Local LLMs for NL2Bash: A Large-Scale Open-Source Model Evaluation for Bash Command Generation

Jef Jacobs
DistriNet, KU Leuven
3001 Leuven, Belgium
jef.jacobs@kuleuven.be

Jorn Lapon
DistriNet, KU Leuven
3001 Leuven, Belgium
jorn.lapon@kuleuven.be

Vincent Naessens
DistriNet, KU Leuven
3001 Leuven, Belgium
vincent.naessens@kuleuven.be

***Abstract*—Large Language Models (LLMs) are increasingly used as autonomous agents in domains such as cybersecurity and system administration. The performance of these agents depends heavily on their ability to interact effectively with operating systems, often through Bash commands. Current implementations primarily rely on proprietary cloud-based models, which raise privacy and data confidentiality concerns when deployed in real-world environments. Locally hosted open-source LLMs offer a promising alternative, but their performance for such tasks remains unclear.**

This paper presents an empirical evaluation of 22 open-source language models (ranging from 1B to 32B parameters) on Natural Language–to–Bash translation tasks. We introduce an improved scoring system for assessing task success and analyze performance under 10 distinct prompting techniques. Our findings show that Qwen3 models achieve strong results in NL2Bash tasks, that role-play prompting significantly benefits most models, and Chain-of-Thought and RAG can surprisingly hurt local model performance if not carefully designed. We further observe that the impact of prompting strategies varies with model size.

I. Introduction

Large language models (LLMs) have rapidly transformed how complex tasks are approached across domains such as code generation, natural language processing and mathematical reasoning. Today's most capable frontier models, such as ChatGPT [1] and Gemini [2], are typically operated as cloud services by hyperscalers, offering high performance, but requiring data transmission. In parallel, a growing ecosystem of open-source models can be deployed locally on commodity hardware, offering critical benefits in privacy, transparency and controllability, including direct access to system prompts and runtime behavior. However, these deployable models are generally smaller in scale and often exhibit lower accuracy and robustness than their hyperscaler counterparts.

Workshop on LLM Assisted Security and Trust Exploration (LAST-X) 2026
27 February 2026, San Diego, CA, USA
ISBN 978-1-970672-05-3
https://dx.doi.org/10.14722/last-x.2026.23049
www.ndss-symposium.org

One of the most impactful emerging applications of LLMs is their use as autonomous agents that are capable of interpreting natural language instructions and interacting with computing environments. In particular, the ability to translate natural language into Bash commands is fundamental for automating workflows such as system configuration, data processing and penetration testing. In a cybersecurity context, for example, such agents can perform automated vulnerability assessments, exploit misconfigurations or attempt privilege escalation [3], [4], [5], [6]. However, the success of these operations critically depends on the precision and reliability of the generated shell commands: even small errors can compromise task execution or introduce operational risks.

Deploying hyperscaler-hosted frontier models for such sensitive tasks, is often infeasible due to strict requirements of data confidentiality, intellectual property protection and regulatory compliance. Consequently, there is growing interest in locally deployable LLMs that operate fully on-premise. Yet, selecting an appropriate model remains challenging: platforms like Hugging Face [7] host hundreds of thousands of models with diverse architectures and parameter ranges, and existing deployable models often underperform in producing robust Bash commands [3], [6]. Careful prompt design, for example, providing structured instructions or in-context knowledge, is therefore essential to improve their performance.

Despite the importance of open-source language models, studies primarily assess frontier models. Systematic studies of how locally deployable LLMs perform on natural language-to-Bash translation tasks, and how different prompting strategies affect their effectiveness in realistic Linux environments, remain limited. This gap leaves practitioners without clear guidance on which models to select given their available hardware, how to prompt them, or how to measure their performance reliably.

