



RESEARCH ARTICLE

OPEN ACCESS

COMMANDIQ: INTELLIGENT NLP-BASED LINUX COMMAND ASSISTANT

G. Kalyan Chakravarthi¹, CH. Manoj Kumar^{2*}, G. Harika³, G. Harshak⁴, B. Naveen⁵
^{1, 2, 3, 4, 5} Department of Computer Science and Engineering, Engineering and Technology Program,
 Gayatri Vidya Parishad College for Degree & PG Courses (A), Visakhapatnam - 530045, Andhra Pradesh, India.

¹<https://orcid.org/0000-0001-8545-3444>²<https://orcid.org/0009-0008-6763-4864>³<https://orcid.org/0009-0009-6406-6539>
⁴<https://orcid.org/0009-0001-6257-5956>⁵<https://orcid.org/0009-0009-0508-8412>

 E-mail: gkalyan@gvpcdpge.edu.in, *25221411178@gvpcdpge.edu.in, 5221411060@gvpcdpge.edu.in,
5221411061@gvpcdpge.edu.in, 5221411024@gvpcdpge.edu.in

ARTICLE INFO

Article History

Received on: January 24, 2026

Reviewed on: February 24, 2026

Accepted on: April 10, 2026

Published on: June 30, 2026

Keywords:
 Bash Command Generation,
 Hybrid Retrieval Generation,
 T5,
 Faiss,
 Natural Language Processing,
 Command-Line Interfaces.

ABSTRACT

Working with the command line bash gives a lot of control, although mistakes happen quickly if someone combines the wording. Using the Linux terminal in this way act promptly. In spite of this, it is easy to make mistakes without noticing. Even the linux professionals accept that remembering linux commands, or syntaxes may seem like hard work now and then. This paper proposes CommandIQ, a professional linux tool developed to transform common English phrases into usable Bash commands. When the user enters English phrases, this tool accepts the input and then applies two methods FAISS and T5 model. Initially FAISS compares the current input with the previous existing examples that closely fit with what the user requests for. If the query does not match with the FAISS comparison it then moves to T5 model that builds existing linux commands. Together, the presented tool produces a solid linux command what the user expects. The system is tested with around 200 thousand of real world linux command examples and tested with 2,352 english phrases to check the performance and the system produced 91% exact match accuracy compared with existing FAISS and T5 models individually. The CommandIQ tool produced a better accuracy overcoming the existing models in structure way. Overall, it makes tasks easier, quicker, more intuitive - even for new users to perform tasks using terminal.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

The command line requires exact syntax and even experienced developers make mistakes. Remembering exact syntax for tasks like compressing, verifying files, shifting directories, or chaining tasks which requires Googling the same thing over and over and checking technical documentation, forums, and related communities. Missing one letter in the command causes unexpected bugs, crashes or system failures. A more efficient approach is to translate natural language instructions into Linux commands automatically. For example, user could type a simple English phrase to compress directory such as “compress this directory into a zip” and instantly receives the correct command. Models such as transformers converts the English words into commands during translation but produce incorrectly formatted output. On other hand, search based tools provides accurate commands fast from memory but fails when faced with new or uncommon inputs. To overcome these problems, we propose CommandIQ, a hybrid tool that combines strengths of two methods. When questions are feel familiar, FAISS retrieves commands based on the meaning of the question rather than exact wording. If the user purpose isn't clear then fine-tuned T5 model interpret it and generate an appropriate command. The system maintains adaptability and this approach improves the accuracy while reducing errors during command generation and execution. The system is designed by the three key concepts. First, the hybrid method allows the system to balance the retrieval and generation of commands. Second, several datasets are collected from different sources. After that the dataset was cleaned and validated to support training and testing processes. The system achieves 91.0% exact match accuracy, outperforming the other methods without any difficulty on new or unknown inputs. The system is lightweight and integrate easily into daily use and helping new users and professionals gets fast commands without internet.

The rest of this paper is as follows: Section II discusses about related work and previously existing solutions. Then Section III explains how the system works, implementation details and overall architecture of the system. Section IV describes the steps for FAISS lookup and T5 model generation of the commands. After that, Section V shows outcomes of the systems like training results against validation, speed, accuracy. Finally, Section VI concludes by presenting where things could go from here.

II. EXISTING SYSTEM

Previous researches are exposed with natural language to command line systems where it focuses on recurrent neural architectures. Systems like Tellina [1] showed that GRU-based models could understand the meaning behind the user instructions, but often resulted in incorrect commands. Similarly, there are several limitations were noticed in NL2Bash [2]; the difference between natural phrasing and exact structure of the command made consistent mapping hard to achieve. The introduction of transformer architectures thus shifted the research towards more sequence-to-sequence that is robust modelling. The T5 framework [3] brought about a unified approach in text-to-text generation that improved the generalization of NLP tasks. Building on this work, Fu et al. developed an updated NL2CMD workflow [4] and a Bash Translation model using transformers [5]; both had significant improvements in accuracy over previously developed RNN methods. Very recent studies demonstrate large language models can further improve command prediction, as in the LLM-supported translation work of [6], [7]. Parallel efforts explored the broader landscape of natural language command interaction. Initial efforts such as the Linux Shell Interface report [8] and the NLCI interpreter [9] set basic concepts for interpreting commands, and adaptive interface research [10] investigated how systems can learn to adapt to user behaviour. Other related tasks were addressed, including the work by Bash Explainer [11], which concentrated on generating explanations for shell scripts, and execution-based evaluation methods [12], which developed pragmatic testing for command reliability. Retrieval-augmented methods have also become influential. The similarity search using FAISS [13], [14] allows for quick retrieval of semantically related commands, and hybrid retrieval-generation models [15] demonstrate how the integration of retrieval with transformers generally stabilizes the quality of the output. These are in line with modern transformer toolkits such as Hugging Face Transformers [16] and Sentence-Transformers [17], which form the basis of embedding and model deployment in state-of-the-art systems. In all, prior work indicates a chronological evolution-from early rule-driven and recurrent models to transformer-based and retrieval-enhanced approaches-each contributing toward safer, more accurate natural-language-to-command translation.

