ChatOps Assistant with NLP for Kubernetes

Tarun E[a,*]

[a] *Vellore Institute of Technology, Vellore, India*

Abstract

Kubernetes has become the standard for container orchestration, but its command-line interface (kubectl) includes a vast array of commands and options that can be daunting for practitioners. This paper explores the training of a custom language model, **Llama-3.1-8B-Instruct**, to act as a Kubernetes assistant by translating natural language instructions into correct kubectl commands (e.g., "create nginx deployment" → kubectl rollout restart deployment frontend). We fine-tune a 8-billion-parameter LLaMA-based model on a specialized corpus of **35k pairs** of human-like instructions [1] and Kubernetes CLI commands to enable natural language interface for cloud operations. We discuss how our approach leverages sequence-to-sequence learning and compare it with other large language models (LLMs) and code-focused models like T5, CodeBERT, and CodeT5. In our experiments, the fine-tuned model demonstrates promising accuracy in generating valid commands and generalizing to unseen tasks, reducing the need for Kubernetes expertise to perform routine operations. This work contributes to bridging NLP and DevOps, illustrating that an instruct-tuned LLM can simplify cluster management by understanding user intent and producing correct, executable commands.

Keywords: Natural Language Interface, Large Language Models, Sequence-to-Sequence, Llama, Code Generation, Kubernetes

1. Introduction

Kubernetes is a powerful yet complex system for automating deployment and management of containerized applications. Administrators and developers interact with Kubernetes primarily through kubectl, a command-line tool that exposes hundreds of commands

*Corresponding author

Email address: `tarun.2022@vitstudent.ac.in` (Tarun E)

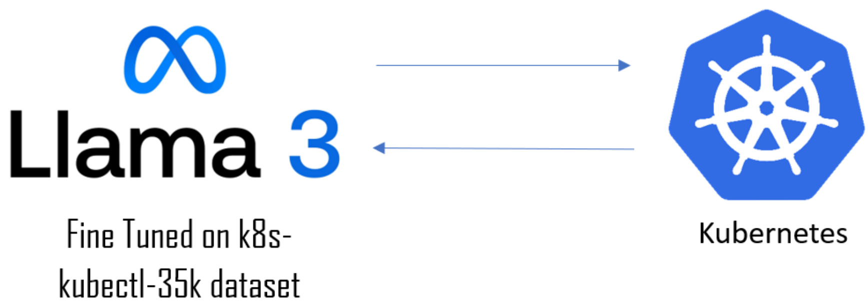

Figure 1: Architecture

and flags. Mastering these commands requires significant expertise, as even common tasks demand precise syntax and knowledge of resource types. Natural language interfaces for developer operations allow users to issue instructions in everyday language and have an intelligent system translate them into the appropriate technical commands. Our work focuses on such an interface for Kubernetes: Given a user query or intent in English, generate the correct kubectl command to execute that intent.

Recent advances in large-scale language models have made this vision feasible. LLms like OpenAI's GPT series and Meta's LLaMA have demonstrated an ability to generate code and CLI instructions from plain language descriptions [2]. This highlights how domain-specific training dramatically improves performance. Meta's LLaMA models further show that even smaller-scale open models can reach state-of-the-art results when trained on enough data [3] . Specifically, **LLaMA's 13B model** outperforms the much larger **175B GPT-3** on many benchmarks [3] , suggesting that with proper fine-tuning, moderately sized models can excel in specialized tasks. We therefore investigate fine-tuning an open-source LLM (based on LLaMA architecture) for the specialized task of Kubernetes command generation. We position our approach in the context of sequence-to-sequence models and code-focused transformers. **Sequence-to-sequence** architectures (like the encoder-decoder Transformer in T5) are well-suited for translation tasks – not only between human languages but also from instructions to commands.**Google's T5** ("Text-to-Text Transfer Transformer") [4] demonstrated the power of treating every NLP task as text-to-text; after massive pre-training, T5 achieved state-of-the-art on a wide range of language tasks by simply varying the input/output text formats [4]. This text-to-text framework can naturally encompass "NL $\rightarrow$ CLI command" translation. We also consider models pre-trained specifically on code. **CodeBERT**, for example, is a bimodal Transformer pre-trained on natural language and programming language data [5]. While CodeBERT is primarily an encoder (good for code search or classification tasks), encoder-decoder models like **CodeT5** combine the strengths of T5 with code-centric training to support both understanding and generation of code [6]. These models learn representations of programming syntax and semantics that could be very useful for mapping natural language to a formal command syntax.

Early efforts like **NL2Bash** [7] laid the groundwork by creating datasets of natural language to shell command pairs. However, Kubernetes manifests and commands introduce domain-specific vocabulary and constraints. Off-the-shelf LLMs might not reliably produce valid kubectl invocations, as they lack focused knowledge of this domain. This motivates our work to fine-tune an LLM on a targeted Kubernetes corpus. By training on 35,000 (instruction, command) pairs curated for Kubernetes tasks, our Llama-3.1-8B-Instruct model [8] learns the syntax and patterns of kubectl commands along with the associations to user intents. We choose a smaller 8B parameter model to allow feasible training on available hardware, **hypothesizing that domain-specific data can compensate for model size to some extent** (a hypothesis supported by recent findings that smaller specialized models can sometimes outperform larger general models on niche tasks [9]).

