LLMs for Virtualized Networking Infrastructures: An Industrial Report

Luigi Pannocchi[1][0000−0002−6250−4939], Sourav Lahiri[2],
Silvia Fichera[2][0000−0002−8066−432X], Antonino Artale[2], and
Tommaso Cucinotta[1][0000−0002−0362−0657]

[1] Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56124 Pisa, Italy
`{luigi.pannocchi,tommaso.cucinotta}@santannapisa.it`
[2] Vodafone, Milan, Italy

Abstract. Machine Learning and Artificial Intelligence techniques are disrupting several fields of engineering, including telecommunications. In this domain, network operators are evaluating the use of data-oriented techniques to improve the management of virtualized networking infrastructures in several aspects, including monitoring, optimization, traffic forecasting, capacity planning, anomaly detection, and others. This paper summarizes our ongoing experience in the area of adopting human-machine interfaces based on Natural Language Processing for easing and speeding-up the interactions between these systems and network operators in their everyday work.

Keywords: Artificial Intelligence · Natural Language Processing · Text-to-SQL · Network Function Virtualization

1 Introduction

Network operators handle the data traffic from millions of users spread throughout vast areas. The amount of data managed by an operator is not limited to traffic and customers' data, but it also includes all the inventory and monitoring data related to the infrastructure. In the last decade, the Network Function Virtualization (NFV) [9] paradigm took momentum [12,27], causing the replacement of traditional physical network appliances sized for peak-hour operations, with Virtual Network Functions (VNFs). These are software components implementing network functions that are deployed as virtual machines or containers on general-purpose servers, following key Cloud Computing principles, like elasticity and dynamic adaptation to continuously changing traffic conditions. For the operator, it is vital to monitor all the appliances, physical and virtual, during the daily activities to keep the health of the network under control .

The challange is to handle such a large quantity of information, because data can be used to perform advanced monitoring operations such as fault detection and avoidance [7,6], performance prediction [22,5,15], capacity planning and instance placement [13,4,1]. Artificial Intelligence (AI) and Machine Learning (ML) techniques may be enablers for readily identifying faults and have a positive impact on the QoS of the hosted services [10].

All this data is organized and stored in traditional relational databases, as well as NoSQL data stores. To access this information, technical knowledge of querying languages is requested to a human operator that has to be able to build complex queries, considering the entire structure of the tables in the databases.

In the past, there has been a noticeable amount of work in the attempt to simplify and automate such interactive step [29]. The direction would be to achieve a more natural interaction with the data management systems, allowing to get the data just using natural language requests. However, considering the variability in application domains, and the dynamism of possible requirements, which constitutes a common factor in modern businesses, that was not enough to have an effortless way to interact with the data.

Still, it is clear that a more direct access to the data pool, even for the non-skilled worker, or even a complete automated retrieval, could provide clear advantages in terms of proactive management, operations efficiency and quality of service. This promise pushed significant research efforts towards the problem, and new tools are entering the scene nowadays.

With the advent of Large Language Models (LLMs), part of the showcase included tasks which are strictly related to what could be needed to accomplish that goal. Precisely, LLMs proved to be good at generating code [17,21], including SQL from natural language requests (NL2SQL) [11]. From the many announcements made by companies and research groups about reaching and outdoing the performance of previous solutions on the NL2SQL problem, one could be tempted to accept the idea that the problem has been already solved.

However, the ease with which the performance of the proposed models is announced is not a match with the difficulties when trying to implement those models in an industrial environment. While we are witnessing the growth of a market of third-party cloud-based services simplifying access to these services through appropriate APIs, we still need on-premise deployments of inference and training pipelines, to respect the confidentiality requirements that often characterize various industrial domains.