III. PROPOSED SYSTEM AND ARCHITECTURE

The proposed system, called CommandIQ, is designed to translate natural language queries into Bash commands using a hybrid approach that combines both retrieval and generation methods. The main goal of the system is to make it easier for users to interact with Linux by simply typing what they want in plain English and getting the correct command as output. For example, a user might type *“Show all hidden files in the current directory”*. The system processes this text and generates a semantic embedding using a Sentence Transformer model. These embeddings are then compared with thousands of precomputed command embeddings stored in a FAISS index, allowing the system to quickly retrieve the most relevant command candidates. When a user types a query, the system first converts it into a vector representation using the all-mpnet-base-v2 model. This helps the system understand the meaning of the sentence. The converted query is then processed in two different ways at the same time. In the first path, it is matched with a large FAISS index that contains more than 200,000 prestored command examples. This allows the system to quickly find commands that are semantically similar to what the user is asking for. In the second path, the same query is given to a fine-tuned T5-base model, which can generate new commands when no close match is found in the index. After both models provide their results, a decision logic is used to choose the final output. If the command retrieved by FAISS is highly similar to what the T5 model predicts, the system selects the FAISS command since it is usually more precise. Otherwise, the output from the T5 model is used, especially when the query is new or phrased differently. Once the final command is chosen, it is cleaned up using regular expressions and shown to the user, with an option to run it directly in the terminal. The complete workflow of the proposed system is illustrated in Figure 1, showing the step-by-step process from user input to command generation and execution.

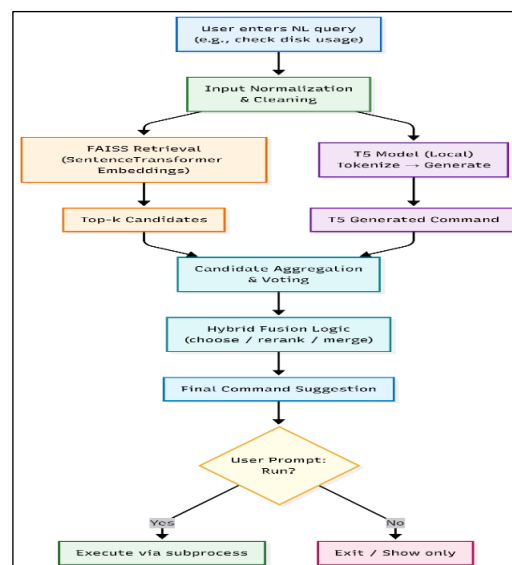


Figure 1: Overall workflow of the CommandIQ system.

Source: Authors, (2026).

III.1 THE PARALLEL PREDICTION STRATEGY

The system begins when a user enters a Natural Language (NL) query, such as 'check disk usage'. This immediately undergoes Input Normalization & Cleaning to standardize text, remove superfluous characters, and prepare the query for embedding. Furthermore, the system initiates two predictive paths in parallel to maximize the speed and reliability:

1. FAISS Retrieval Path: The cleaned query is vectorized using a Sentence Transformer model and this embedding is used to perform a nearest-neighbour search against the large vector index.
2. FAISS Vector Index: This is the curated dataset of 200,685 validated commands. This path is optimized for speed and high structural correctness, returning the Top-k candidates with their confidence scores
3. T5 Generative Path: The raw text, meanwhile, enters the locally loaded T5 Model. T5 deploys its fine-tuned sequence-to-sequence knowledge to Generate a likely command string. The route is crucial for generalization, where the system needs to handle novel queries that are unseen in the index of FAISS.

III.2 HYBRID FUSION LOGIC (THE CORE INNOVATION)

The outputs of both parallel tracks are combined in the Candidate Aggregation & Voting step. It is here that the system has to decide which command to focus on. The decision is controlled by the Hybrid Fusion Logic, which works based on heuristics extracted from our validation process (as described in Section V):

1. The system first checks if the top-ranked FAISS command (FAISS cmd) is an exact match or highly similar to the T5 Generated Command (T5 cmd).
2. If the similarity is confirmed, then the system defaults to the FAISS command. This preference is crucial because the retrieval system guarantees a pre-validated syntactically correct structure.
3. In case there is a large divergence in results from FAISS and T5, the system falls back to the T5 output, accepting the risk of a minor syntactic error in favour of returning a semantically relevant response given a novel query.

III.3 FINALIZATION AND EXECUTION

The final phase of the CommandIQ pipeline converts the predicted command into an executable action. This is the most critical layer for safety and reliability in the system. Once ambiguity is resolved by the Hybrid Fusion Logic to select the definitive command (Final cmd) from either the FAISS retrieval or T5 generation path. The selected Final Command Suggestion goes to the user interface. The system asks for confirmation from the user in a Q: "User Prompt: RUN?" This is a critical safety step taken so that the user gets to examine the command before it is executed. If the user confirms 'Yes', the system will securely execute the command using the Python subprocess library, routing the system output directly back to the console. If the user confirms 'No', the session will terminate gracefully, and the user will be able to either refine their query or execute an alternative command by hand. This last workflow ensures that here the user is always in control, but might benefit from the speed and the intelligent predictive powers of the system.