The salient contributions of the paper are,

1. **Domain-Specific Fine-Tuning of LLaMA-3.1-8B-Instruct:** We fine-tune an open-source LLaMA-3.1-8B-Instruct and T5-Base model on the ComponentSoft/k8s-kubectl-35k [1] dataset to accurately translate natural language queries into executable k8s kubectl commands.

2. **Parameter-Efficient Adaptation via LoRA:** We employ Low-Rank Adaptation (LoRA) techniques to achieve efficient fine-tuning, significantly reducing computational and memory requirements while maintaining high accuracy in Kubernetes command generation, demonstrating the viability of parameter-efficient methods

for domain adaptation.

2. Literature Survey

Natural Language to Command Translation: Early research on translating natural language to executable commands falls under semantic parsing and program synthesis. A notable contribution in this area is **NL2Bash** by Lin et al. (2018), who constructed a corpus of 9,000 natural language descriptions matched with Bash shell commands [7] . Their approach utilized a semantic parsing model to map English sentences to Bash syntax, representing one of the first attempts at an NL-to-CLI interface. The NL2Bash dataset was largely derived from web forums and expert annotation, and it revealed that even relatively simple shell tasks could be difficult for sequence models due to the need for reasoning about filenames, pipes, flags. Subsequent improvements came from applying neural machine translation techniques. For example, Fu et al. (2021) employed a Transformer-based sequence-to-sequence model on the Bash translation task, significantly improving accuracy over **RNN-based** baselines [10].By 2022, Fu et al. introduced an augmented dataset and workflow to synthetically expand training examples, achieving over 50% exact-match accuracy in translating English to Bash on their benchmarks [11] - a substantial leap from earlier performance 13% exact match.

Another line of work focused on interpretability: Bharadwaj and Shevade (2021) proposed an explainable method using Abstract Syntax Trees (ASTs) for the NL $\rightarrow$ Bash task [12]. By generating intermediate AST representations, their system made the translation more transparent and allowed users to understand which parts of the command corresponded to which words. These efforts, summarized in **Table 1**, demonstrated the viability of natural language interfaces for systems operations but also highlighted limitations: small dataset sizes, limited generalization to unseen commands, and lack of context about the system state.

S. No.	Title & Reference	Methodology / Dataset	Key Findings / Contributions	Limitations / Gaps
1	*NL2Bash: A Corpus and Semantic Parser for NL to Bash* – Lin et al., 2018 [7]	Collected 9k NL-command pairs; semantic parsing with seq2seq model for Bash CLI. Dataset from StackOverflow + expert curation.	Released first large dataset for NL→Shell; demonstrated feasibility of translating English to Bash commands.	Limited coverage of commands; baseline accuracy was low (<30% exact match); struggled with complex multi-step commands.
2	*Transformer-based Approach for NL2Bash* – Fu et al., 2021 [13]	Applied Transformer model (encoder-decoder) to NL2Bash using expanded training data (incl. synthetic examples).	Achieved substantial accuracy improvement (SOTA at the time, 50% exact match); showed effectiveness of data augmentation and modern NMT architecture.	Focused only on Bash; still errors on unseen utilities or rare arguments; needed execution feedback to further improve.
3	*Explainable NL2Bash (AST-Based)* – Bharadwaj & Shevade, 2021 [12]	Used Abstract Syntax Tree intermediate representation for parsing NL to Bash. Model predicts AST nodes which are then rendered as commands.	Improved interpretability of command generation; model's decisions more transparent by aligning NL phrases to parts of command syntax.	Slightly lower raw accuracy than pure neural approaches; limited to commands that can be represented by a known AST grammar.
4	*NLC2CMD Competition* – Agarwal et al., 2021 [11]	Organized NeurIPS competition with a new dataset (10k) of NL to Bash tasks. Various teams tried seq2seq, retrieval, ensemble methods.	Established standard evaluation for NL→Command; introduced execution-based metrics and energy efficiency considerations. Winning models achieved 70% execution success via ensembling.	Focus on Bash only; some solutions relied on task-specific tricks; generalization beyond competition data not proven.
5	*T5: Text-to-Text Transformer* – Raffel et al., 2020 [4]	Unified seq2seq architecture pre-trained on massive text corpus ("Colossal Clean Crawled Corpus"). Fine-tuned on diverse NLP tasks by framing each as text→text.	Achieved SOTA on many NLP benchmarks; demonstrated flexibility of a single model on translation, QA, summarization, etc. Provided a paradigm for treating code generation as "text translation."	Not specifically trained on code or CLI data; performance on code tasks improved only with further fine-tuning. Large model (11B for T5-11B) – resource-intensive.