Paper contributions: This work is an industrial report, covering part of our ongoing attempts to achieve a higher degree of automation in the management of virtualized networking infrastructures, leveraging AI techniques. After an introduction to the general architecture in which AI/ML techniques are being used, we dig into the specific problem of developing an LLM model for the NL2SQL challenge, tailored to the Vodafone's specific business requirements, showing preliminary experimental results obtained using a variety of LLMs on a benchmark of specific queries from the NFV infrastructure management domain.

First, we introduce some useful background on human-like interaction with data in the following section.

2 Background on NL2SQL

The idea to automatically generate SQL code from Natural Language is not novel. SQL queries are highly syntactically structured in separate, but correllated, parts. Leveraging this structured nature, it is possible to approach the problem with methods coming from automatic software generation techniques, sometimes called "Sketching" [23], to generate each part [30,26]. This technique consists in providing a blueprint or partial implementation of the required code, and the algorithm completes that. Unfortunately, these approaches were not able to achieve near human performance, and often they were limited to specific cases. A big jump in performance was given by LLMs that showed a stunning capability to parse natural language text and produce meaningful output. Immediately, the feeling was to have a tool which is able to adapt to human requests without the need to nestle the operations into carefully pre-prepared blueprints.

The origin of this success was the conception of the "Transformer Architecture" in the Vaswani et. al. paper [24], originally composed of two parts: Encoder, which is processing the input into an internal representation; Decoder, which is producing the output in an autoregressive mode. After that work, different variations of the original Transformer were proposed leading to different models. The main branches can be summarized as: Encoder-Decoder models as the original Transformer; Encoder-only models, such as the *Bert* [8]; Decoder-only models, such as *Chat-GPTx* [2]. Even if the Encoder part is considered to be responsible of the "understanding" capabilities of the models, Decoder-only models have shown to be able to handle that too.

Indeed, whilst in the NL2SQL problem one could expect that the encoder is necessary to better capture the request, it is often the case that good performing models are of the autoregressive type, that is, Decoder-only [16,28].

Talking about performance, the works dealing with the NL2SQL problem usually are centered around specific datasets. Each dataset is made from a different set of databases, containing different topics, with different formats and complexity. One of the most popular one is the WikiSQL, which is reported as challenging due to the high number of tables it contains. This feature is expected to force the LLM models to generalize more [18], avoiding overtraining on a single database's vocabulary. However, the limitation of the WikiSQL dataset lies in the simple nature of the queries and schemas. In the original WikiSQL work [30], the dataset was proposed with several assumptions to make the problem simpler and tractable at that time. For this reason, whilst WikiSQL is a good baseline, it is not sufficient to cover more realistic cases. Having said this, it follows that a lot of results of LLM models achieving top performance on the WikiSQL dataset could not transfer directly to real applications.

The need for more challenging material pushed some research groups to propose more realistic datasets, such *Spider* [28] and *BIRD* [16]. The realism is given by introducing complex requests that require using the large spectrum of SQL language and reasoning. An interesting contribution of the *BIRD* dataset is the addition of "hints" in the input data. This add-on is necessary to realistically make it possible to parse more complex requests and also to train the model to reason on the question/schema pairs. A sample from the *BIRD* dataset is shown in Table 1. As it can be seen there, the question requires to associate the concept of "highest eligible rate" to a quantity that has to be computed indirectly from the columns of a table. The hint consists in defining the requested quantities such that the model can learn to recognize that concept.

Question	What is the highest eligible free rate for K-12 students in the schools in Alameda County?
Hint	Eligible free rate for K-12 = 'Free Meal Count (K-12)' / 'Enrollment (K-12)'
SQL	SELECT 'Free Meal Count (K-12)' / 'Enrollment (K-12)' FROM frpm WHERE 'County Name' = 'Alameda' ORDER BY (CAST('Free Meal Count (K-12)' AS REAL) / 'Enrollment (K-12)') DESC LIMIT 1;

Table 1. Sample from the *BIRD* dataset showing the presence of "Hint".

