Generating Commit Messages for Configuration Files in 5G Network Deployment Using LLMs

YANG Beining*†, SAMBA Alassane*, FRAYSSE Guillaume*, CHERRARED Sihem*
* *Orange Innovation, France*, † *Université Grenoble Alpes*

{beining.yang, alassane.samba, guillaume.fraysse, sihem.cherrared}@orange.com

***Abstract*—Network automation is crucial for improving network performance. Commit messages describes the different actions of the modification of network configuration files and deployments. This paper presents experiments and studies on automated commit message generation in the deployment of 5G networks. We extracted data from repositories of various projects engineered in Orange's 5G network. We then developed five prompts for experiments to identify the most suitable methods for this task. To select large language models, we used an in-house GPT-4 interface provided by Orange, and locally deployed popular large models such as Llama3, Mistral. We used both automated and human evaluation methods, selecting BLEU, ROUGE, and METEOR as our metrics for automated assessment. Our experiments shows that commit messages for configuration files generated by Large Language Models (LLMs) have better scores when using one-shot and Retrieval-Augmented Generation (RAG) technologies, for messages generated both by humans and bots.**

***Index Terms*—Commit Message Generation, Large Language Model, network automation**

I. Introduction

In recent years, the technology sector has increasingly focused on network automation due to the growing complexity of network configuration and management [1], [2].Automation technologies not only enhance efficiency but also help to minimize human errors in large-scale network deployments. In this evolving landscape, advancements in Deep Learning (DL) and Large Language Models (LLMs) have opened new avenues in network automation [3]–[6]. This paper specifically investigates the use of LLMs to automate the generation of network configuration files, with a particular emphasis on 5th Generation (5G) network deployments.

While the application of LLMs in generating commit messages for software development has been explored, there remains a notable gap in research focusing on their use for 5G-specific network files [7], [8]. Currently, network configuration processes predominantly rely on manually crafted scripts and templates. Although these traditional methods are somewhat effective, they require considerable expertise and extensive time investment. The emergence of advanced language models presents a significant opportunity to streamline these tasks in network automation. This paper aims to bridge the existing research gap by evaluating the effectiveness of LLMs in automating the creation of 5G network configurations, potentially transforming how networks are managed and deployed.

This research presents a case study centering particularly on the automation of commit messages in 5G network deployments. By extracting data from a real 5G network project deployed by Orange, we constructed a dataset containing Secure Hash Algorithm (SHA) values, authors, diffs, and commit messages. Based on this dataset, we designed several experiments to explore the most suitable LLM application methods for this task. The experiments used various advanced technologies, including Retrieval-Augmented Generation (Retrieval-Augmented Generation (RAG)) [9], and various large language models such as Llama3 [10] and Mistral [11]. RAG combines the capabilities of generative language models with a powerful retrieval mechanism, allowing the model to refer to a large amount of relevant historical data before generating responses. In this project, we used RAG technology to improve the precision of automatically generated commit messages for network configurations.

The primary objective of this study is to examine the ability of LLMs to automatically generate meaningful commit messages that are not only technically accurate but also contextually appropriate for network automation. To validate the models' effectiveness, we used a combination of automated and manual evaluation methods. The automated evaluation included metrics such as BLEU [12], ROUGE [13], and METEOR [14], providing a quantitative measure of the models' performance. Meanwhile, the manual evaluation involved a detailed review by project staff, using a scale to rate the accuracy, integrity, readability, and applicability of the generated messages. This comprehensive approach allowed us to assess the practical utility of automated evaluations in real-world settings.

II. Context and Problem Statement

This section explores the context of automating Network as Code (NAC) workflows using LLMs to generate informative commit messages. It establishes the importance of NaC in modern network management, introduces LLMs and their text processing capabilities, and defines the problem of automatically generating accurate commit messages for NaC using LLMs.