S. No.	Title & Reference	Methodology / Dataset	Key Findings / Contributions	Limitations / Gaps
6	*CodeBERT: Pre-trained NL-PL Model* – Feng et al., 2020 [5]	Bi-modal Transformer (RoBERTa-based) trained on NL and source code data (GitHub) with MLM+RTD objective. Used for code search, generation via fine-tuning.	Learned joint embeddings for code and text; improved tasks like code search and doc generation. Enabled understanding of both code syntax and semantics.	Encoder-only architecture means it's not inherently generative; requires pairing with a decoder for generation. Not focused on CLI or YAML commands.
7	*CodeT5: Encoder-Decoder for Code* – Wang et al., 2021 [6]	Pre-trained T5 model on 8.5M functions in multiple languages plus comments. Introduced identifier-aware masking and dual-generation (code to comment) tasks.	Achieved SOTA on code summarization and synthesis tasks; handles both understanding and generation. Code-specific objectives enhanced performance.	Large model (220M–770M) but still smaller than GPT; may miss domain-specific tokens (e.g., K8s resource names) unless fine-tuned.
8	*CodeT5+: Open Code LLMs for Understanding and Generation* – Wang et al., 2023 [14]	Unified seq2seq architecture with modular encoder-decoder, trained on large-scale code datasets using mixed objectives; supports models from 220M–16B.	Outperforms larger models in many code tasks; small models (<1B) can compete well with specialized objectives.	T5-Base scale models still underperform on hard code tasks; lacks focus on CLI/IaC domains.
9	*KGen: Kubernetes Manifest Generation* – Angi et al., 2025 [9]	Pipeline fine-tuning LLMs for K8s YAML creation. Used few-shot prompt analysis and fine-tuned models (e.g., LLaMA3-8B, Mixtral-8x7B) on intent→manifest data.	Validated LLMs can generate correct K8s configs. Smaller MoE models with good examples outperformed larger general ones.	Focused on YAML configuration, not imperative commands. Pipeline is complex. Syntax correctness issues remain.
10	*Intent-Based Cloud Management (Appleseed)* – Lin et al., 2023 [15]	Few-shot system for infrastructure automation. Users express intents (e.g., network, cloud), and LLM generates actions or configs.	Multi-domain applicability; reduced burden on users needing deep technical knowledge. Enabled intent-to-action translation.	Still early-stage; performance varies across domains. Requires robust grounding and feedback mechanisms.
11	*Llama 2: Open Foundation and Fine-Tuned Chat Models* – Touvron et al., 2023 [16]	Introduced LLaMA 2 models (7B–70B) and fine-tuned chat variants using instruction tuning + RLHF on web-scale datasets.	Matched performance of closed models in helpfulness and safety; open-source and reproducible methodology for instruction tuning.	Still lags GPT-4 in complex tasks; vulnerable to hallucinations; long-context reasoning is limited.

S. No.	Title & Reference	Methodology / Dataset	Key Findings / Contributions	Limitations / Gaps
12	*Code Llama: Open Foundation Models for Code* – Rozière et al., 2023 [17]	Fine-tuned LLaMA 2 on curated code datasets (multi-language, Python-specialized, and instruction variants); supports 16K context and infilling.	SOTA performance on HumanEval, MBPP, and MultiPL-E benchmarks; Python-specialized 7B outperforms general 70B.	Limited performance in non-Python domains; does not guarantee secure or deployable code.
13	*Balancing Continuous Pre-Training and Instruction Fine-Tuning* – Jindal et al., 2024 [18]	Compared strategies for updating LLaMA 3 models while preserving instruction-following; evaluated 1.5B–8B scale models.	Proposed compute-efficient training pipeline to retain instruction skills after updating base models.	Less effective for very small models; assumes access to both base and instruction-tuned models.
15	*Deployability-Centric IaC Generation: An LLM-based Framework* – Zhang et al., 2025 [19]	Iterative generation + feedback using LLMs (e.g., Claude) on real IaC deployment tasks; introduced DPIaC-Eval benchmark.	Boosted deployability to 98% using feedback loops; evaluated intent, syntax, and security alignment.	Initial deploy success rate 30%; intent alignment only 25%, security compliance just 8%; small models fail without feedback integration.
16	*The Unreasonable Effectiveness of Few-Shot Learning for Code Generation* – Mishra et al., 2022 [20]	Compared T5-base and larger models on few-shot tasks (e.g., APPS, HumanEval); evaluated accuracy vs. model size.	Showed that T5-base and other small models fail to generalize in low-data regimes; large models are disproportionately better for code.	Small seq2seq models struggle with long-range dependencies and structured output formats (like code/CLI).

2.1. Summary

From the above survey,

1. We identify several gaps that motivate our work. First, while translation of NL to general code or Bash commands has been studied, the specific case of Kubernetes CLI commands remains under-explored – prior models don't natively understand Kubernetes resource types or command structures

2. Existing LLMs can produce code, but without fine-tuning they often hallucinate or produce incorrect outputs in niche domains; a focused **fine-tuning on Kubernetes data** is needed to achieve reliability (addressing domain knowledge gap)

3. We aim to develop an NLP solution that understands Kubernetes-specific intents and reliably generates the exact kubectl commands to using **LoRA training**. By doing so, we tackle both the NLP challenge (mapping ambiguous natural language to a formal action specification) and the software engineering challenge.