Given the new level of complexity, it was expected that on these new datasets the performance of the LLMs would be lower than on the trivial WikiSQL one. For example, the best *execution accuracy* on the *BIRD* dataset, after 1 year of many attempts, is around 65%. The *execution accuracy* is a measure the capability of the model to generate SQL queries that, when executed on the respective SQL database, are able to return the expected answer. These results show that the problem is still open and more effort is needed to reach the level of accuracy and reliability needed for real industrial applications.

3 Approach

This report springs from a research project [10] that has the goal of setting up a modular data-driven framework for operating Virtualized Networking Infrastructures. Such infrastructure has the capability to be paired with Artificial Intelligence modules that can support, co-operate or automate the handling of the Network Operations. Figure 1 depicts the main components of the *Vodafone* NFV intelligence system that already integrates AI modules to perform optimization of the operative costs [4] and near real-time anomaly detection [7].

The missing link for attempting full automation will require an integrative module, reported as "Network Automation Intelligence" in Figure 1, that leverages the vast integration capabilities of LLMs. Eventually, this module will be able to autonomously provide reports on the system status as shown in Fig. 2 and co-operate with the other AI modules to improve the system performance.

In the following, we describe the strategy by which we tackled the NL2SQL problem, reporting on various issues we encountered during the implementation. First, as mentioned in Section 2, modern LLMs exhibit non-negligible limitations in realiably tackling real use-cases, as needed in a network operator production environment. Therefore, we aimed at realizing a data processing pipeline using a customized model, obtained by fine-tuning a general-purpose LLM, which are expected to lead to improved accuracy figures.

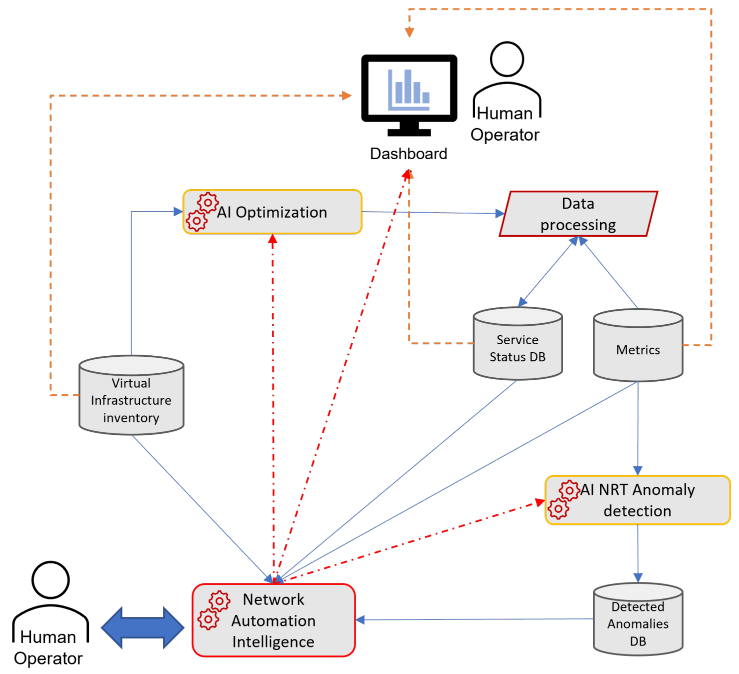

Fig. 1. Network Operations and associated Data Management.

3.1 Model Selection

In setting up the LLM framework, the first step consisted in selecting a model. Given the large number of contributions on LLMs, it was possible to benefit from different pre-trained models, based on different architectures. Precisely, an initial group of models have been selected as possible candidates for the task:

- *T5-large* model with 738M parameters [19];
- *Flan-T5-large* model with 783M parameters [3];
- *CodeT5+* model with 770M parameters [25];
- *StarcoderBase* model with 7B parameters [17].