A. Network as code

In modern network management, the concept of is becoming increasingly popular. At its core, Network as Code (NAC)

978-3-903176-66-9 ©2024 IFIP

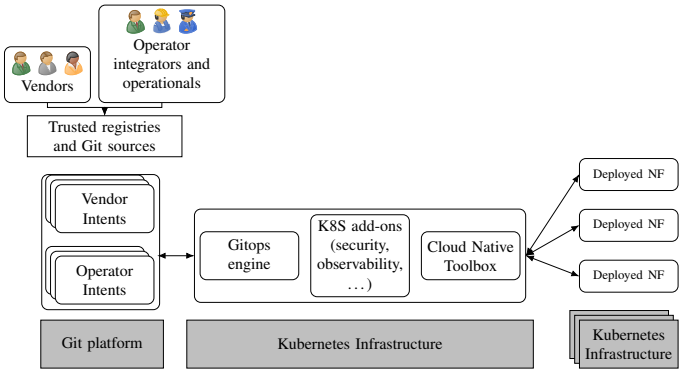

Fig. 1: Network as code

employs software development methodologies to manage and configure network devices and services [15]. It allows network configurations to be managed via version control systems and implemented through automated tools and processes, akin to operations in software development. This approach not only enhances operational efficiency and reduces human errors but also accelerates the deployment of network configurations. The introduction of NAC significantly improves the consistency, traceability, and replicability of network configurations, thus offering unprecedented flexibility and control in network operations.

Our study on NAC concentrates on the leftmost part of Fig. 1, integrating vendor intentions and operations into a network configuration system that includes the Open Container Initiative (OCI) registry and Git for source control. This system segment manages "Vendor Intents," which are specific configuration requirements from external suppliers, and "Operator Intents," representing internal configurations set by operator's network management team. We create and manage these intents through code, aiming for systematic, traceable, and efficient deployment and modification of network configurations, thereby enhancing efficiency, consistency, and flexibility in network operations.

B. LLM for network automation

LLMs leverage deep learning techniques to automate and enhance various computational processes, including network management. Central to these models is the transformer architecture that uses an attention mechanism to understand contextual relationships in text, which is crucial for interpreting complex network configurations and aiding in coding tasks for NAC practices.

The development of LLMs typically involves two crucial phases: pre-training and fine-tuning. During pre-training, the model learns from a vast corpus to grasp the language's structure, context, and semantics broadly. Fine-tuning then adapts the model to specialized tasks by training on narrower data sets, enhancing its proficiency in specific fields such as legal, medical, or technical domains.

C. Problem Statement: commit message suggestion

In network configuration management, accurate commit messages are crucial for version control and subsequent audit trails. However, manually crafting these messages can be tedious and error-prone. LLMs can automatically suggest descriptive and precise commit messages based on changes in configuration files, thus improving documentation quality and operational transparency.

III. Commit Representation

The representation of commits, how tools and technologies display and understand code changes, is key to improving the efficiency and accuracy of code reviews. In collaborative projects, a clear representation of commits can help team members quickly grasp the core content of changes, promoting more efficient communication and faster decision-making processes. Currently, many tools and platforms still have room for improvement, especially in how they clearly display complex code changes and conflict resolution solutions. The *git diff* command is an essential Git command that shows differences between two commits or between the working directory and the index. It follows the de-facto standard format introduced by the *diff* tool introduced in the 5^{th} edition of Unix in 1974. It details changes in file content, such as added, deleted, and modified lines, and is an indispensable tool in code review and version control. This command allows developers to precisely see the specific details of code changes through intuitive symbols; a "-" symbol indicates deleted lines, while a "+" symbol denotes added lines.

```
SHA 9e6ffb0e4f2647aab5dbdb1c03bab1650e894137
diff –git a/.gitlab-ci.yml b/.gitlab-ci.yml
index 3d85993..de7e42e 100644
– – – a/.gitlab-ci.yml
+ + + b/.gitlab-ci.yml
@@ -34,8 +34,8 @@ include:
    - project: 'oln/nif/cd/tools/renovate'
      file: 'gitlab-ci-renovate-lint-mr.yml'
    - project: 'oln/nif/cd/tools/template-template'
–     ref: 1.6.18
+     ref: 1.6.19
      file: 'templates/gitlab-ci-template.yaml'
    - project: 'oln/nif/cd/tools/template-template'
–     ref: 1.6.18
+     ref: 1.6.19
      file: 'templates/gitlab-ci-template-allow-push-to-main-branch.yaml'
```

Fig. 2: Example of diff ommit message.