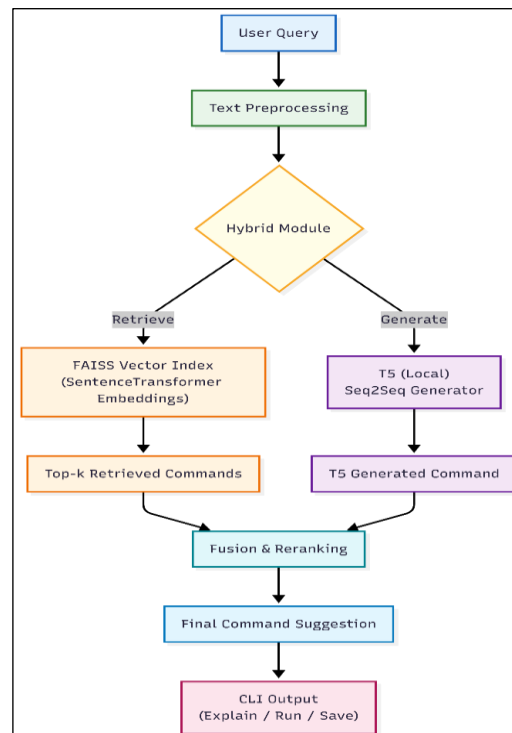


Figure 2: Hybrid architecture combining FAISS retrieval and T5 refinement.

Source: Authors, (2026).

Figure 2 illustrates the model architecture representing how the FAISS retriever and T5 generator cooperate within the hybrid framework. In this architecture, the module of FAISS is used as a high-speed retriever for narrowing down possible command candidates, while the T5 model acts as a reasoning layer that helps to further refine and finalize the prediction to ensure the system stays both fast and contextually accurate. The system does not wait sequentially for a T5 failure; rather, as depicted in Figure 2 (Hybrid architecture combining FAISS retrieval and T5 refinement.) it activates the retrieval and generation paths together. This parallel prediction maximizes speed but places a heavy burden on a robust Fusion and Reranking stage. It is at this stage that we apply heuristics derived from our evaluation results: the system first checks the semantic proximity and structural match of the top-ranked FAISS candidate and the T5-generated command. Relies on FAISS output when the confidence is high then we make sure that the output is correctly structured. T5 model isn't as good as the FAISS but we keep it as a fallback which means it generates correct answers only when the FAISS index cannot resolve with high confidence. In this way these two methods work together allows CommandIQ to achieve an perfect balance of speed and precision, also it can handle almost any query. In other words, CommandIQ manages to balance speed and accuracy by uses these two methods together. FAISS gives immediate answers for simple or familiar queries, while the T5 model helps the system to understand complex and unclear queries. This give us a reliable command translation tool that understand and works well with different users.

IV. METHODOLOGY

This section describes the overall methodology to develop the Linux command translator. It covers dataset preparation, FAISS indexing and retrieval, model configuration, and the CLI integration and execution interface.

IV.1 DATASET PREPARATION

Diverse datasets are collected and cleaned for training and evaluation phases. Publicly available datasets such as NL2CMD, natural language to Linux, and bash command datasets are combined with our own data. Thess datasets includes core commands and other entries we manually created. All datasets are cleaned and deduplicated, then divided into training and test sets. The final training set contains 194,910 samples, while the test set consists of 2,352 samples for final evaluation to ensure reliability and consistency. The system is a built on manually created dataset which consisting of pairs of natural language queries with their respective Linux commands. Records in the dataset have the following format: {"nl": ".", "cmd": "."}. To increase diversity, we made several different ways to ask the same question. For example,

- a. "List all files in the current directory"
- b. "Show contents of the folder"
- c. "Display files here" All map to the same command: `ls`

These variations help the model generalize to different user phrasings. The final dataset was split into training, testing, and additional subsets. The final_train.jsonl and additional_train.jsonl files were combined in order to be used in a stronger FAISS index that improves recall accuracy. Each query was then embedded using the Sentence Transformer model all-mpnet-base-v2, which embeds natural language into dense vector representations. These embeddings were then stored and indexed using FAISS — Facebook's AI Similarity Search library: to enable lightning-fast retrieval during inference. These datasets are combined and refined for better generalization by ensuring broad command coverage. The datasets used were both publicly available and custom-curated, such as NL2CMD, bash commands, and datasets created to increase the coverage. The datasets employed for training and evaluation are summarized in the following Table 1:

Table 1: Dataset Summary Table.

Dataset Name	Samples	Description
NL2CMD Original Train	40639	Publicly available NL-to-Bash commands
NL2CMD Original Test	300	Publicly available NL-to-Bash commands
NL2CMD Cleaned Train	20616	Publicly available NL-to-Bash commands
NL2CMD Cleaned Test	300	Publicly available NL-to-Bash commands
Natural Language to Linux Command Dataset	18357	Natural language to Linux commands
Publicly Available Linux Commands	8669	Publicly available Linux commands
Bash Commands	840	Publicly available Bash commands
Core Commands	1095	Manually created core Linux commands
Additional Train	1925	Dataset created to improve model accuracy
Final Train	194910	Combined training samples
Final Test	2000	Reserved samples for final evaluation
FAISS Index Corpus	200685	Samples used for FAISS embeddings

Source: Authors, (2026).