The T5-based models were selected due to the Encoder-Decoder architecture that, at least in theory, should guarantee good properties in terms of instruction understanding. Indeed, the final application sees a human operator interacting, using natural language, with the LLM model to get some answer out of the company databases. There were several models available, which were reporting good capabilities to perform well in a diversity of application domains.

One of them was the *Flan-T5*, which was advertised as good at generalizing well on different prompts, without requiring the few shots approach. Considering that our final goal is to have something that can be used by operators asking different things, we decided to include this model in our evaluation.

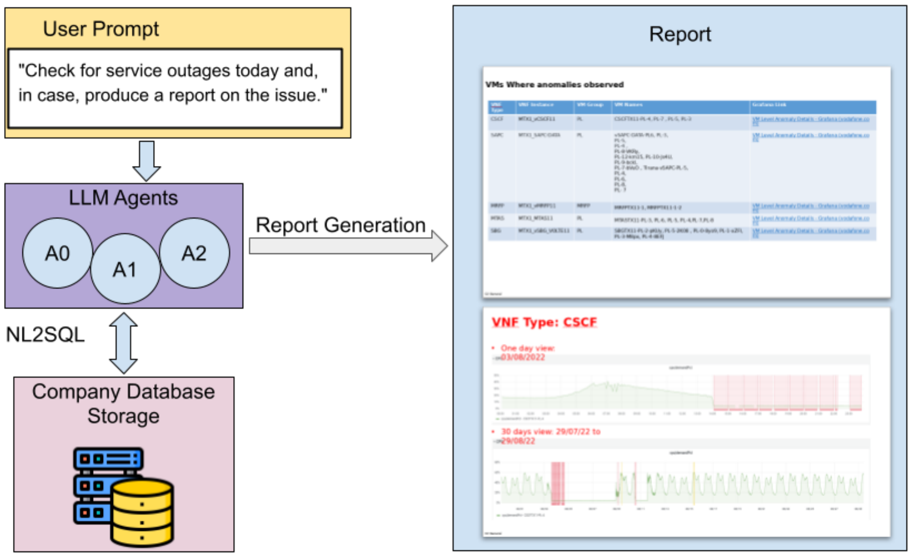
User Prompt
"Check for service outages today and, in case, produce a report on the issue."
LLM Agents
A0
A1
A2
Report Generation
NL2SQL
Company Database Storage
Report
VMs Where anomalies observed
VNF Type: CSCF

Fig. 2. Example of possible application where a user's natural language request triggers automatic data retrieval and report generation about the status of the VNF infrastructure by a set of LLMs deployed on-premises.

Another model, that was interesting for our task, was the *CodeT5+*, which was pre-trained on coding tasks. Given the code nature of the SQL output, this model had the opportunity to behave well.

We decided to keep also the baseline *T5* to see the differences. In the family of the *T5*, all the models were selected in the sub-billion parameters size. The criterion driving the size selection was the possibility to finetune the model with the available hardware without requiring too many optimization workarounds. Whilst the community developed several methods to make the training more manageable, usually it is a matter of trade-off between model performance and training feasibility. In this preliminary study we preferred to avoid introducing too many tunable variables which will make the results more difficult to analyse. In our case, the sub-billion models could undergo fine-tuning, without requiring extra quantization or computation offloading. The goal was to see what can be achieved with a relatively simple model, once trained on a specific dataset.

Even if our initial focus was on the full Encoder-Decoder architecture, many other works and public benchmarks were reporting better performances for Decoder-only models. One example is the *Starcoder* that was the substrate of some LLMs at the top of the *BIRD* [16] benchmark. This LLM is pre-trained on the curated coding dataset *The Stack* [14], thus it was expected to have a good starting point on which to build NL2SQL LLMs. Curiously, those good results were achieved with the 7-billion version of the model. Even if it is ten times bigger that the *T5-large*, it is one of the smallest among the better performing ones. For this reason, we decided to consider also the *Starcoder* model in our pool, expecting it to be a sort of good reference standard in our set.