In this work, we address this gap with a large-scale empirical study focused on open-source LLMs, constraining the tested models to run on a single consumer GPU (with 32GB VRAM). Our main contributions are as follows:

- We present an empirical evaluation of 22 locally deployable LLMs ranging from 1B to 32B parameters, assessed

under 10 distinct prompting strategies in an interactive Linux environment.
- We propose an improved scoring methodology for evaluating natural language-to-Bash translation, enabling more accurate and fine-grained performance comparisons.
- We improve existing datasets by adding realistic incorrect samples, improving equivalence threshold calibration and facilitating a comparison with static and dynamic weighted evaluation.
- We provide actionable insights and guidance for prompt selection and model choice, aimed at developers building LLM-based agents for real-world system automation and cybersecurity tasks.

This paper is structured as follows. Section II discusses related work, which evaluates language models and their abilities to solve tasks in a Bash environment. Section III describes our test methodology, including model and prompt selection, as well as the experimental setup. Section IV presents the results and impact analysis of model characteristics and prompting techniques on task success rate. Finally, we discuss our recommendations for model selection and prompt design in Section V.

II. Related Work

Translating natural language into Bash commands is a challenging problem, mainly due to the inherent ambiguity of language and the multitude of commands that can achieve an equivalent outcome. Initial approaches evaluated generated commands by comparing them directly against a single reference (the so-called "gold" command) [8], this approach fails to capture the fact that many syntactically different commands can be functionally equivalent.

To address this, recent research shifted towards execution-based evaluation, where commands are run in a controlled environment and their success is measured based on resulting system changes [9], [10], [11], [12]. InterCode [12] introduced a benchmark for Bash, SQL and Python tasks that combines command output similarity, file system tracking and checksum validation of file modifications in a Docker container. File system changes were monitored using Git, which recorded added, modified or deleted files within predefined test directories. However, this approach provided only partial coverage of file system changes, missing modifications outside tracked paths as well as metadata changes such as ownership, group or permission updates. Furthermore, its statically weighted scoring system fails to reflect whether a generated command achieved the intended task objective.

InterCode-ALFA [11] improved and expanded the dataset to 300 tasks, and evaluated functional equivalence heuristics (FEH) by incorporating a second gold command to each task. The authors determine the success of a FEH by constructing a confusion matrix, treating the second gold command as the positive sample, while a command from ten rows ahead in the dataset, serves as the negative sample. This allowed for a more nuanced comparison of solutions, but it continued to rely on Git-based tracking, and retained many scoring limitations.

TABLE I
Overview of models used in related studies

Paper	# OSS models	Param range (B)	# Technique
InterCode	7	13 - 70	4
InterCode-ALFA	7	0.5 - 8	5
Vo et al.	6	7 - 34	3
Cao et al.	0	-	1
Ours	22	1 - 32	10

Their evaluation covered 11 models, seven of which were open-source with up to 8B parameters.

Vo et al. [10] proposed an alternative setup using Podman [13] containers and *podman diff* in combination with a verify-script to assess the result. However, this method still fails to capture all file system modifications. Additionally, their pipeline lacked capabilities to reset the environment, requiring an additional cleanup-script. Their study assessed six open-source models from three families, ranging from 7B to 34B parameters.

Cao et al. [9] adopted a more task-specific approach, pairing 150 Bash tasks with dedicated validation scripts. While this approach can validate whether the task's intended outcome is achieved, it relies on creator capabilities to assess all possible correct solutions, and may miss unexpected side effects such as modifications outside the task scope, and thus provides only a partial view of command correctness. The method was evaluated only on a single frontier model (GPT-4).

Despite these contributions, prior work remains limited in two important aspects. First, most studies focus on a few open-source models and prompting techniques, as summarized in Table I. Second, current evaluation strategies fail to account for unexpected side effects or the specific intent of the task. By limiting the file system monitoring scope to the directories under evaluation, they create blind spots in other areas, such as critical system locations.

Our work addresses these gaps by covering 22 deployable, open-source models ranging from 1B to 32B parameters. We further assess performance across 10 prompting strategies and introduce improved file system monitoring and task-aware scoring. This broader evaluation scope enables more robust conclusions about model selection and prompt design for real-world, on-premise Bash automation.