Fig. 2 displays a the output of a *git diff* for the .gitlab-ci.yml file, illustrating changes committed in a recent update. The SHA identifier at the top specifies the unique commit hash. Differences between the old version (a/.gitlab-ci.yml) and the new version (b/.gitlab-ci.yml) are clearly outlined. The diff indicates modifications in the Continuous Integration (CI) configuration, specifically the removal of an older project and file references (project: 'oln/nif/cd/tools/renovate' and file: 'gitlab-ci-renovate-lint-mr.yml' with version 1.6.18), and the addition of new template files (templates/gitlab-ci-template.yaml, templates/gitlab-ci-template-allow-push-to-main-branch.yaml) under a new project reference (project: 'oln/nif/cd/tools/template-template') with an updated version 1.6.19. This example highlights how version control systems like Git help manage changes in software projects by providing a detailed record of code modifications.

IV. Datasets

Our datasets are sourced from a comprehensive collection of commits across various Git projects related to real-world 5G deployments conducted by Orange Group. The dataset is structured into two primary components: the first part focuses on performance evaluation of the generated commit messages, while the second part serves as the training set aimed at refining prompts and enhancing the generation process.

A. Evaluation Dataset

We meticulously extracted data from a total of 581 commits, out of which 168 were authored by human developers, and 413 were generated by automated systems (i.e., bots). This dataset encompassed a variety of information, including the differences between commits (i.e., diffs), commit messages, authorship details, and unique SHA identifiers for each commit. Such comprehensive data capture facilitates an in-depth analysis of the project's evolutionary development and the incremental modifications made over time.

Each commit was systematically archived into JSON files, a format that supports hierarchical and complex data structures. This choice of data format enhances compatibility with diverse programming environments and simplifies the integration with various data processing and analysis tools. In each JSON object, we stored detailed commit data including:

- **diff**: A textual representation of the changes made in the commit.
- **message**: The commit message that describes the intent and content of the changes.
- **author**: Information about the commit's author, which is crucial for understanding the context of the changes.
- **sha**: A unique identifier assigned to each commit, ensuring the traceability of every change within the repository.

B. Training Dataset for RAG

The training dataset is composed of 577 commits including both bot and human commits.

The training dataset for the RAG model was constructed to bolster the model's proficiency in producing relevant and accurate commit messages. Automated scripts navigated and extracted data from Git repositories, performing the following steps:

- **Data Extraction and Processing**: Retrieved all commits from each repository, extracted SHA identifiers, and computed diffs between consecutive commits using Git's native diff function. Commit metadata, including anonymized author details, messages, and diffs, was collected.
- **Data Structuring and JSON Storage**: Compiled the extracted data into JSON objects, encapsulating each commit's SHA, message, anonymized author, and diff, facilitating straightforward access during model training.

This structured dataset enhances the RAG model's training, focusing on synthesizing commit messages that reflect the nuances of software development workflows, thereby advancing the efficacy and precision of automated software engineering tools. Since the messages from bot commits are already automated, the inclusion of these commits in both the evaluation and training datasets provides deterministic reference commit messages. This allows for the assessment of the LLM's capabilities, combined with prompt engineering techniques, to identify the desired message patterns and content.

V. Models and prompt engineering

We used several base models to which we applied several prompt engineering methods, including RAG.

A. Base models used

In our research, we used a generative AI chat software, developed internally by Orange, which supports multiple types of model selections, including GPT, Mistral, Gemini, among others. Additionally, we locally deployed other large models including Llama3 [10], Mistral [11]. Our experiments primarily focused on how to effectively use the different configurations of these models to optimize the quality and relevance of the generated commit messages. Below are the model parameters we selected and a preliminary analysis of the results:

- **GPT-4**: used the GPT4-128k version, with a temperature setting of 0.6.
- **Llama3**: Employed the Llama3 8B version, with a temperature setting also at 0.6.
- **Mistral**: Used the Mistral 7B v0.2 version.

B. Prompt engineering

In our study, we explored the application of prompt engineering in automatically generating code commit messages by designing and implementing five different prompts. Each prompt contained distinct elements and instructions, aimed at evaluating their effects on improving the quality of the generated results. Below are the detailed designs of each prompt:

Background: You are a bot producing commit messages.

Task: Write a commit message that follows the Conventional Commits format ('<type>[optional scope]: <description>') based on the provided changes description as Input.