3. Problem Description

Kubernetes is the industry-standard platform for container orchestration, yet its command-line interface (kubectl) is notoriously complex. Practitioners, especially those new to DevOps, face significant challenges in navigating the vast number of commands, flags, and options required for routine cluster management. Even experienced engineers may struggle to recall the precise syntax for tasks such as scaling deployments, restarting pods, or retrieving logs, leading to inefficiencies, errors, and steep learning curves.

While documentation and cheat sheets exist, they demand constant reference and manual effort, which is neither scalable nor user-friendly in fast-paced cloud environments.

This creates a gap between natural language intent and executable cluster operations. Bridging this gap requires a system that can understand human-like instructions (e.g., "restart the frontend pod") and translate them into correct kubectl commands (kubectl rollout restart deployment frontend). Therefore, the problem addressed in this work is:

- How to design an intelligent assistant that reduces Kubernetes complexity by enabling users to interact with clusters through natural language.

- How to leverage large language models (LLMs) effectively for this task, ensuring accuracy, generalization, and reliability compared to traditional code-focused models. We will be fine tuning Llama and Google T5 models using ComponentSoft/k8s-kubectl-35k dataset with PEFT LoRa Method.

3.1. Framework

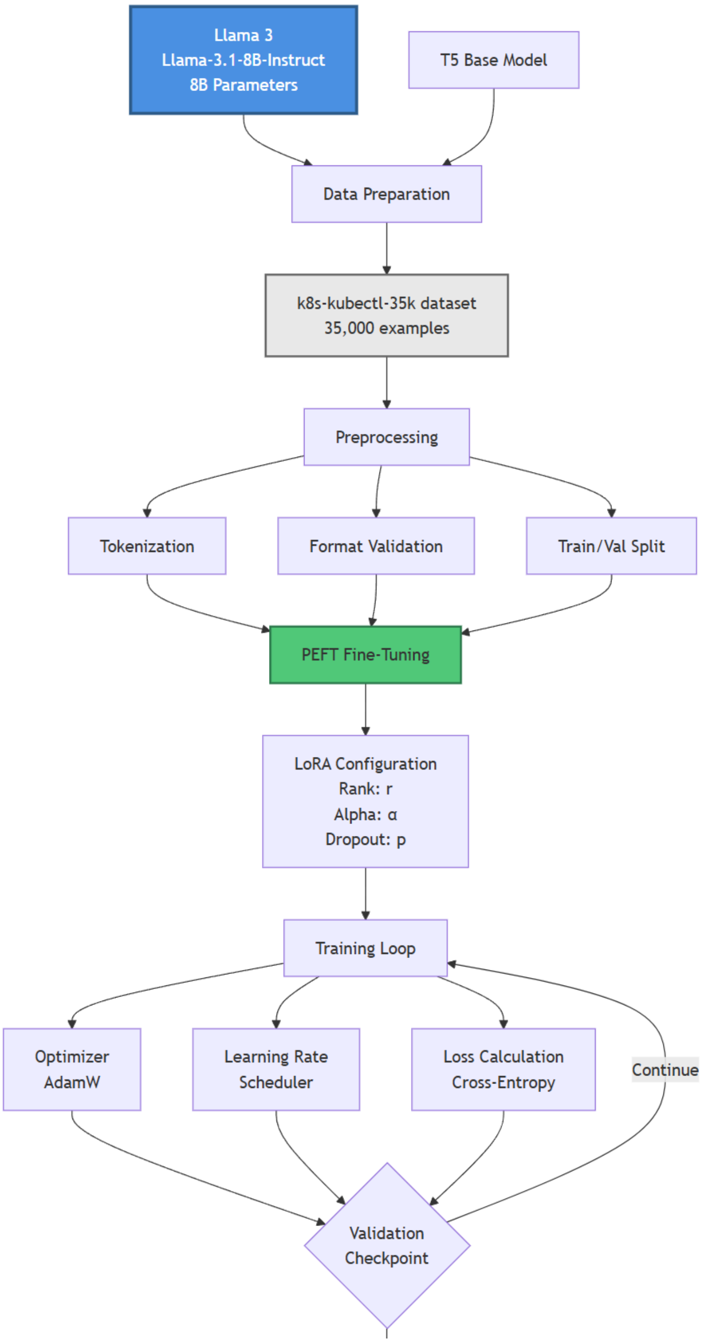

Figure 2: Framework of the project

1. **Initialize Environment**
   - Import libraries (`torch`, `transformers`, `datasets`, `sklearn`, etc.)
   - Set `device = "cuda"`
2. **Load and Preprocess Data**
   - Read CSV file containing (question, command) pairs