III. Methodology

In this section, we describe the applied methodology to evaluate the capabilities of open-source language models. First, we describe the models selected in our evaluations. Secondly, we define the prompting techniques that were applied in this study to increase successful task completion. Then, we describe the modifications made compared to prior benchmarks to facilitate our large scale evaluation. Lastly, we discuss and validate a new evaluation system to provide accurate test results. The implementation and prompts can be found on our GitHub project.[1]

[1]GitHub repository https://github.com/JefJacobs00/Local-LLMs-4-Bash.

A. Open-Source Language Model Selection

Hugging Face [7] is a platform where model creators can publish models, allowing others to further fine-tune them for specific use cases. Since Bash is a text environment, we select models that are capable of text generation tasks. At the time of writing, the platform contains over two hundred thousand text-generation models of varying sizes. We constrain our study to language models that use a single consumer hardware device for inference, specifically an NVIDIA RTX 5090. This limits the VRAM available for the model weights and context to 32 GB. This restriction limits the models we can assess based on the number of parameters. We apply quantization to reduce the precision of model weights, allowing for more parameters at the cost of lower numerical accuracy. By constraining the available VRAM, we impose a practical limitation on the tested models and their weight precision, enabling us to assess the trade-off between parameter quantity and numerical accuracy.

To ensure our evaluation contains relevant language models, we assess which open-source models are commonly used. We estimate model popularity based on two metrics:

- **inference**, indicating how much a model is used for inference, based on the average number of daily downloads; and
- **fine-tuning**, reflecting how often a model is fine-tuned, measured through the number of child models in its Hugging Face model tree.

The average-daily-downloads metric does not perfectly represent real-world model usage (e.g., it omits offline use). Nevertheless, it provides a popularity approximation and offers a more grounded selection criterion than the ad-hoc model choices often seen in related work.

For both inference and fine-tuning metrics, we identified the top 50 models over the two categories. The 30 highest-ranking models were selected for further screening, and we removed models that did not meet the following criteria:

1) *Models under 1 billion parameters*: Models below this threshold generally lack the reasoning capabilities required for Bash tasks, while offering modest gains in inference efficiency on our target hardware;
2) *Base models*: These models have not been tuned to follow instructions, instead they are trained for text completion. This results in prompt continuation rather than a direct answer; and
3) *Vision-Language models*[2] *with a text-only version*: Since our benchmark does not utilize visual processing capabilities, evaluating these variants would introduce unnecessary redundancy.

Our model-selection process initially yielded 23 candidates. The Gemma-3-1B model was excluded from the final analysis due to performance issues with vLLM [14] on our hardware, resulting in a final set of 22 models for evaluation.

Table II lists the selected models per family with their respective sizes. Our evaluation primarily includes models instruction-tuned by their original creator (Google, Meta, Microsoft and Qwen). The selection also contains five specialized models that have undergone additional fine-tuning. Four of these were fine-tuned by DeepSeek through a distillation from their R1 model. The other specialized model has been fine-tuned specifically for coding (Qwen2.5-Coder-7B). This model selection allows us to assess the impact of specialization and distillation on task success rates.

Table III summarizes model size groupings. Each model except Phi 4 has a 32 thousand token context window. Since the full weights of medium-large and large models exceed our available hardware memory, we applied quantization to make inference feasible. The medium-large models were evaluated at FP8 precision, while large models used w4a16 quantization (4-bit model weights and 16-bit attention precision).