Input:

Change details:
[Repository Tree] + [DIFF]

Output:

Using the Conventional Commits as a format('<type>[optional scope]: <description>')

Fig. 3: Prompt1 structure.

- **Prompt 1**: Basic Prompt with a background field to explain the role we want the model to consider when answering, the task, the input, and finally the output that contains the desired output format of the commit message as showcased in Figure 3.

- **Prompt 2**: Prompt with Negative Instruction
  This prompt adds a negative instruction to Prompt 1. (*Your output must be strictly in one line and in the format '<type>[optional scope]: <description>' without any extra text like 'this is the commit message:' etc., neither before, nor after*). The purpose of the negative instruction is to guide the model to avoid generating commit messages that do not meet the task requirements, such as avoiding unclear or overly simplistic statements.
- **Prompt 3**: Prompt without Repository Tree
  Prompt 3 removes information related to the code repository tree from Prompt 2. This change aims to test the model's performance without specific code organizational structure information, thereby assessing the model's sensitivity to environmental dependencies.
- **Prompt 4**: Prompt with one-shot
  Prompt 4 adds a specific commit message example (one-shot) to Prompt 2. This method helps the model learn how to construct commit messages through a concrete example, potentially improving the accuracy and relevance of the generated information.
- **Prompt 5**: Prompt with RAG
  Prompt 5 integrates RAG into Prompt 2. through RAG technology, the model queries related documents or existing data before generating commit messages, enhancing the accuracy and richness of the generated content.

```
Prompt1: background + instruction +
input data + output indicator
  └─ Prompt2: Prompt1 + negative prompting
       ├─ Prompt3: Prompt2 - Repository tree
       ├─ Prompt4: Prompt2 + one-shot
       └─ Prompt5: Prompt2 + RAG
```

C. RAG

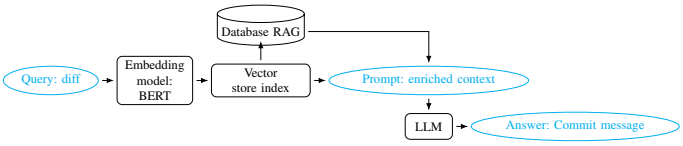

Fig. 4: Applied RAG architecture.

RAG enhances commit messages generation by integrating information retrieval with a generative model [9]. RAG operates in two stages: first, the model retrieves relevant data from a large dataset based on the input query; second, it combines this retrieved data with the input to generate the final output. This approach ensures that generated commit messages are accurate and contextually relevant.

In our research, RAG presented in Figure 4 is employed to refine commit message generation for version control systems, as explained in the following:

Information Retrieval: The model retrieves relevant historical commit data based on code changes, ensuring the context is well-understood.

Text Generation: The retrieved data is synthesized with the input to produce the final commit message, ensuring it is detailed and contextually accurate.

The implementation involves:

- **Embedding Model Selection**: We use the BERT model to convert textual data into embeddings, capturing the nuances of code changes.
- **Embedding Generation**: BERT generates embeddings for each code diff, representing the data in a machine-learning-friendly format.
- **Retrieval Mechanism**: Cosine similarity is used to find the most relevant historical commits, ensuring new messages align with successful past logs.
- **Message Generation**: The final commit message is generated using the combined input and retrieved data, ensuring accuracy and relevance.

VI. EVALUATION

In this study, to comprehensively assess the performance of the different AI models GPT-4, Llama3 and Mistral, as detailed in Section V, in generating code commit messages, we used both automated and manual evaluation methods. Each evaluation method has its advantages: automated evaluation provides quantifiable, comparable metrics, while manual evaluation allows for a deeper analysis of the practical usability and contextual adaptability of the generated text.

A. Automated Evaluation Methods

To quantitatively assess the efficiency of our generative models in creating commit messages that are consistent and contextually appropriate, we use three established metrics, these metrics typically range from 0 to 1. However, to enhance the visibility of differences between the results, we multiply all scores by 100. These scores are computed by comparing each generated commit message against a reference message : BLEU [12], ROUGE [13], and METEOR [14].

BLEU (Bilingual Evaluation Understudy) is used to measure the correspondence between a machine's output and a set of reference texts, primarily focusing on the accuracy of word sequences.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) assesses the overlap of n-grams between the generated text and the reference texts, emphasizing the recall of content.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) evaluates translation output by considering a range of factors including exact word matches, synonym matches, and the order of words, aiming for a more balanced assessment of both precision and recall.