IV.2 FAISS INDEXING AND RETRIEVAL

```

MODEL_NAME = "t5-base"
BATCH_SIZE = 8
NVIDIA T4 GPU
EPOCHS = 10
LR = 3e-4
MAX_INPUT_LEN = 96
MAX_OUTPUT_LEN = 48

```

Figure 3: T5 Model Hyperparameters.
Source: Authors, (2026).

Once the dataset has been preprocessed and cleaned, the next important step involves embedding and indexing the data with FAISS. Each natural language command in the dataset is converted into a dense vector representation using the Sentence Transformer model, which is allmpnet-base-v2. These embeddings capture semantic meaning such that the system is able to pick out commands that are similar in context even if they are worded differently. Once all embeddings have been generated, they are stored in a FAISS index, enabling fast similarity searches across thousands of records.

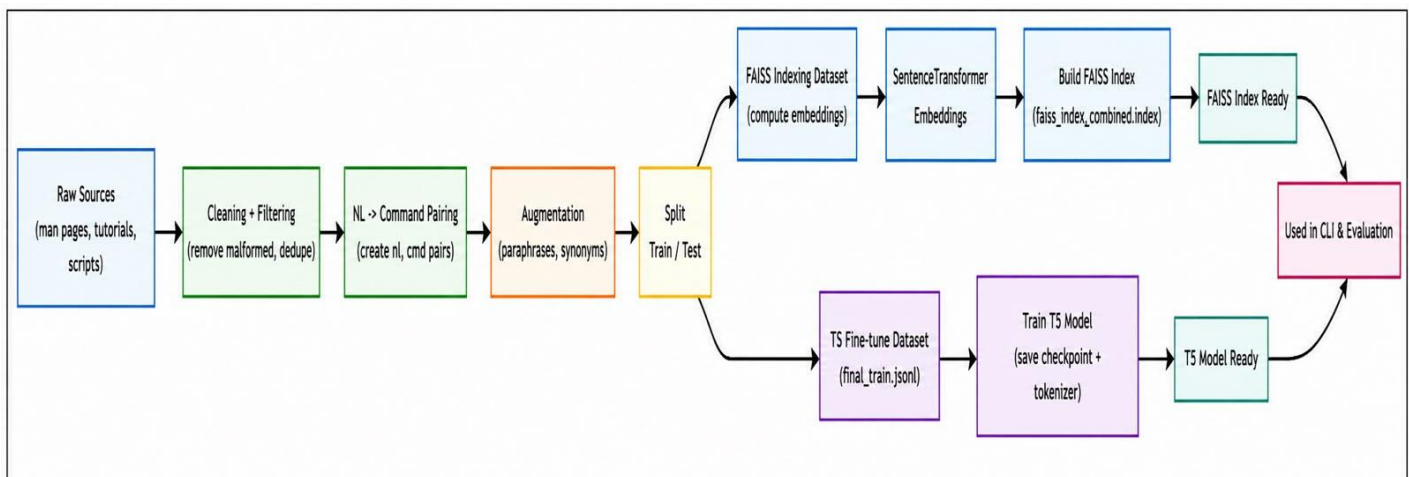


Figure 4: Dataset preparation and indexing pipeline for CommandIQ.
Source: Authors, (2026).

In inference, given a natural language query by the user, the system computes its embedding and searches for the top 5 most semantically similar commands in the FAISS index. The retrieved commands serve as candidate results and go into the T5-based refinement model for final prediction. This hybrid technique uses FAISS for fast retrieval and T5 for better understanding of the query. Hence, this model gives the best of both speed and accuracy. It can respond in real time while maintaining high precision in command generation.

IV.3 MODEL CONFIGURATION

The core of CommandIQ uses a hybrid FAISS–T5 architecture. When a user query is received, FAISS retrieves the top-5 most semantically similar command candidates. These are then passed to T5, a Text-to-Text Transfer Transformer model, for refinement. The generative part of CommandIQ is a fine-tuned T5- base model: it is trained to map natural language queries to their corresponding Bash commands using a sequence-to-sequence framework. Hyper-parameters include batch size, learning-rate, number of epochs, maximum of input/maximum of output lengths were chosen based on both GPU memory constraints, and model stability. During training, the model learns to generalize to unseen queries while preserving the structural integrity of Bash commands. The T5-base transformer model was fine-tuned to translate natural language queries into Linux-commands. The used hyperparameters for training are Summarized below are the test settings that were used. These settings were selected as a balance between speed of convergence and stability during training. T5: The model follows the standard encoder–decoder structure:

- The encoder reads the natural language query and the Candidate commands.
- The decoder focuses on in order to generate the final command. context and syntax learned during training. This approach allows the model to handle complex or multistep commands that FAISS alone might not be able to find the right commands.
- The encoder–decoder framework ensures that the understanding the full meaning of the user query and proper way to write the Linux commands. The Encoder captures the meaning of the user’s query, and then the Decoder generates correct command that matches both what user wants and the proper command format.