TABLE II
OVERVIEW OF BASE MODEL FAMILIES INCLUDED IN THE EVALUATION

Publisher	Model Family	Sizes (Billion Parameters)
Google	Gemma3 [15]	4, 12, 27
Qwen	Qwen2.5 [16]	1.5, 3, 7, 14, 32
	Qwen2.5-Coder [17]	7
	Qwen3 [18]	4, 8, 14, 32
Meta	Llama3.1 [19]	1, 3
	Llama3.2 [19]	8
DeepSeek	Qwen2.5 [20]	1.5, 7, 14, 32
	Llama3.2 [20]	8
Microsoft	Phi 4 [21]	14

TABLE III
MODEL DISTRIBUTION BY SIZE GROUP

Size Group	Number of Models	Parameter Range
Tiny	3	1 to 2 billion
Small	4	3 to 4 billion
Medium	6	7 to 8 billion
Medium-Large	5	12 to 14 billion
Large	4	27 billion and more
Total	**22**	

B. Datasets

Our evaluation leverages the same test set as InterCode-ALFA, as discussed in Section II. In addition, we curate two new datasets to be used with Retrieval Augmented Generation (RAG):

- **Environment Man Pages**: This dataset contains 981 manual pages extracted from the available tools in our test environment. This dataset contains 91% of the tools used by the gold commands in the test set.
- **NL2Bash Reasoning**: Containing the NL2Bash dataset [22] enriched reasoning traces created with Gemini [2]. Illustrating how a state-of-the-art model rationalizes the selection of specific command-line tools.

For the creation of reasoning traces with Gemini, we remove examples from the NL2Bash reasoning dataset with tasks similar to tasks present in the test set, to ensure the examples do not

[2]Vision-Language model: a model trained to understand images and text

contain the solution for a given task. We remove similar tasks by embedding the task description and comparing them with the task descriptions present in our test set. We embed the task descriptions with all-MiniLM-L6-v2 [23] and remove tasks that exceeded the empirically determined similarity threshold of 0.85. The final dataset contains a total of 9,812 examples with 7,177 unique natural language tasks, the duplicate tasks contain a different reasoning.

C. Prompting Techniques

In addition to assessing model performance, we evaluate the impact of different prompting techniques. These techniques aim to improve task completion either by enriching the model's context with relevant knowledge or by guiding how the model approaches problem-solving. Our study includes prompting techniques that target different mechanisms for improvement.

The prompting techniques are applied to the first user prompt, which provides the model with general information and the task description. We leverage a simple translation prompt (STP) as a baseline. This prompt requests the models to translate the natural language into a Bash command, similar to the base prompt from InterCode-ALFA [11]. We compare this baseline to an extended base prompt (EBP). This prompt adds extra context (e.g., no scripts allowed) and does not constrain the models to respond with a single Bash command.

In addition, we enhance the base prompt by adding eight prompting techniques to it. Some add knowledge, others guide problem-solving.

Guidance-Based:

- **Role Play**: Instructs the model to assume the persona of a Linux expert to ground its responses in this domain [24].
- **Few-shot 1, 3 and 5**: Provides static, in-context single-turn examples demonstrating how the model should generate its response [25]. We test this with one, three, and five manually curated examples.
- **Chain-of-Thought (CoT)**: Directs the model to generate a step-by-step reasoning process before providing the final command, steering it toward a more deliberate solution [26].

Knowledge Retrieval:

- **RAG with examples**: Matches the five-shot prompting technique by retrieving five task-relevant examples, as opposed to the static examples used in few-shot prompting. The examples are retrieved from the NL2Bash reasoning dataset.
- **RAG with manual pages**: Provides the model with the two manual pages retrieved from the man pages dataset.
- **RAG with manual pages and examples**: Combines the previously described retrieval methods, by adding three examples to the context.

Our three RAG techniques leverage the all-MiniLM-L6-v2 embedding model for retrieval. We truncate the retrieved documents to an estimated 4,000 tokens, to ensure the context window has ample room for reasoning and command outputs. In cases where truncation is applied, we prepend the prompt notifying the model that the prompt was truncated. By applying this truncation, we ensure that the task description remains in the context. We apply this method over the default vLLM truncation method, which retains the last tokens that fit in the available context. In our evaluation, that method would remove the task description.