B. Human evaluation methods

In addition to automated evaluation, we conducted a thorough manual evaluation to complement the automated metrics. This manual evaluation was performed by four human evaluators, all of whom are staff members associated with the project. They assessed a random selection of 10% of the total commits, which included 50 different commits. Each commit

was evaluated 11 times, resulting in a total of 550 individual assessments. Among these, 30 commits were manually written by humans, and 20 were generated by machines. Given that the large-scale GPT series developed by OpenAI is not open-source, we chose to manually employ the GPT-4 model for our studies to ensure information security. But in the end, we only conducted the experiment for prompt 1. Due to the limitations of the chat platform, we couldn't automate the retrieval of commit messages; instead, we had to manually input them and generate many at the same time. Therefore, the method of experimentation was different from other models and was very time-consuming.

The evaluators rated the commit messages generated under different prompts using a scoring scale from 1 to 3, defined as follows:

- 1 Point: Does not meet standards.
- 2 Points: Meets basic standards.
- 3 Points: Fully meets standards.

Ratings were based on the following five criteria:

- **Accuracy**: Evaluates whether the generated commit message accurately reflects the content of the code changes.
- **Integrity-What**: Assesses whether the commit message completely describes the changes made.
- **Integrity-Why**: Evaluates whether the commit message explains the reasons for the changes.
- **Readability**: Assesses whether the language of the commit message is clear and the format is correct.
- **Applicability**: Evaluates whether the commit message is applicable in a real software development environment.

VII. Results and discussion

A. Metrics evaluation results

The performance of different models across various prompts was quantitatively assessed using three metrics: BLEU, ROUGE (ROU), and METEOR (MET). Table I summarizes the results for both human and bot-generated commits.

The evaluation across various prompts shows that the Llama3 consistently outperforms the others, particularly in bot-generated commits where it achieved the highest scores in BLEU, ROUGE, and METEOR metrics. Mistral also performed well, especially in METEOR and ROUGE scores, making it a strong contender. GPT-4, while only evaluated on a single prompt, showed potential but requires further testing across more prompts to fully assess its capabilities. Overall, Llama3 stands out as more effective model across both human and bot-generated commits.

However, when focusing on the prompt techniques, we observe that for bot commits, Prompt 5, enhanced with RAG techniques, consistently performs the best across different models. In contrast, for human commits, there are instances where Prompt 4 also performs very well. This suggests that human commits allow for greater flexibility; even though Prompt 4 might not provide the most suitable examples for the model, it can still produce commendable commits. On the other hand, because bot commits are more structured, providing the model with examples that closely match the required format results in higher scores for the generated commits.

B. Human evaluation results

Based on the human evaluation data, this section provides a comprehensive analysis of the performance of three AI language models (i.e., GPT-4, Llama 3, and Mistral) across various evaluation metrics. The radar charts presented in Figure 5 compares the models under different prompts and evaluation criteria. Here we only compare the models llama3 and mistral. Table II, presents the numerical scores for each model under different prompts. These scores are consistent with the observations from the radar charts but offer more granular insights:

- **Llama 3**: For Llama3, whether it's human commits or bot commits, prompt5 consistently performs the best in all aspects, demonstrating that RAG technology significantly enhances Llama3's performance.
- **Mistral**: Mistral shows a very balanced distribution of scores across different prompts, indicating that the performance of commits generated by Mistral does not vary significantly from one prompt to another.

These findings suggest that while all models are capable of producing high-quality outputs, their performance may vary significantly depending on the nature of the prompt and specific evaluation metrics. Notably, Llama 3 performs best in human evaluations after the application of RAG techniques.

C. Comparing human evaluation and objective metrics

In Fig. 6 we compare human evaluation results with objective metrics by ranking.

The Llama3 model demonstrates relatively stable scores across both human evaluations and objective metrics, indicating a certain level of consistency between human preferences and automated scoring methods. For human commits, prompt5 or occasionally prompt4 perform the best, while for bot commits, prompt5 consistently shows the best performance. This indicates that across all metrics, prompt5, or sometimes prompt4, is the best.