One aspect, which was important to consider during our exploration, was the context length and memory occupation of the models. Depending on the specific application, the NL2SQL task could require a lot of context in the prompt. The models should allow for long context length without hitting the common Out-Of-Memory (OOM) error during fine-tuning. The *Starcoder* model allows a max context of 8K tokens, which turned out to be enough for our task. Conversely, the smaller *T5* models, even if occupying less memory, with their 512 context length, and supporting a relative attention which does not bound the max context length, suffered from OOM when trying to increase to context lenght above 1.5K tokens on the same hardware used to fine-tune the 7B *Starcoder*. This needs to be considered when selecting LLM models, or when deciding the input format.

3.2 Datasets and Repositories

As reported in Section 2, the development of LLMs is built around datasets for fine-tuning. Other research groups and companies released curated data for the NL2SQL task, together with code repositories freely available to perform hassle-free fine-tuning and evaluation of custom models. However, we found it difficult to reproduce the experiments given the fast obsolescence of the APIs used for LLM operations. This is probably an indicator that the technology is still not quite mature for industrial usage. For example, some models can benefit from using a specific datatype, such as the new "bf16" that is not well-supported on a 7-years-old *Tesla v100* and the same training script could lead to different results and unexpected failures. Even without considering hardware acceleration issues, we also found that a less-than-2-years-old research related repository was already incompatible with the current version of the Huggingface library, leading to subtle errors and software misbehavior that translated into weeks of lost development time. Another problem we found is that each repository provides the dataset with a specific format, requiring to perform adaptation in case one requires cross comparisons.

For our project, we decided to make an extended dataset, re-elaborating the data from *Wiki-SQL* [30], *Spider* [28] and *BIRD* [16] in a common format that can be easily employed during our experiments. It is possible to find more datasets online, particularly using the *Huggingface Hub*, but the selection of the previous datasets was driven by the fact that they also provide the original databases to perform execution accuracy tests.

It is common knowledge that, to obtain good accuracy in the NL2SQL task, LLMs should be provided with an informative context [20]. Precisely, it was highlighted that embedding the structure of the database in the input is necessary to allow the contextualization of the natural language question on the specific database. The level of details of the database structure may vary and depends on the specific application and model. There were works that considered table headers sufficient to contextualize the questions, whilst others embedded the full schema in the input. In realistic scenarios, the context should include as much information as possible about the database schema. The way the schema is encoded is another design choice. Some implementations are using a natural language description of the schema, others are using the SQL "create table ..." statement, whilst others use just a string with information about tables, columns' names and their types.

After different attempts, in this work, we opted for a compact representation of the full schema, with information about tables and keys. Precisely, we extract all the database structure in json format, and then we serialize it in a string to be included in the LLM prompt. The rationale behind our choice was that the natural language representation of the schema was too long and unnecessarily variable. The "create table..." format was dependent on the specific SQL dialect, and it was also long once tokenized. Our approach is trying to pack all the information needed to perform queries and link tables together in a uniform format, while being also independent from specific SQL dialects.

It is worth to add some considerations regarding this "in-house" customization. Changing the representation format of the LLM input and context could hinder the capability to leverage pre-trained models. This is because the pre-training could have been biased towards a specific format. It's then a good custom to check whether the customizations of the LLM inputs could lead to performance deterioration when using pre-trained models.

4 Preliminary Results

Currently, some preliminary results, obtained after some fine-tuning campaigns, using an *NVIDIA A100-40G*, on our extended dataset, report the *T5-large* and *Flan-T5-large* models as low-performing, with a bare 5-7% execution accuracy. The dataset included about $20K$ training samples and $1.5K$ test samples. After shuffling, the traning dataset was split into 70% training, 30% validation parts. The evaluation loss was reaching a minimum after about 5 epochs. The obtained low performance could have been expected, considering that it is in line with other comparable models on online leaderboards. However, the similar size model *CodeT5+* was able to deliver a 35% execution accuracy on the *BIRD* dataset. This confirms that a pre-training on code is beneficial for the NL2SQL task. The *StarcoderBase-7B* model, fine-tuned using LoRa adapters, was not achieving better performances due to the limitation in terms of our computation availability, which was forcing the LoRa rank to be kept low.