D. Test Environment

Our evaluation is conducted using the InterCode-ALFA test set, containing 300 natural language tasks, paired with two solution commands. We selected this benchmark over alternatives like Terminal-Bench for two primary reasons. First, at the time of our selection, InterCode-ALFA offered a broader set of tasks. Second, Terminal-Bench is specifically designed to evaluate complex agents through difficult, multi-turn scenarios. The Terminal-Bench leaderboard [27] is dominated by state-of-the-art agents powered by proprietary frontier models. The benchmark creators evaluated their agent terminus 1 [28] with massive open-source models such as Qwen3-235B and DeepSeek-R1, and achieved a task success rate of 6.6% and 5.7%, respectively. Similarly, Codex CLI driven with GPT-4.1 [1] obtained an accuracy of 8.3%. In this context, small models without complex agentic capabilities would likely fail to solve enough tasks to provide meaningful insights. In contrast, the tasks in the InterCode benchmarks are generally solvable with a single command, making them a more appropriate target for evaluating the raw capabilities of local models.

We discuss the test environment in two parts: first, how the models interact with the test environment; and second, the modifications introduced to enable large-scale evaluation.

a) Model Interaction: Each model has three turns in a single session (attempt) to complete its task, allowing for iterative problem-solving. Each turn consists of a user prompt and the model's response. The first user prompt describes the task and instructs the model to complete it by providing commands within a Markdown code block. This prompt includes additional information depending on the prompting technique being evaluated. The commands generated by the model in a multi-line code block are then executed sequentially, line by line, within the Docker container.

All subsequent user prompts provide feedback to the model on the executed commands (i.e., the standard output and exit code). Similar to RAG documents, command outputs can be long, potentially exceeding the model's context window. To address this, we apply the same 4,000 tokens limit to the responses, truncating any that exceed this limit. After the third turn, the interaction terminates, and the environment is restored to its original state. This three-turn interaction represents an attempt and is repeated three times per task. Once the three attempts are completed for a given task, we proceed to the next task.

b) Execution Architecture: To facilitate our large-scale study, we make two key adjustments compared to the

InterCode-ALFA setup. First, we implemented a highly parallelized execution architecture centered around a vLLM [14] inference server running on an NVIDIA RTX 5090. We distribute the tasks for every prompting technique across a thread pool, with every thread interacting with an independent containerized environment. This design yields two critical benefits: it maximizes throughput by leveraging vLLM's efficient request batching, and it ensures experimental integrity by guaranteeing strict isolation between every concurrent test run.

Second, to enable independent container execution per thread, we employ a single Docker image rather than the four used in InterCode-ALFA. In their setup, each image is produced by a distinct initialization script that creates task-specific directories. There is no overlap in these directories, allowing them to be safely merged into the same image without impacting the task. The single container simplifies container management and removes the need for task-based container swaps.

E. Evaluation and Metrics

To robustly assess NL2Bash performance, we design an evaluation framework that builds on previous work [12], [11] but introduces several key improvements. Our evaluation extends the scope of system monitoring, improves fidelity of change detection, and adopts a task-aware scoring scheme that better reflects real task objectives. These changes result in more accurate, comprehensive and meaningful measurements of model performance.

Specifically, our methodology consists of three main components:

1) **Expanded File System Monitoring**: broader coverage of critical directories to detect relevant changes;
2) **Fine-Grained Change Detection**: a more expressive tracking mechanism that captures a wider range of system modifications; and
3) **Task-Aware Scoring**: a dynamic evaluation scheme aligned with task intent rather than fixed weighting.

These components together provide a more comprehensive assessment of system-level actions, allows for a more accurate success measurement, and mitigates blind spots that could previously lead to incorrect scoring.

a) Expanded File System Monitoring: Previous benchmarks restricted monitoring to a small subset of directories, which meant that changes to important locations (e.g., configuration files) were missed. The limited scope misses directories required to score certain tasks (e.g. changes to the user home folder). Moreover, a potentially malicious command such as `ls;rm -rf /etc` would receive a perfect score for correctly listing files, while it removes critical untracked files.