Mistral shows some discrepancies between human evaluations and objective metrics, particularly in bot commits. While objective metrics show extremely high scores in Prompt 5, human evaluations, although still high, are not as extreme. This suggests that while Mistral may excel in specific automated metrics, human evaluators perceive its outputs differently, potentially due to nuances that metrics like BLEU, ROUGE, and METEOR may not fully capture.

The *Integrity why* score is almost always 2, and shows that the models is not able to fully apprehend the context of the changes. Also the *Applicability* score is around 2 except for Bot commit with Llama3, and shows that the models are not fully ready to be used without human intervention.

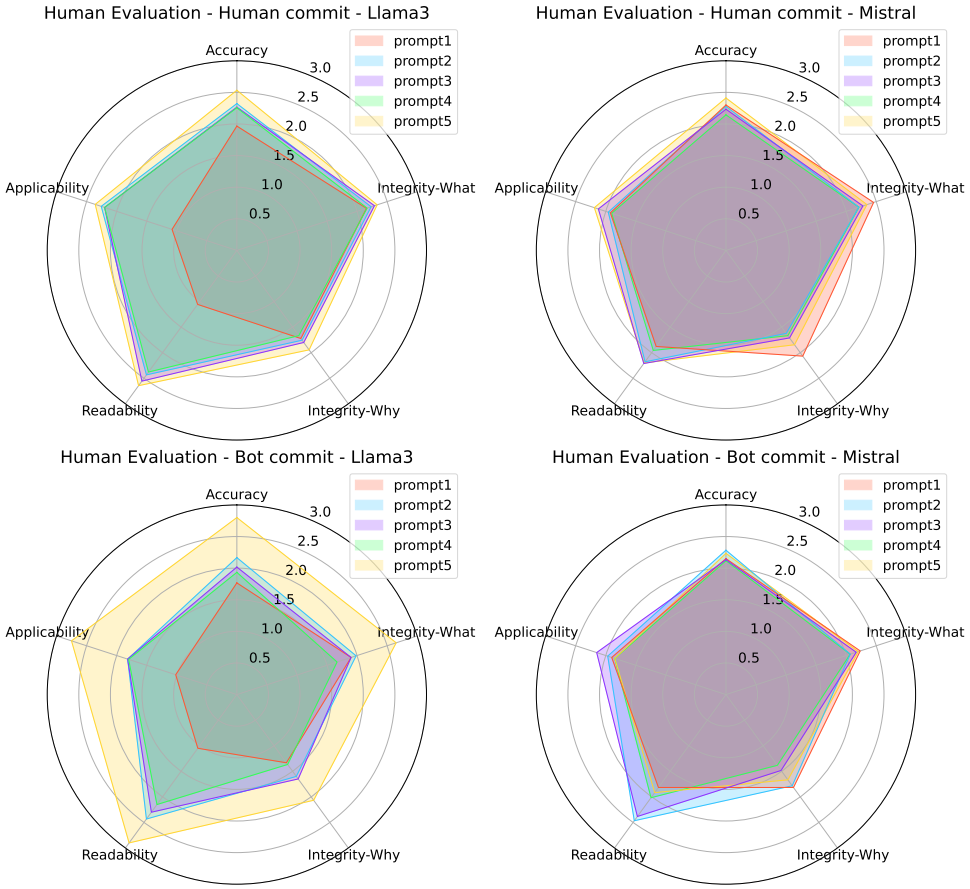

Fig. 5: Human evaluation results

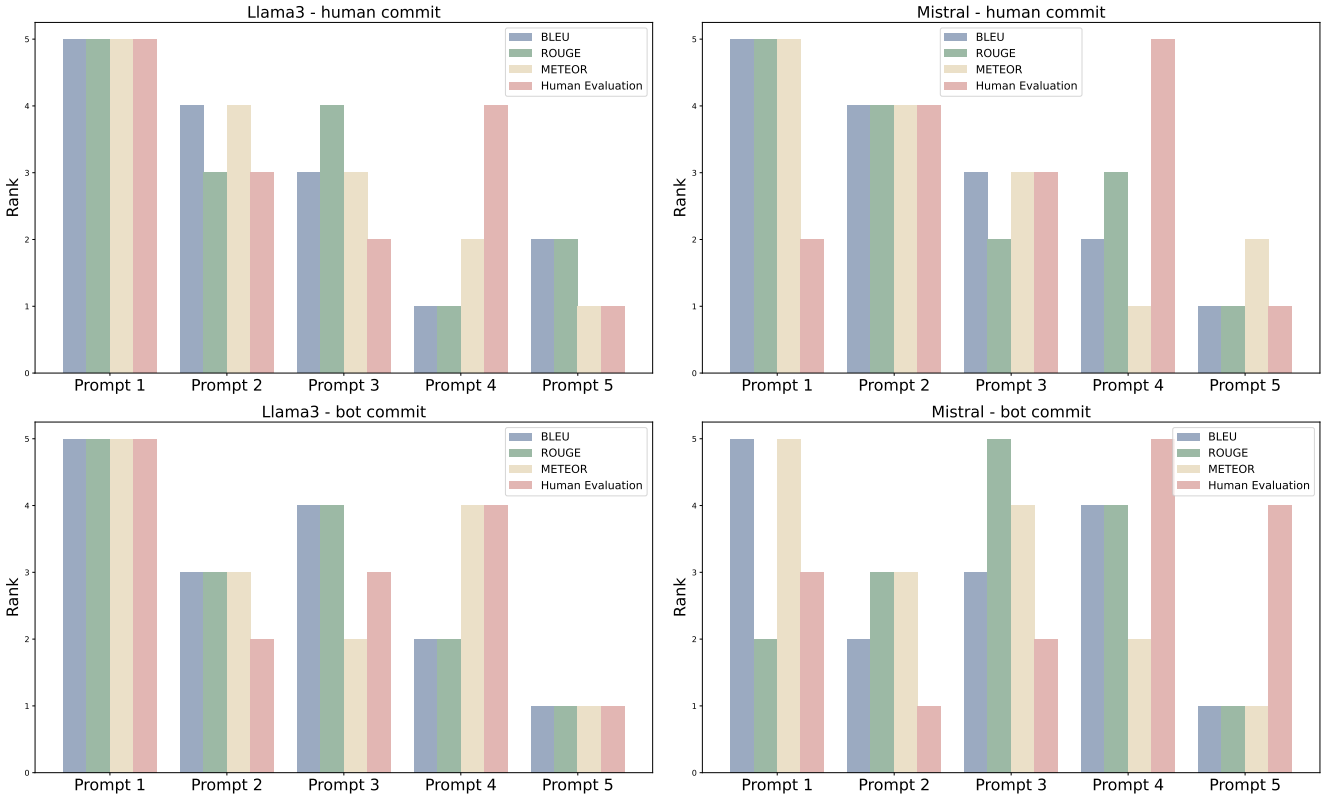

Fig. 6: Comparing human evaluation and objective metrics

TABLE I: Objective metrics evaluation results (multiplied by 100 for readability)