3. **Tokenization**
   - Load tokenizer from base model (`Llama-3.1-8B-Instruct`)
   - Add special tokens if missing (e.g., padding)
   - Convert (question, command) into instruction-style prompt
   - Encode into `input_ids` and `attention-mask`
4. **Create Dataset Objects**
   - Wrap encoded data into a PyTorch Dataset class
   - Implement `__getitem__` returning {`input_ids`, `attention_mask`, `labels`}
   - Prepare `train_dataset` and `val_dataset`
5. **Model Setup**
   - Load base model (`AutoModelForCausalLM`)
   - Enable gradient checkpointing for memory efficiency
   - Assign model to device
6. **Define Training Arguments**
   - `output_dir = "./k8s-command-model"`
   - `num_train_epochs = 3`
   - `learning_rate = 2e-5`
   - `warmup_steps = 100`
   - Use `fp16` or `bf16` for mixed precision
   - Perform evaluation and checkpoint saving every N steps
7. **Trainer Initialization**
   - Pass model, training arguments, and datasets
8. **Training Loop**
   - Trainer trains model for the specified epochs
   - Periodically evaluates on the validation set
   - Saves the best model checkpoint (lowest validation loss)
9. **Save Model & Tokenizer**

- Save final fine-tuned model
- Save tokenizer to `output_dir`

10. **Inference Testing**
    - Load trained model from checkpoint
    - For a sample input question, generate the corresponding `kubectl` command

3.3. Flow Diagram

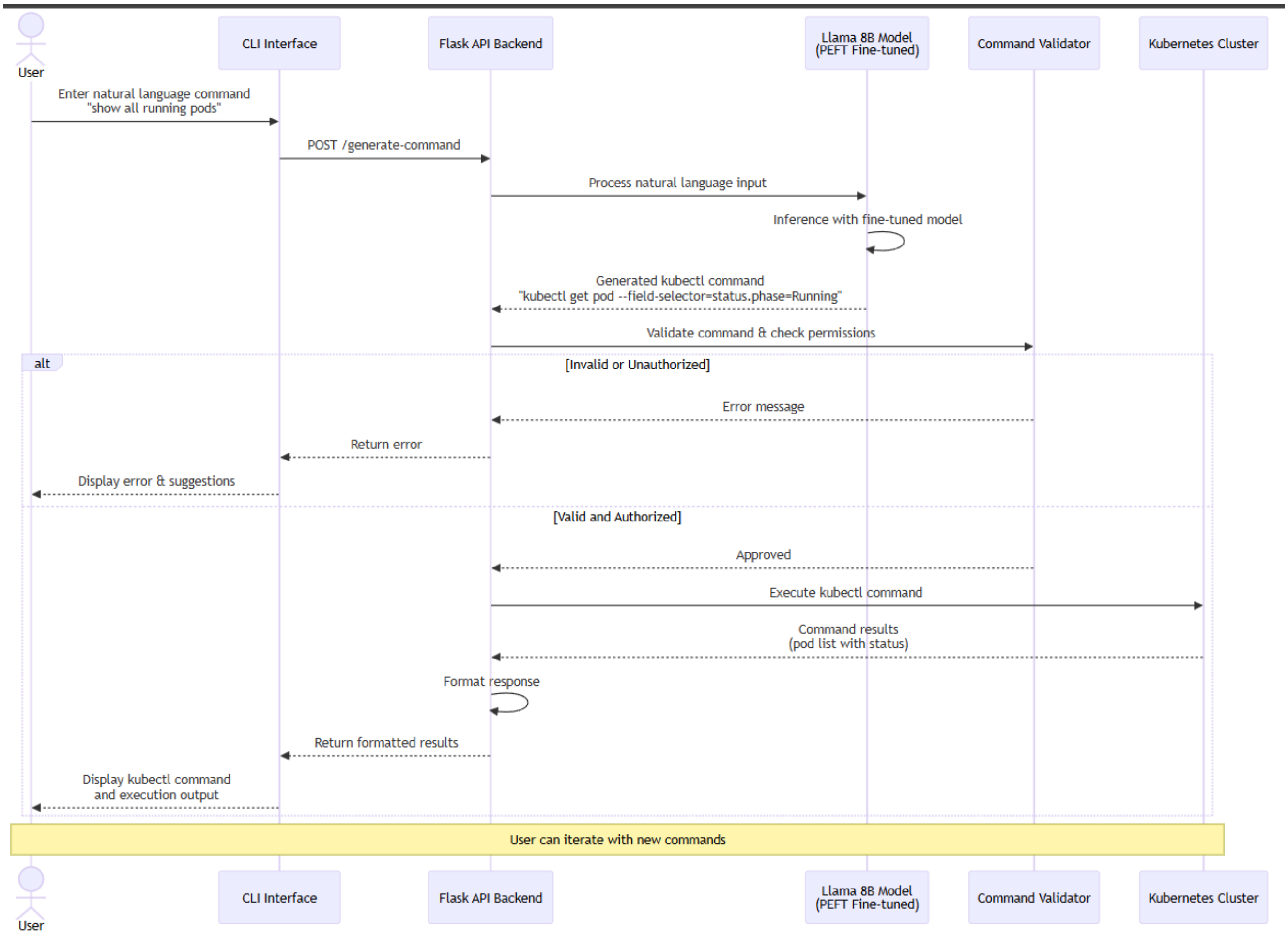

Figure 3: Framework of the project

4. Experiments

4.1. Dataset Description

The dataset used for training and evaluation in this work is the **ComponentSoft/k8s-kubectl-35k** dataset, sourced from Hugging Face. It comprises approximately 35,000 pairs of natural language (NL) commands and their corresponding Kubernetes (`kubectl`) command-line interface (CLI) equivalents.