We address this issue by expanding the monitored scope to include all relevant system directories except those representing virtual file systems or temporary runtime data, including `/proc, /sys, /dev, /run, and /tmp`.

b) Fine-Grained Change Detection: We implement a new change-detection mechanism based on `rsync`, which captures a richer set of file system events than previous `Git` or `podman`-based methods. This includes tracking of permission changes, ownership and group modifications, creation and deletion of empty directories, and detection of symbolic links and their targets.

To assess the necessity of this approach, we analyzed the dataset of InterCode-ALFA and found 18 tasks whose score was invalid due to incomplete change detection. This validated our design choice and demonstrated that it has a measurable impact on benchmark outcomes.

We run `rsync` with the following flags to monitor file system changes: `archive, omit-dir-times, dry-run, itemize-changes`, and `compare-dest`. Our current implementation does not track hard links, access control list (ACL) permissions and, extended attributes. While this choice introduces a potential blind spot, tracking these types of modifications are not required to evaluate the tasks in the current test set. Furthermore, `rsync` represents hard links as new files (unless specifically instructed to preserve them). This results in our evaluation being able to distinguish between symbolic and hard links, but not between hard links and new files. We accept this trade-off for the scope of this work, as it avoids unnecessary overhead while still capturing all changes relevant to the evaluated tasks, since no task specifically requests hard links, ACL permissions or extended attributes. Importantly, `rsync` has the capability to monitor them, and our framework can therefore be extended to include them in future benchmarks or in scenarios where stricter guarantees are required.

c) Task-Aware Scoring: The core of our evaluation framework is a task-aware scoring system that directly reflects the intent of each task. Previous benchmarks such as InterCode and InterCode-ALFA applied a fixed scoring scheme (weighting terminal output at one-third and file system changes at two-thirds), regardless of the intended task result. This static weighting often led to misleading scores. For instance, all commands that make no file system changes for tasks expecting only terminal output receive a minimum score of 2/3, while tasks involving file system modifications would penalize commands that provide harmless terminal output with a maximum of 2/3.

To address these shortcomings, we classify tasks based on the expected outcomes of their two gold-standard solutions and tailor the scoring criteria accordingly:

1) Output Tasks: that generate terminal output, but do not modify the file system;
2) File system Tasks: that modify the file system without producing terminal output;
3) Both: Tasks that require both output and file system changes;

For each category, we apply metrics aligned with the task objective. As in prior work, *output tasks* are evaluated using cosine similarity between large model embeddings of the generated and gold output. Prior work leverages mxbai-large as embedding model. However, this model is limited to a context of 512 tokens and scores relatively low on the Massive Text Embedding Benchmark (MTEB) [29]. Therefore, we use

Qwen3-Embedding-8B due to its high score on that benchmark and its increased context window to 32,000 tokens. *File system tasks* are evaluated by calculating the Gaussian error function on the number of symmetrically different changes between the set of gold and model file system changes. This encompasses both the extra and missed changes made by the model. The Gaussian error function limits the score to a range between zero and one.

$$S_{\text{task}} = \begin{cases} S_o = \cos(\mathbf{o}_g, \mathbf{o}_m) & \text{if } T = \text{Output} \\ S_{fs} = 1 - erf(|C_m \Delta C_g|) & \text{if } T = \text{File system} \\ \frac{1}{2}\left[S_o + S_{fs}\right] & \text{if } T = \text{Both} \end{cases} \tag{1}$$

Here, T is the task type (output, file system, or both), C_g is the set of gold file system changes, and C_m is the set of the model-predicted changes. $\cos(\mathbf{o}_g, \mathbf{o}_m)$ denotes the cosine similarity between the gold output embedding $\mathbf{o}_g$ and the output embedding from model commands $\mathbf{o}_m$. Since models can submit multiple commands, the final score is derived from the command with the highest similarity score during the attempt.

In addition, we identify commands as unverifiable when (i) neither gold command yields a detectable result (no terminal output and no file system change) or (ii) the two commands conflict. We identified twelve tasks in the test set as unverifiable, these tasks were excluded from the evaluation, since these commands can not reliably be scored (e.g. announcing system maintenance using the `wall` command).

