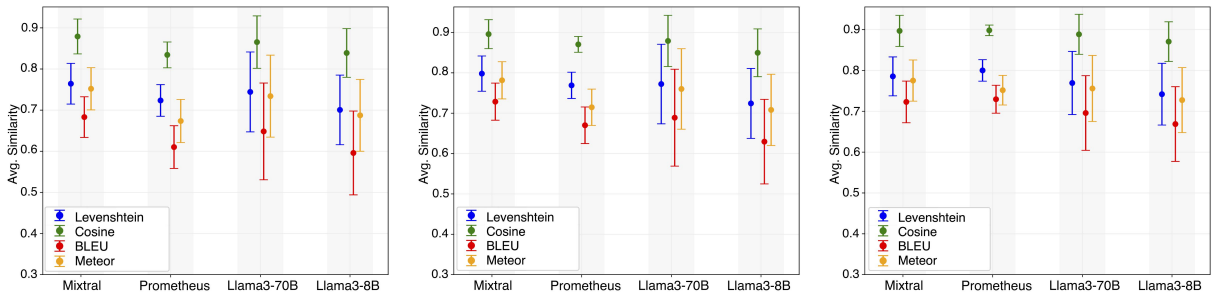


(a) `Llama3-8B` 0-shot learning (b) `Llama3-8B` 1-shot learning (c) `Llama3-8B` 3-shot learning

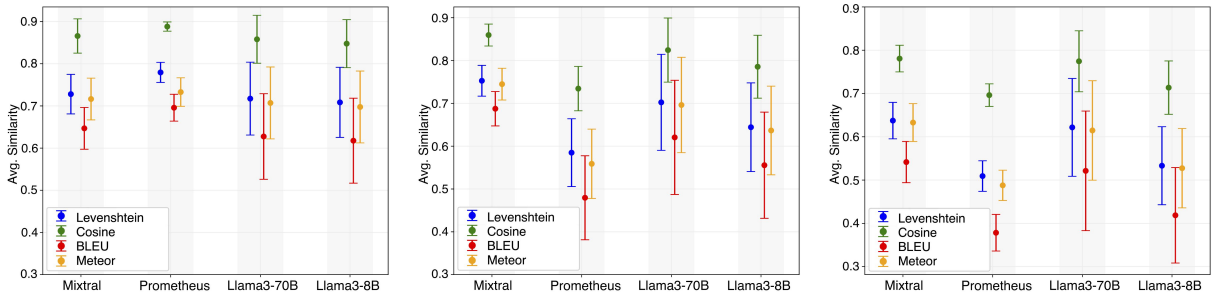


(d) `Llama3-8B` 5-shot learning (e) `Llama3-8B` 7-shot learning (f) `Llama3-8B` 9-shot learning

Figure 7: Deep analysis for `Llama3-8B` at increasing number of provided examples when the intents were generated from the LLMs on the x-axis.

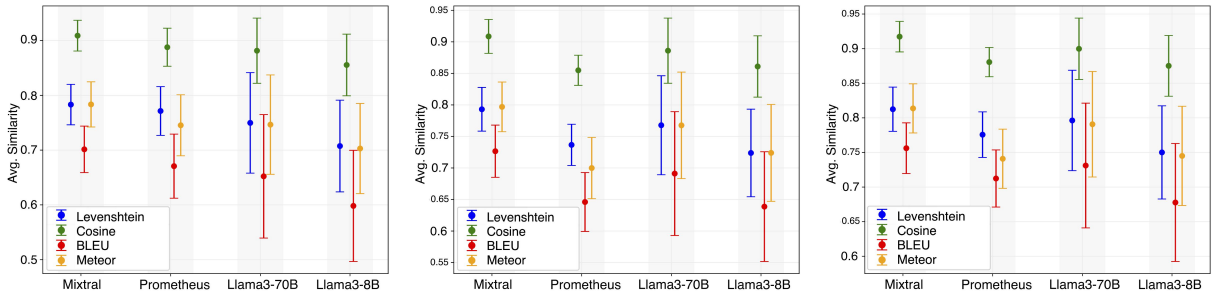


(a) `Llama3-70B` 0-shot learning (b) `Llama3-70B` 1-shot learning (c) `Llama3-70B` 3-shot learning

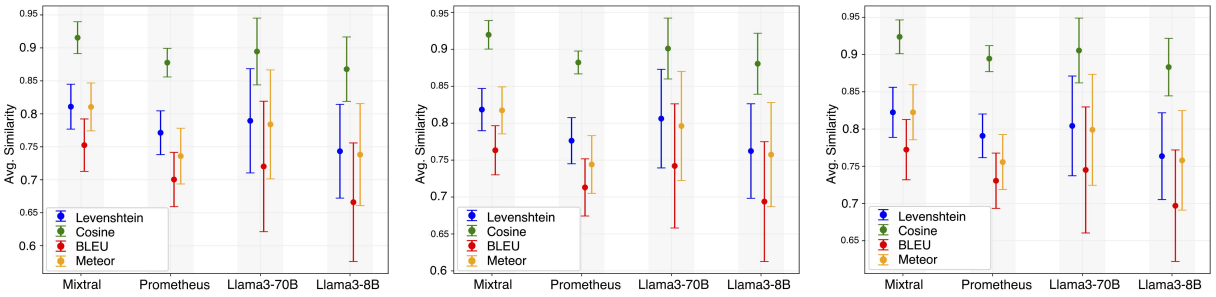


(d) `Llama3-70B` 5-shot learning (e) `Llama3-70B` 7-shot learning (f) `Llama3-70B` 9-shot learning

Figure 8: Deep analysis for `Llama3-70B` at increasing number of provided examples when the intents were generated from the LLMs on the x-axis.



(a) `Prometheus-8x7B` 0-shot learning (b) `Prometheus-8x7B` 1-shot learning (c) `Prometheus-8x7B` 3-shot learning



(d) `Prometheus-8x7B` 5-shot learning (e) `Prometheus-8x7B` 7-shot learning (f) `Prometheus-8x7B` 9-shot learning

Figure 9: Deep analysis for `Prometheus-8x7B-v2.0` at increasing number of provided examples when the intents were generated from the LLMs on the x-axis.