Each entry consists of:

- A natural language instruction, e.g., *"Create nginx deployment"*
- A target command, e.g., `kubectl rollout restart deployment frontend`

The dataset includes diverse categories of Kubernetes operations such as:

- **Pod management:** creating, deleting, listing, and describing pods.
- **Deployment operations:** rolling updates, scaling, and restarts.
- **Service configuration:** exposing, port-forwarding, and inspecting services.
- **Cluster and node management:** managing contexts, nodes, and namespaces.

A subset sample of the dataset is shown below.

Table 2: Sample of the ComponentSoft/k8s-kubectl-35k Dataset

ID	Natural Language Query	Target Kubernetes Command
001	restart the frontend pod	`kubectl rollout restart deployment frontend`
002	show all running pods in the default namespace	`kubectl get pods -namespace=default`
003	delete the nginx pod	`kubectl delete pod nginx`
004	scale backend deployment to 3 replicas	`kubectl scale deployment backend -replicas=3`
005	list all namespaces	`kubectl get namespaces`

4.2. Preprocessing and Data Cleaning

Before fine-tuning the models, the dataset underwent the following preprocessing steps:

1. **Normalization of Text:** All queries were converted to lowercase, and punctuation inconsistencies (e.g., extra commas or dots) were removed.
2. **Tokenization Compatibility:** Special characters such as `-`, `/`, and `-` were preserved to ensure accurate mapping to Kubernetes CLI syntax.
3. **Deduplication:** Duplicate entries and trivial command variants were filtered out to avoid overfitting.

4.3. Model Configurations

4.3.1. LLaMA-3.1-8B-Instruct

The **LLaMA-3.1-8B-Instruct** model, known for its strong instruction-following capability, was fine-tuned on the curated dataset using the LoRA (Low-Rank Adaptation) technique to reduce GPU memory usage.

- Model Size: 8 billion parameters
- Training Batch Size: 4
- Learning Rate: 2e-5
- Epochs: 3
- Optimizer: AdamW
- Training Platform: NVIDIA H100 GPU SXM (80 GB VRAM)
- Precision: FP16 mixed precision

4.3.2. T5-Base

The **T5-Base** model (220M parameters) was selected for its compact size and flexibility for local usage on devices with limited memory (around 1 GB VRAM).

- Model Size: 220M parameters
- Tokenizer: SentencePiece tokenizer
- Learning Rate: 3e-5
- Batch Size: 8
- Epochs: 4
- Framework: PyTorch with Hugging Face Transformers
- Loss Function: Cross-entropy loss

4.4. Training Procedure

The fine-tuning followed a supervised sequence-to-sequence paradigm:

- **Input:** Natural language query (*"Get detailed information about the backend pod."*)
- **Output:** Corresponding `kubectl` command (e.g., `kubectl rollout restart deployment frontend`)

4.5. Evaluation Metrics

The models were evaluated using the following quantitative metrics:

- **BLEU score** – measures n-gram overlap between predicted and reference command.
- **ROUGE-L** – captures sequence-level similarity.
- **Exact Match (EM) accuracy** – fraction of perfectly matched commands.
- **Edit Distance (Levenshtein Distance)** – measures structural closeness.

5. Results and Discussion

5.1. Quantitative Results

The overall comparison between **LLaMA-3.1-8B-Instruct** and **T5-Base** is shown in Table 3.

Table 3: Quantitative comparison between LLaMA-3.1-8B-Instruct and T5-Base

Model	BLEU	ROUGE-L	Exact Match (%)	Edit Distance ↓	Size
LLaMA-3.1-8B	0.76	0.95	90%	1.0-2.0 tokens	25 GB
T5-Base	0.602	0.965	85%	2.0-3.5 tokens	800 MB

Observation:

LLaMA-3.1-8B-Instruct achieved higher overall accuracy and fluency due to its larger parameter count and stronger contextual understanding, while T5-Base performed competitively despite its smaller size demonstrating high local deployability and inference efficiency for real-time Kubernetes assistance.