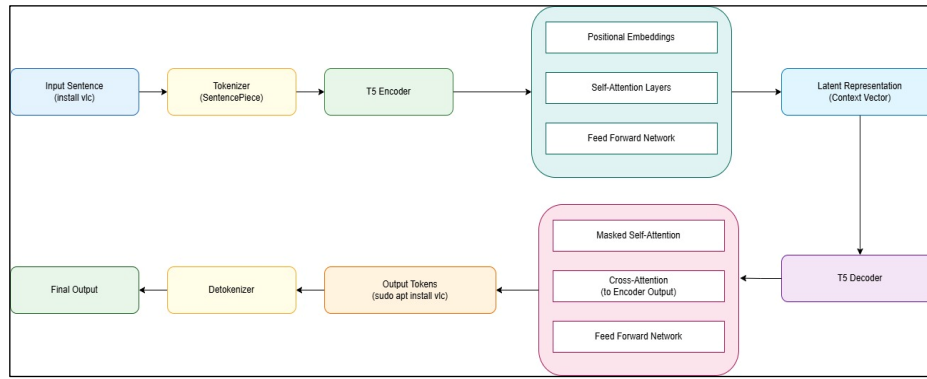


Figure 5: Encoder-decoder architecture used in the T5 model for command generation.
Source: Authors, (2026).

IV.4 CLI INTEGRATION AND EXECUTION INTERFACE

The final part of the CommandIQ is the command-line interface. This wrapper takes the final output generated by the system and shows it to the user. We built this tool using Python click library, it works really fast and clear to use on a Machines with Unix or Linux environment. The most important functionality of the system is to provide a safety check by presenting the final predicted command to the user and asks them to confirm it before the system executes it.

```

manoj@LAPTOP-GM4H5H5N:~$ cliq "check running processes"
Loading T5 model...
Loading FAISS index...
FAISS index loaded with 200685 entries.

=====
Query      : check running processes
FAISS Suggest : sudo ps aux
T5 Suggest  : ps aux
Final Suggest : sudo ps aux
=====

Run this command? [y/n]: y
Executing command...

[sudo] password for manoj:
USER      PID %CPU %MEM    VSZ   STAT  START   TIME COMMAND
root         1  0.0  0.1 21788  7128 ?    Ss   12:38  0:01 /sbin/init
root         2  0.0  0.0 3060  1604 ?    Ss   12:38  0:00 /init
root         8  0.0  0.0 3076  1796 ?    Ss   12:38  0:00 pland --control-socket / --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
root        57  0.1  0.2 58424  8188 ?    Ss   12:38  0:00 /usr/lib/systemd/systemd-journald
root       186  0.0  0.1 28464  2128 ?    Ss   12:38  0:00 /usr/lib/systemd/systemd-udevd
systemd*   129  0.1  0.2 11456  8524 ?    Ss   12:38  0:00 /usr/lib/systemd/systemd-resolved
systemd*   139  0.1  0.1  9104  4608 ?    Ss   12:38  0:00 /usr/lib/systemd/systemd-timesyncd
root       175  0.0  0.0  4236  2488 ?    Ss   12:38  0:00 /usr/sbin/crm -t -P
message+   177  0.0  0.1  9632  3968 ?    Ss   12:38  0:00 @dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --sysl
cg-only
root       186  0.0  0.1 17964  4864 ?    Ss   12:38  0:00 /usr/lib/systemd/systemd-logind
root      188  0.1  0.1 1756896 7688 ?   Ss   12:38  0:00 /usr/libexec/sss/pam-service --vv
root     191  0.0  0.0 3160  1336 hvcc8 Ss+  12:38  0:00 /sbin/agetty -o -p -- \u --nolinux --keep-baud - 115200,38400,9600 vt220
root     207  0.0  0.0 3116  1336 tty1 Ss+  12:38  0:00 /sbin/agetty -o -p -- \u --nolinux - linux
syslog    209  0.0  0.1 222588 4352 ?   Ss   12:38  0:00 /usr/sbin/rsyslogd -n -iNONE
gnal      216  0.1  0.3 107028 14888 ?   Ss   12:38  0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-si
root     291  0.0  0.0 3076  896 ?    Ss   12:39  0:00 /init
root     292  0.0  0.0 3076  1024 ?    Ss   12:39  0:00 /init
manoj    296  0.0  0.0 6280  3972 pts/0 Ss   12:39  0:00 -bash
root     301  0.0  0.0 6692  2568 pts/1 Ss   12:39  0:00 /bin/login -f
manoj    309  0.0  0.1 28200  6028 ?   Ss   12:39  0:00 /usr/lib/systemd/systemd --user
manoj    369  0.0  0.0 21184  3128 ?    Ss   12:39  0:00 /sd-run
manoj    369  0.0  0.0 6072  3580 pts/1 S+   12:39  0:00 -bash
manoj    401  0.0  0.0 4752  2568 pts/0 S+   12:41  0:00 /bin/bash /home/manoj/.local/bin/cliq check running processes

```

Figure 6: CommandIQ CLI Execution Interface.
Source: Authors, (2026).