For these reasons, our attention was concentrated on the small *CodeT5+*. An indicative value for a good performance on the *BIRD* dataset would be around 60% of execution accuracy, achieved by specialized models such as the *DeepSeek-Coder-7B*. Yet, the final goal would be to obtain a good performance on a specific dataset coming from the Vodafone use case.

A limited number of natural language requests have been gathered from Vodafone network operators and a set of state-of-the-art LLMs ready-to-go models have been selected as a comparison set. Precisely, our choices were *DeepSeek-*

Coder-7B[11], *SQLCoder*[3] and *Mistral-7B*[4]. These models are advanced, and they can be used more interactively. Indeed, in the case of *DeepSeek* and *Mistral* we tried also a "2-passes" approach, where the model was fed with the error messages returned by the SQL engine in case a first attempt to generate a valid SQL query failed. In both cases, the models were able to correct some first-attempt mistakes and lead to SQL queries that could be parsed by the SQL engine.

Performing a quick test on the specific Vodafone data, leveraging only LLMs fine-tuned on other databases, the results were disappointing, as reported in Fig. 3, where different indicators were gathered during the tests. It was considered a Success when the model produced a valid SQL query and the value obtained from running this query on the database matched the value from the ground truth query. In terms of Failures, there were two possibilities: the SQL query was not valid, or the numerical value returned from the database did not match the ground truth. In the former case, it was often due to LLMs tendency to make up the answer, a phenomenon also called "hallucination". The typical hallucination observed in the tests consisted in the models referring to tables or columns not in the database schema. In the latter case, the LLMs were not "understanding" the user request and were retrieving the wrong quantitites.

On the 62 Vodafone samples, the *CodeT5+* was able to successfully hit around 15 of them. This is not a good result, however, as it can be seen from the figure, also the state-of-the art reference models were performing on the same level as our small model. The *DeepSeek* model is the one which was able to perform better in terms of valid SQL generation, however it anyway fell short in retrieving the right quantities. This is the reason why, from the figure it reports a higher number of value fails. The lower number of value fails of the other models are due to the fact that their generated queries were not able to be run by the SQL engine. In general, considering the overal success rate, both the *DeepSeek* and the *CodeT5+* were able to achieve around 20% of execution accuracy.

These results seem to indicate that the capabilities of the selected LLMs were not sufficient when used unmodified, and a tailored fine-tuning, using data similar to the final use-case, can be beneficial. The *CodeT5+* model was one tenth of the other models in size, but it performed comparatively well, so it seems a good candidate for further studies, focusing on a more specific fine-tuning.

5 Conclusions and Future Work

In this paper, we briefly mentioned the challenges being faced at Vodafone to industrialize a NL2SQL LLM-based pipeline to help network operators query the vast amounts of data being collected in the operator geo-distributed NFV infrastructure spanning across several EU countries.

Despite the wide availability of open-source software for LLM-based NLP computations, building on these technologies an effective human-machine interface for network operators proved to be all but straightforward. Indeed, several

[3] Defog SQLCoder: `https://github.com/defog-ai/sqlcoder`.
[4] Mistral-7B: `https://mistral.ai/news/announcing-mistral-7b/`.