Model	Prompt 1			Prompt 2			Prompt 3			Prompt 4			Prompt 5		
	BLEU	ROU	MET	BLEU	ROU	MET	BLEU	ROU	MET	BLEU	ROU	MET	BLEU	ROU	MET
Human Commit															
Llama3_8B_Instruct	0.48	10.85	17.23	2.13	26.6	21.92	2.27	26.09	25.57	**4.99**	**42.58**	36.99	3.99	38.56	**38.62**
Mistral_7B_Instruct_v0_2	0.68	12.49	14.56	1.57	21.1	17.89	1.76	22.13	18.18	2.17	22.08	**32.27**	**3.63**	**29.66**	27.5
Bot Commit															
Llama3_8B_Instruct	0.28	10.15	14.66	2.64	27.64	20.02	2.0	25.13	25.23	3.02	28.73	15.83	**63.02**	**83.38**	**85.65**
Mistral_7B_Instruct_v0_2	1.11	25.98	23.46	2.14	25.25	27.85	1.93	24.53	27.34	1.84	24.81	28.45	**8.06**	**44.45**	**35.57**

TABLE II: Human evaluation results

Model	Prompt1	Prompt2	Prompt3	Prompt4	Prompt5
Human Commit					
GPT4	10.47	-	-	-	-
Llama3	7.98	11.00	11.11	10.67	**11.81**
Mistral	10.62	10.25	10.56	9.88	**10.99**
Bot Commit					
GPT4	10.35	-	-	-	-
Llama3	7.01	10.00	9.68	8.92	**13.17**
Mistral	9.92	**10.57**	10.32	9.43	9.87