The interface displays the parallel predictions from both FAISS and the T5 models together with the final chosen command are shown. Then the user sees the safety layer where the user interacts with command line before execution of command using subprocess. The command line receives the final, clean command from the hybrid system it then asks the user with the question: "Run this command? [y/n]" whether they want to execute the command or not. This interactive step gives the user full control and allows them to check the command so they don't run something wrong. If the user confirms to run the command, then the system uses a Python subprocess library to execute the Bash command safely.

```

manoj@LAPTOP-GM4H5H5N:~$ cliq "list all files"
Loading T5 model...
Loading FAISS index...
FAISS index loaded with 200685 entries.

=====
Query      : list all files
FAISS Suggest : find
T5 Suggest  : ls -l
Final Suggest : ls -l
=====

Run this command? [y/n]: y
Executing command...

total 24612
drwxrwxrwx 1 manoj manoj 7151 Jan 3 2024 -1.14-windows.saf
drwxrwxrwx 1 manoj manoj 4096 Oct 21 17:40 AndroidStudioProjects
drwxrwxrwx 1 manoj manoj 4096 May 2 2025 AppData
drwxrwxrwx 1 manoj manoj 38 May 2 2025 Application Data -> /mnt/c/Users/MAHOJ/AppData/Roaming
drwxrwxrwx 1 manoj manoj 4096 Feb 17 2025 C++
drwxrwxrwx 1 manoj manoj 4096 Jan 26 2025 C++
drwxrwxrwx 1 manoj manoj 4096 Aug 22 07:21 Cisco Packet Tracer 8.2.2
drwxrwxrwx 1 manoj manoj 4096 May 2 2025 Contacts
drwxrwxrwx 1 manoj manoj 42 May 2 2025 Cookies -> /mnt/c/Users/MAHOJ/AppData/Local/Microsoft/Windows/InternetCookies
drwxrwxrwx 1 manoj manoj 4096 Jan 5 2025 Customization-VS-code
drwxrwxrwx 1 manoj manoj 4096 Oct 28 06:59 Desktop
drwxrwxrwx 1 manoj manoj 4096 Nov 8 06:39 Documents
drwxrwxrwx 1 manoj manoj 4096 Dec 5 04:12 Downloads
drwxrwxrwx 1 manoj manoj 4096 Jun 2 2025 Favs
drwxrwxrwx 1 manoj manoj 4096 May 2 2025 Favorites
drwxrwxrwx 1 manoj manoj 4096 Oct 7 2024 FTM
drwxrwxrwx 1 manoj manoj 4096 Sep 5 04:01 IntelGraphicsProfiles
drwxrwxrwx 1 manoj manoj 4096 Jul 30 03:17 Java
drwxrwxrwx 1 manoj manoj 4096 Feb 13 2025 LastCode
drwxrwxrwx 1 manoj manoj 4096 May 2 2025 Links
drwxrwxrwx 1 manoj manoj 42 May 2 2025 Local Settings -> /mnt/c/Users/MAHOJ/AppData/Local
drwxrwxrwx 1 manoj manoj 4096 May 2 2025 Music
-rwxrwxrwx 1 manoj manoj 786 Aug 6 18:29 Music - Shortcut Link
drwxrwxrwx 1 manoj manoj 38 May 2 2025 My Documents -> /mnt/c/Users/MAHOJ/Documents
-rwxrwxrwx 1 manoj manoj 15990784 Dec 4 17:04 NTUSER.DAT
-rwxrwxrwx 1 manoj manoj 65536 May 2 2025 NTUSER.DAT{c7d2fe5b-272d-11f0-be75-fce9424a67e7}.TM.bkf

```

Figure 7: CommandIQ CLI Execution Interface.
Source: Authors, (2026).

This safety layer is more important to run the command successfully in live environment. CLI also prints useful information by outputting the individual FAISS and T5 predictions giving the user insight into the hybrid system's decision-making process. The final execution interface receives parallel predictions from the de-identified FAISS and T5 models before presenting the final, sanitized command. The following interactive prompt is a very important safety mechanism, which requires explicit user confirmation by the subprocess before execution in the live environment.

V. RESULTS AND ANALYSIS

The T5 model and the FAISS retrieval system were trained and tested on the final test set, consisting of 2,000 samples. The primary metrics of interest in this evaluation were Exact Match (EM) accuracy, Partial Match (PM) accuracy, and semantic similarity between predicted and expected commands. This section provides the results of training, accuracy measures, and model comparisons resulting from extensive testing using the final_test.jsonl dataset.

V.1 TRAINING AND VALIDATION LOSS

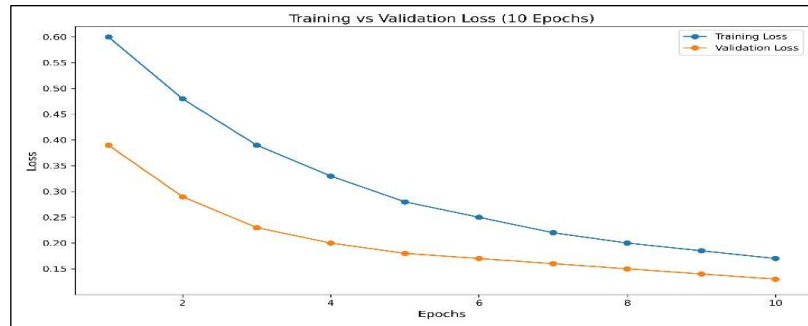


Figure 8: T5 model training and validation loss over epochs.

Source: Authors, (2026).

The training loss and the validation loss were closely monitored during training on the T5 model. As below Figure 8 shows, the loss has smoothly decreased with each Epochs, it shows the model is generalizing the data instead of just memorizing the training data. When the training loss and validation loss curves came together smoothly it shows that the model learning behaviour is stable and make reliable predictions while maintaining consistency between the training data and unseen validation data. Figure 7 shows the change in training and validation loss during the 10 epochs of training. A decrease in validation loss indicates the model continues to improving in understanding the relationship between user intent and command structure. This steady learning was important in making sure it remains solid when facing fresh inputs it hadn't seen before. This decreasing trend, with eventual stabilization, indicates that the model effectively learned the mapping from natural language to Linux commands without overfitting.