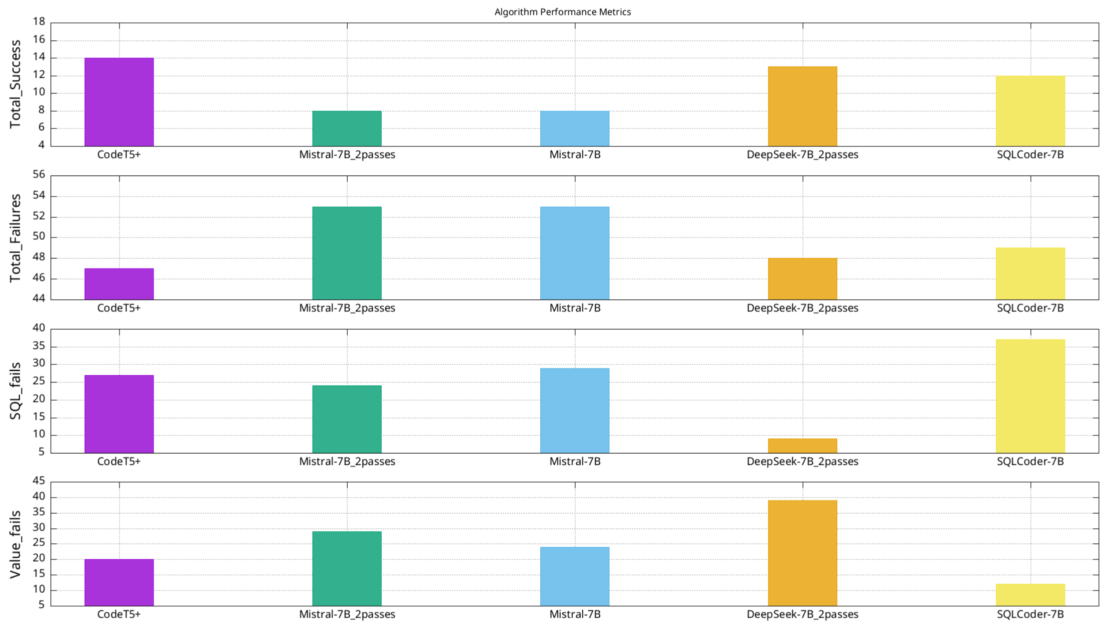

Fig. 3. Performance Counters of different models on the Vodafone test dataset.

challenges come in the way, from software dependency problems, to a wide range of incompatible data-sets formats that cannot be readily translated among each other, to problems related to the memory capacity and processing capabilities of the GPU(s) under use. These are to be added to the classical problem of creating high-quality specialized data sets over which to fine-tune domain-specific models, that is typical of the industrial deployment of AI-based solutions.

Following the issues highlighted in the preliminary results reported in this paper, it is possible to outline a set of future actionable steps. Precisely, the fine-tuning operation on the *CodeT5+* led to a noticeable improvement, however, hallucination issues were making it very unreliable on the Vodafone test dataset. In this regard, the following future steps will be taken: produce a custom dataset for extended fine-tuning, built on the schema of the real databases at Vodafone, and including human-curated domain-specific knowledge “hints”; consider the usage of Retrieaval Augmented Generation (RAG) techniques; consider the usage of multiple LLMs with routing.

Moreover, we plan to combine the NL2SQL building block with several others that are needed to realize the vision outlined in this paper: querying has to be performed across heterogeneous data storage systems, including local SQL-based and NoSQL, cloud-based ones; the results of the queries need to be represented in formats that are more readily usable by operators than tables, like graphs, heat maps and others. Finally, human-machine interaction may take advantage of voice-based exchanges. These are all aspects that will be explored in our ongoing research plans on the topic.