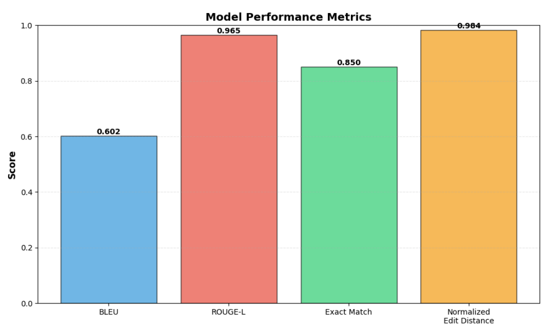

Figure 4: T5 Model Metrics

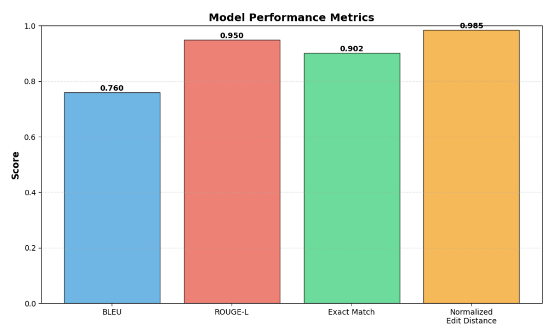

Figure 5: Llama-3.1-8B-Instruct Model Metrics

5.2. Visual Analysis

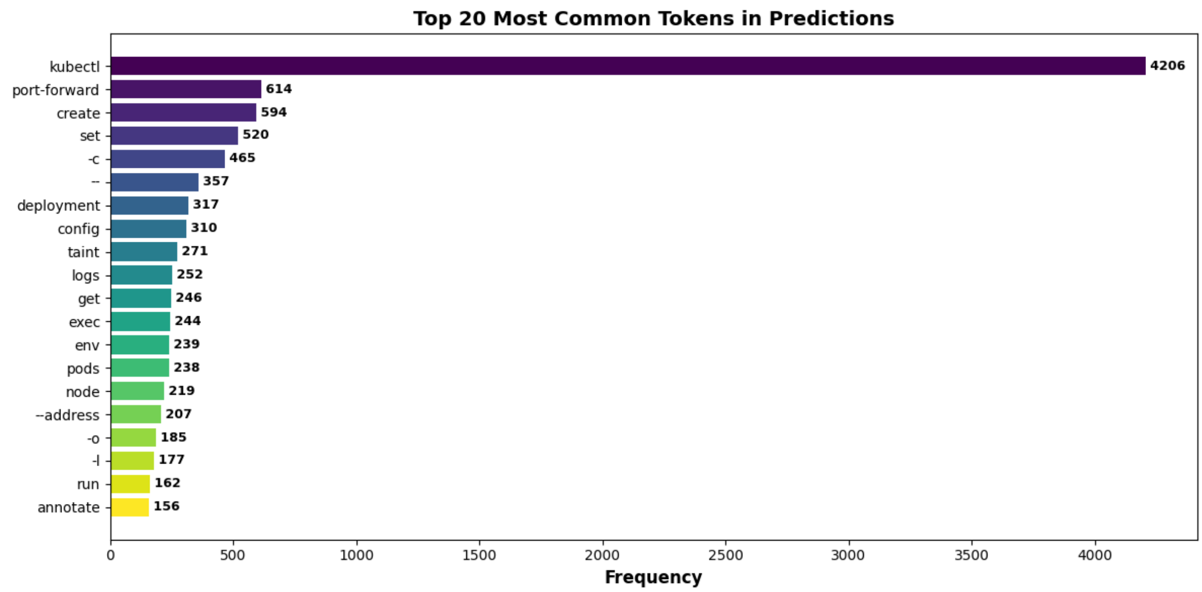

Figure 6: Top 20 Most Common Tokens in Prediction

Epoch	Training Loss	Validation Loss
1	0.057900	0.044453
2	0.038000	0.035293
3	0.031700	0.031808
4	0.023200	0.030555

Figure 7: T5-Base Training

Step	Training Loss	Validation Loss
50	2.20	2.30
100	1.85	1.90
150	1.60	1.65

Figure 8: Llama-3.1-8B-Instruct Training Loss

5.3. Qualitative Observations

A few example predictions from both models are summarized below.

Analysis:

- LLaMA's larger context window captured longer dependencies, producing highly accurate commands with correct flags and namespaces.
- T5 sometimes missed CLI flags (e.g., `-replicas`, `-namespace`) but still produced syntactically close commands, making it suitable for edge or offline tools where full 8B model inference is impractical.

Table 4: Sample qualitative predictions from both models

Natural Language Input	LLaMA-3.1-8B Prediction	T5-Base Prediction	Reference Command
restart frontend pod	`kubectl rollout restart deployment frontend`	`kubectl restart pod frontend (minor syntax variation)`	`kubectl rollout restart deployment frontend`
show all pods in kube-system	`kubectl get pods -n kube-system`	`kubectl get pods -namespace kube-system`	`kubectl get pods -n kube-system`
scale backend to 5 replicas	`kubectl scale deployment backend -replicas=5`	`kubectl scale deploy backend 5 (incomplete flag)`	`kubectl scale deployment backend -replicas=5`

5.4. Interface and Outputs

The fine-tuned model was integrated into a web-based interface (built using an **Express.js** backend and **React** frontend), where users can input natural language instructions and receive the corresponding Kubernetes command output in real time.