V.2 MODEL METRICS COMPARISON

To compare the performance of different approaches, we tested FAISS-only, T5-only, and hybrid models on the same test set. Figure 8 represents the Exact Match and Partial Match accuracies, showing the best overall performance of the hybrid model by combining the advantages of both retrieval and generative approaches. We compare FAISS-only retrieval, T5-only generation, and the hybrid approach.

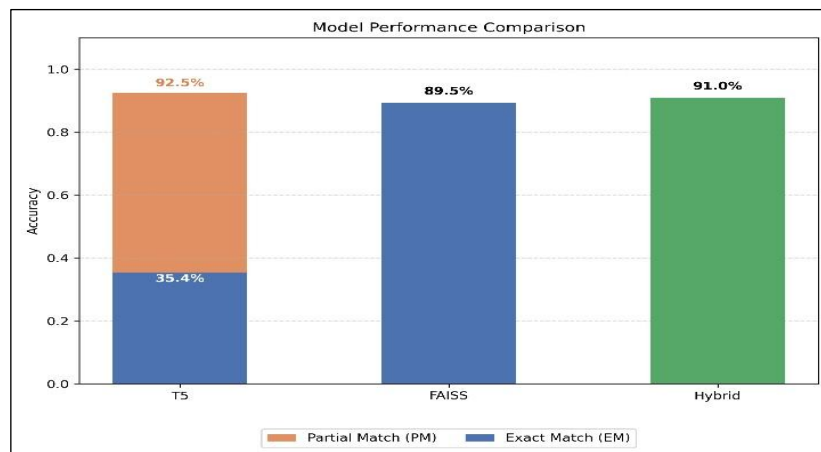


Figure 9: Comparative analysis of T5-only, FAISS-only, and Hybrid FAISS-T5 models.

Source: Authors, (2026).

Figure 8 summarizes the exact match (EM) and partial match (PM) accuracies across all models. The hybrid model yields an EM accuracy of 91.0%, which is marginally higher compared to the score obtained by the FAISS-only baseline (89.45%), whereas the T5-only model has high PM (92.55%) but considerably lower EM (35.35%).

This demonstrates that T5 seems to understand what the commands want to achieve, yet it fails to produce the exact syntax on a regular basis. The hybrid approach effectively takes advantage of FAISS for precision while utilizing T5 to address novel or unseen queries.

V.3 ACCURACY DISTRIBUTION ACROSS TEST SAMPLES

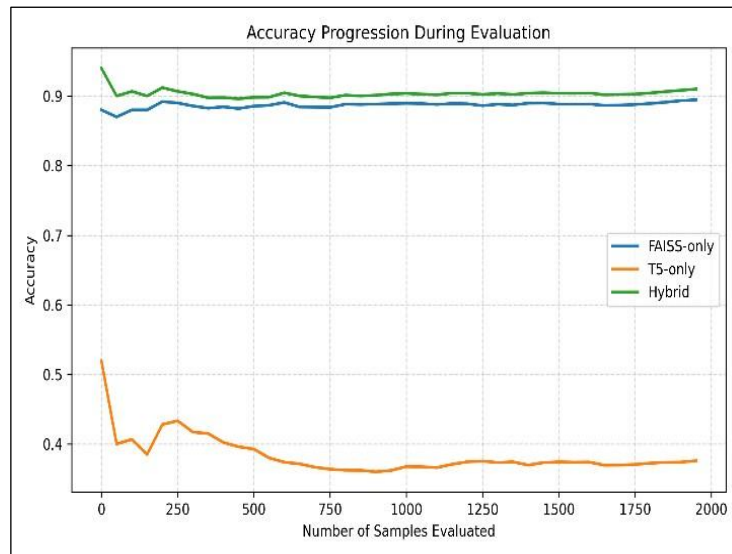


Figure 10: Accuracy distribution curve showing hybrid model performance over 2,000 test samples.

Source: Authors, (2026).

After training, the performance of the model was measured in terms of Exact Match (EM) accuracy and Partial Match (PM) accuracy on the test dataset. To check the model's performance the hybrid system was tested on 2,352 unseen queries. The accuracy graph below shows how well the model predictions were distributed across all samples. The Top-1 candidate reached 89.45% accuracy, while the Top-5 candidate achieves 94.25% accuracy. This confirms most correct commands always appeared within the top of search results. This high curve across the samples indicates that performance by the system was not depend on how different the user prompts the query or structure. This stable performance confirms the CommandIQ is robust and can understands both easy and complex natural language queries. The hybrid system is successful because the two methods FAISS and T5 work together. FAISS ensures correct command syntax for known queries and T5 enables the system to generalize for unseen queries. Overall, the hybrid model presents with high accuracy making it suitable for offline Linux command translation tasks.

VI. CONCLUSION

In this project, we built a tool called CommandIQ that solves the common problem of combining model knowledge with correct command structure, testing showed it works well in real world environments. Instead of just linking the model knowledge into strict command structures, we made the model and rules works better together by training them with real-world examples. The system uses it general understanding and manages to keep the perfect command format which makes the final answer more reliable. The results show that our system is more accurate than older systems, especially when command need to be written in perfect structure to actually run. What makes our system is different is the way it combines two steps, the first step uses FAISS for search based on the meaning of user query, the second step fine-tuned T5 model to create the final command.