References

1. Andreoli, R., Forti, S., Pannocchi, L., Cucinotta, T., Brogi, A.: A logic programming approach to vm placement. In: Proceedings of the 14th International Conference on Cloud Computing and Services Science. SCITEPRESS (2024)
2. Brown, T.B., et al.: Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS '20, Curran Associates Inc., Red Hook, NY, USA (2020)
3. Chung, H.W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al.: Scaling instruction-finetuned language models. Journal of Machine Learning Research **25**(70), 1–53 (2024)
4. Cucinotta, T., Pannocchi, L., Galli, F., Fichera, S., Lahiri, S., Artale, A.: Optimum VM placement for NFV infrastructures. In: 2022 IEEE International Conference on Cloud Engineering (IC2E). IEEE (Sep 2022)
5. Cucinotta, T., et al.: Forecasting operation metrics for virtualized network functions. In: IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing. IEEE (May 2021)
6. Dandekar, A.: An approach for anomaly detection and prediction in time-series telecommunication data. In: 2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON). pp. 1–6 (2023)
7. Derstepanians, A., Vannucci, M., Cucinotta, T., Sahebrao, A.K., Lahiri, S., Artale, A., Fichera, S.: Near real-time anomaly detection in nfv infrastructures. In: 2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). pp. 26–32 (Nov 2022)
8. Devlin, J., et al.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1. pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019)
9. ETSI: Network Functions Virtualisation. White Paper 1, SDN and Openflow World Congress, Darmstadt, Germany (2012)
10. Fichera, S., Artale, A., Derstepanians, A., Pannocchi, L., Cucinotta, T.: Artificial Intelligence in virtualized networks: a journey. In: Proceedings of the Ital-IA Thematic Workshop: AI for Industry (2023)
11. Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y.K., Luo, F., Xiong, Y., Liang, W.: Deepseek-coder: When the large language model meets programming – the rise of code intelligence (2024)
12. Han, B., Gopalakrishnan, V., Ji, L., Lee, S.: Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine **53**(2), 90–97 (Feb 2015)
13. Jeong, B., Baek, S., Park, S., Jeon, J., Jeong, Y.S.: Stable and efficient resource management using deep neural network on cloud computing. Neurocomputing **521**, 99–112 (2023)
14. Kocetkov, D., et al.: The stack: 3 tb of permissively licensed source code (2022)
15. Lanciano, G., Galli, F., Cucinotta, T., Bacciu, D., Passarella, A.: Predictive autoscaling with OpenStack monasca. In: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing. ACM (Dec 2021)

16. Li, J., et al.: Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In: Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 42330–42357. Curran Associates, Inc. (2023)
17. Li, R., et al.: Starcoder: may the source be with you! Transactions on Machine Learning Research (2023)
18. Qin, B., et al.: A survey on text-to-sql parsing: Concepts, methods, and future directions. CoRR **abs/2208.13629** (2022)
19. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(1) (jan 2020)
20. Rajkumar, N., Li, R., Bahdanau, D.: Evaluating the text-to-sql capabilities of large language models (2022)
21. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C.C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., Synnaeve, G.: Code llama: Open foundation models for code (2024)
22. Saxena, D., Kumar, J., Singh, A.K., Schmid, S.: Performance analysis of machine learning centered workload prediction models for cloud. IEEE Transactions on Parallel and Distributed Systems **34**(4), 1313–1330 (2023)
23. Solar-Lezama, A., et al.: Combinatorial sketching for finite programs. In: ASPLOS XII (2006)
24. Vaswani, A., et al.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 6000–6010. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
25. Wang, Y., et al.: CodeT5+: Open code large language models for code understanding and generation. In: Bouamor, H., Pino, J., Bali, K. (eds.) Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 1069–1088. Association for Computational Linguistics, Singapore (Dec 2023)
26. Xu, X., Liu, C., Song, D.: Sqlnet: Generating structured queries from natural language without reinforcement learning (2017)
27. Yang, S., Li, F., Trajanovski, S., Yahyapour, R., Fu, X.: Recent advances of resource allocation in network function virtualization. IEEE Transactions on Parallel and Distributed Systems **32**(2), 295–314 (Feb 2021)
28. Yu, T., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Brussels, Belgium (2018)
29. Zelle, J.M., Mooney, R.J.: Learning to parse database queries using inductive logic programming (12 1996)
30. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning (2017)