VIII. Conclusion and Future Work

In this paper, we proposed the application of LLMs for network commit messages automation. Overall, LLMs generally perform well in tasks involving automatically generated commits messages, particularly with Llama 3 after the application of RAG techniques. The bot commits generated by Llama 3, enhanced by RAG, exhibit strong performance in both automated and human evaluations. This is likely due to Llama 3's enhanced capability to produce content that is already well-structured and formatted. In contrast, the evaluation scores for human commits are more balanced and consistent across different models and prompt methods. We have also observed that automated metrics are not stable when assessing commit message generation. For instance, in evaluating the Mistral model, there is a significant discrepancy between the results of automated assessments and human evaluations, indicating inconsistency. Therefore, traditional natural language processing metrics such as BLEU, ROUGE, and METEOR are not particularly effective in evaluating the generation of commit messages. Consequently, it is crucial to identify and develop appropriate evaluation metrics for different tasks related to large language models. Low score on the human evaluation of the *Integrity why* criteria could benefit from additional information either in the prompt or using an additional component to gather context information.

Moreover, in addition to commit messages automation, LLMs are poised to revolutionize network management by understanding and automating network changes based on both developer and business intents. Similarly, LLMs can bridge the gap between high-level business requirements and network operations. This dual capability enables seamless integration of business strategies and technical execution, streamlining the process of network configuration. However, these applications need to be further addressed and will open up future research directions.

References

[1] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "Deepconf: Automating data center network topologies management with machine learning," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, ser. NetAI'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 8–14. [Online]. Available: https://doi.org/10.1145/3229543.3229554

[2] H. Huang, S. Guo, G. Gui, Z. Yang, J. Zhang, H. Sari, and F. Adachi, "Deep learning for physical-layer 5g wireless techniques: Opportunities, challenges and solutions," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 214–222, 2020.

[3] D. M. Manias, A. Chouman, and A. Shami, "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)*, 2024, pp. 1–6.

[4] H. Zhang, A. B. Sediq, A. Afana, and M. Erol-Kantarci, "Large language models in wireless application design: In-context learning-enhanced automatic network intrusion detection," 2024. [Online]. Available: https://arxiv.org/abs/2405.11002

[5] P. Sikorski, L. Schrader, K. Yu, L. Billadeau, J. Meenakshi, N. Mutharasan, F. Esposito, H. AliAkbarpour, and M. Babaiasl, "Deployment of nlp and llm techniques to control mobile robots at the edge: A case study using gpt-4-turbo and llama 2," 2024. [Online]. Available: https://arxiv.org/abs/2405.17670

[6] Y. Kim, D. Kim, J. Choi, J. Park, N. Oh, and D. Park, "A survey on integration of large language models with intelligent robots," *Intelligent Service Robotics*, Aug. 2024. [Online]. Available: http://dx.doi.org/10.1007/s11370-024-00550-5

[7] P. Xue, L. Wu, Z. Yu, Z. Jin, Z. Yang, X. Li, Z. Yang, and Y. Tan, "Automated commit message generation with large language models: An empirical study and beyond," 2024. [Online]. Available: https://arxiv.org/abs/2404.14824

[8] L. Zhang, J. Zhao, C. Wang, and P. Liang, "Using large language models for commit message generation: A preliminary study," 2024. [Online]. Available: https://arxiv.org/abs/2401.05926

[9] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez, "Raft: Adapting language model to domain specific rag," 2024. [Online]. Available: https://arxiv.org/abs/2403.10131

[10] W. Huang, X. Zheng, X. Ma, H. Qin, C. Lv, H. Chen, J. Luo, X. Qi, X. Liu, and M. Magno, "An empirical study of llama3 quantization: From llms to mllms," 2024. [Online]. Available: https://arxiv.org/abs/2404.14047

[11] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[13] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[14] B.-K. Lee, C. W. Kim, B. Park, and Y. M. Ro, "Meteor: Mamba-based traversal of rationale for large language and vision models," 2024. [Online]. Available: https://arxiv.org/abs/2405.15574

[15] J. A. Shah and D. Dubaria, "Netdevops: A new era towards networking & devops," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2019, pp. 0775–0779.