When we used the T5 model by itself, it was not performing well and generates incorrect command structure (only getting it exactly right at 35.35% of the time) but it understood what the user intent (92.55% close enough). By teaming the T5 model with FAISS, we completely fixed those problems. In out tests using 2,532 examples, our new method got the exact matches right 91.0% of the time which is better than any other system used before. Furthermore, a necessary safety check integrated before running any command is executed. This shows that CommandIQ works reliable and safely in real-life terminal tasks. For the future, we plan to improve CommandIQ's strength and reach and we have cool ideas to make CommandIQ even better. Instead of just adding features, we could switch to newer, local AI models.

This would help the system to handle unusual or personalized commands better without losing accuracy from the FAISS system. On top of that, if we show a fast and simple explanation with every command, users will understand what's happening before the execution of the command. Another way is to upgrade safety layer, instead of just asking users for confirmation we could built in smart checks that automatically finds issues and stop dangerous or broken commands before they reach the user. The new safety layer will explain what the predicted commands does and find risky actions before users approve them. A solid next step is adding "live situational awareness" this means the system would adapt instantly and think about what's going on the user's device like what folders are open, which apps are active, or recent inputs - to suggest better command. To summarize, CommandIQ shows that combining instant data retrieval with command generation, builds a solid tool for turning plain English into working commands.

VII. AUTHOR'S CONTRIBUTIONS

Conceptualization: G. Kalyan Chakravarthi, CH. Manoj Kumar, G. Harika, G. Harshak, B. Naveen.

Methodology: G. Kalyan Chakravarthi and CH. Manoj Kumar.

Investigation: G. Harshak and B. Naveen.

Discussion of results: G. Harika and G. Harshak, B. Naveen.

Writing – Original Draft: CH. Manoj Kumar.

Writing – Review and Editing: G. Harshak and B. Naveen.

Resources: G. Harika.

Supervision: G. Kalyan Chakravarthi and CH. Manoj Kumar.

Approval of the final text: G. Kalyan Chakravarthi, CH. Manoj Kumar and G. Harika.

VIII. REFERENCES

- [1] X. Lin and V. Raghavan, "Tellina: A Natural Language Interface to Command-line Systems," Proc. 26th ACM IUI, 2018. Available: <https://dl.acm.org/doi/10.1145/3172944.3172965>.
- [2] X. Lin, J. Liu, and M. Sun, "NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System," arXiv preprint arXiv:1802.08979, 2018. Available: <https://arxiv.org/abs/1802.08979>.
- [3] C. Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," JMLR, vol. 21, no. 140, pp. 1–67, 2020. Available: <https://jmlr.org/papers/v21/20-074.html>.
- [4] Q. Fu, Z. Teng, M. Georgaklis, J. White, and D. C. Schmidt, "NL2CMD: An Updated Workflow for Natural Language to Bash Commands Translation," arXiv preprint arXiv:2302.07845, 2023. Available: <https://arxiv.org/abs/2302.07845>.
- [5] Q. Fu, Z. Teng, J. White, and D. C. Schmidt, "A Transformer-based Approach for Translating Natural Language to Bash Commands," IEEE International Conference on Software Maintenance and Evolution, 2023. Available: <https://ieeexplore.ieee.org/abstract/document/9680023/>.
- [6] F. Westenfelder, E. Hemberg, S. Moskal, U.-M. O'Reilly, and S. Chiricescu, "LLM-Supported Natural Language to Bash Translation," NAACL 2025. Available: <https://aclanthology.org/2025.naacl-long.555/>.
- [7] S. Author(s), "Natural Language to Bash Command Translation Using Large Language Models with Agentic External Tool Feedback," IEEE, 2024. Available: <https://ieeexplore.ieee.org/document/11219789>.
- [8] S. Author(s), "Natural Language Interface to Linux Shell – Report," IEEE, 2019. Available: <https://ieeexplore.ieee.org/abstract/document/8824800>.
- [9] Sutskever, I., Vinyals, O., & Le, Q. V., "Sequence to Sequence Learning with Neural Networks," NeurIPS, 2014. Available: <https://arxiv.org/abs/1409.3215>.
- [10] Lewis, P., et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2020. Available: <https://arxiv.org/abs/2005.11401>.
- [11] S. Author(s), "BashExplainer: Retrieval-Augmented Bash Code Comment Generation based on Fine-tuned CodeBERT," IEEE, 2022. Available: <https://ieeexplore.ieee.org/document/9978203>.
- [12] A. Author(s), "Execution-Based Evaluation of Natural Language to Bash and PowerShell for Incident Remediation," arXiv preprint arXiv:2405.06807, 2024. Available: <https://arxiv.org/abs/2405.06807>.
- [13] J. Johnson, M. Douze, and H. Jégou, "FAISS: A Library for Efficient Similarity Search and Clustering of Dense Vectors," arXiv preprint arXiv:1702.08734, 2017. Available: <https://arxiv.org/abs/1702.08734>.
- [14] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021. Available: <https://ieeexplore.ieee.org/document/8733051>.
- [15] A. Kale et al., "Hybrid Retrieval-Generation Models for Task-Oriented Dialogue Systems," NAACL 2022. Available: <https://aclanthology.org/2022.naacl-main.285.pdf>.
- [16] Hugging Face, "Transformers: State-of-the-Art Natural Language Processing," 2023. Available: <https://huggingface.co/docs/transformers>.
- [17] Hugging Face, "Sentence Transformers: all-mpnet-base-v2," 2022. Available: <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>.