- **Backend:** Handles model inference and API calls.
- **Frontend:** Provides a user-friendly input box and command preview pane.
- **Deployment:** AWS GPU Instance / Runpod GPU pods.

5.5. Discussion

The experiments confirm that large instruction-tuned models such as **LLaMA-3.1-8B** can be successfully fine-tuned to serve as Kubernetes command assistants, enabling developers and DevOps engineers to operate clusters via natural language. However, due to the computational demand of LLaMA-3B, smaller models like **T5-Base** remain practical for on-premise or local setups.

Key Takeaways:

- **LLaMA-3.1-8B** achieved superior command generation accuracy, especially for multi-flag commands.

- **T5-Base** offers high portability and real-time inference speed suitable for lightweight integrations.

- The **ComponentSoft/k8s-kubectl-35k** dataset effectively bridges the semantic gap between natural and operational languages.

6. Conclusion and future scope

In this work, we explored the task of translating natural-language queries into kubectl commands using two pretrained language model architectures: the large instruction-tuned LLaMA-3.1-8B-Instruct and the more compact T5-Base (220 M parameters). We fine-tuned both models on the domain-specific dataset ComponentSoft/k8s-kubectl-35k comprising approximately 35 k natural-language → kubectl pairs, and compared their performance across standard metrics. Our results demonstrate that LLaMA-3.1-8B achieved superior accuracy and command correctness in most categories, especially for complex queries involving flags, namespaces, and combined operations. Meanwhile, T5-Base delivered competitive performance although lower absolute accuracy but offers a much smaller footprint, making it highly suitable for local on-premise deployment or resource-constrained environments.

Taken together, our findings suggest that fine-tuned instruction-models can effectively serve as Kubernetes CLI assistants, reducing the cognitive burden on DevOps practitioners and enabling natural-language interfaces to cluster operations. Moreover, the use of smaller models like T5-Base illustrates a credible trade-off: slightly reduced accuracy in exchange for improved deployment feasibility in edge or offline contexts.

Potential future work includes incorporating reinforcement learning with human feedback (RLHF) to further refine natural command alignment and integrating error correction for invalid command generation.

References

[1] C. Ltd., k8s-kubectl-35k, `https://huggingface.co/datasets/ComponentSoft/k8s-kubectl-35k`, accessed: 2025-08-08 (2025).

[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards,

Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, Journal of machine learning research 21 (140) (2020) 1–67.

[5] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al., Codebert: A pre-trained model for programming and natural languages, arXiv preprint arXiv:2002.08155.

[6] Y. Wang, W. Wang, S. Joty, S. C. Hoi, Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, arXiv preprint arXiv:2109.00859.

[7] X. V. Lin, C. Wang, L. Zettlemoyer, M. D. Ernst, Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system, arXiv preprint arXiv:1802.08979.

[8] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The llama 3 herd of models, arXiv e-prints (2024) arXiv–2407.

[9] A. Angi, L. Nedoshivina, A. Sacco, S. Braghin, M. Purcell, A perspective on llm data generation with few-shot examples: from intent to kubernetes manifest, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), 2025, pp. 345–354.

[10] Q. Fu, Z. Teng, J. White, D. C. Schmidt, A transformer-based approach for translating natural language to bash commands, in: 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2021, pp. 1245–1248.

[11] Q. Fu, Z. Teng, M. Georgaklis, J. White, D. C. Schmidt, Nl2cmd: An updated workflow for natural language to bash commands translation, arXiv preprint arXiv:2302.07845.

[12] S. Bharadwaj, S. Shevade, Explainable natural language to bash translation using abstract syntax tree, in: Proceedings of the 25th Conference on Computational Natural Language Learning, 2021, pp. 258–267.

[13] Q. Fu, Z. Teng, J. White, D. C. Schmidt, A transformer-based approach for translating natural language to bash commands, in: 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2021, pp. 1245–1248.

[14] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, S. C. Hoi, Codet5+: open code large language models for code understanding and generation (2023), URL https://arxiv. org/abs/2305.07922.

[15] J. Lin, K. Dzeparoska, A. Tizghadam, A. Leon-Garcia, Appleseed: Intent-based multi-domain infrastructure management via few-shot learning, in: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), IEEE, 2023, pp. 539–544.

[16] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288.

[17] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al., Code llama: Open foundation models for code, arXiv preprint arXiv:2308.12950.

[18] I. Jindal, C. Badrinath, P. Bharti, L. Vinay, S. D. Sharma, Balancing continuous pre-training and instruction fine-tuning: Optimizing instruction-following in llms, arXiv preprint arXiv:2410.10739.

[19] T. Zhang, S. Pan, Z. Zhang, Z. Xing, X. Sun, Deployability-centric infrastructure-as-code generation: An llm-based iterative framework, arXiv preprint arXiv:2506.05623.

[20] X. Garcia, Y. Bansal, C. Cherry, G. Foster, M. Krikun, M. Johnson, O. Firat, The unreasonable effectiveness of few-shot learning for machine translation, in: International Conference on Machine Learning, PMLR, 2023, pp. 10867–10878.