We evaluate each attempt with the task-aware scoring system. For an attempt to be considered successful, it must achieve the maximum score of 1. We use a similarity threshold to round S_o to 1 whenever this threshold is exceeded. Consequently, a file system task is considered successful when the file system matches the gold file system on the last turn. Finally, a model has solved a task whenever the majority of the attempts (at least two out of the three) are successful. This design aims to evaluate models on consistency, rather than counting a single lucky attempt.

F. Validation of Evaluation

Westenfelder et al. [11] validated functional equivalence heuristics by introducing a second ground truth. This allows them to assess evaluation strategies based on known correct and incorrect solutions. We leverage this methodology to assess our decision to change embedding models and determine the similarity threshold that maximizes the F1-score. Figure 1 presents the F1-score for similarity thresholds ranging from 0.7 to 1.0.

The result suggests, counter-intuitively, that lower threshold result in higher scoring accuracy. In addition, the figure displays that the Qwen3-embedding model does not provide the expected improvement. Upon investigation, we found that the simplicity of negative samples resulted in a low false positive rate. Additionally, the commands leveraged as incorrect samples are not representative of realistic failures the language models might make. We suspect that the threshold chosen in prior work (0.75) is therefore too low.

To address this issue, we manually extend the dataset with incorrect solutions for each task, creating commands that represent realistic mistakes. These commands are specifically designed to closely represent incorrect attempts that try to solve the task. For instance, removing all files in a directory instead of only text files or listing the amount of words from the wrong file.

Figure 2 displays the same F1-score comparison with the realistic negative samples. We notice a drop in F1-Score due to the increased difficulty of correctly determining a solution as incorrect. This results in the best performing threshold shifting to a higher value, for both embedding models. Our results indicate that mxbai-large performs best at a threshold of 0.92 and Qwen3 at a threshold of 0.849. We also find that evaluating with realistic failures increases the impact of a better embedding model. The Qwen3 embedding model shows a flatter top, indicating it is less susceptible to changes in threshold. In Our evaluation we used a threshold of 0.85.

In addition, we determine the impact of adding task-aware scoring on the F1-score. To ensure there is no information leakage, we determine the task type on the single gold command, since the other is used as positive sample. By changing our evaluation to task-aware scoring with a better embedding model and similarity threshold improves the F1-score from 0.735 to 0.798. This evaluation uses `rsync` to monitor file-system changes for both embedding models. We limit the validation to the embedding similarity heuristic presented in related work, since LLM as a judge significantly increases the amount of time and compute required, which would not be feasible with the scale of our evaluation (a total of 198,000 attempts). Furthermore, the prior study done by Westenfelder et al. demonstrated that only a frontier cloud model as a judge improved over embedding.

IV. Evaluation

We evaluate the models based on their percentage of solved tasks, with the results summarized in Table IV. In this section, we compare the model and prompting technique results based on model characteristics (e.g. size, quantization, family, and specialization). In addition, we gain insights into the failure reasons by manually investigating the generated solutions.

A. Model Performance Analysis

To analyze the impact of model performance, we compare the model base prompt results. Figure 3 displays the base results from the models, grouped by size and colored per family. We analyze these results and investigate the impact of the following model characteristics:

1) **Model Weights and Quantization**: The results show a general trend indicating model performance is highly correlated with size. However, the scores of the largest model stagnate and even decreases, likely due to the quantization;

TABLE IV
TASK COMPLETION RATES ACROSS MODELS AND PROMPTING TECHNIQUES. **BOLD** VALUES REPRESENT THE BEST SCORING TECHNIQUE PER MODEL. UNDERLINED VALUES DISPLAY THE BEST SCORING VALUE PER TECHNIQUE. ADDITIONAL RESULTS TO ASSESS NEGATIVE RESULTS ARE SHOWN IN GRAY.

					Few shot			RAG			
Model Name	STP	EBP	Role Play	CoT	1	3	5	Man-pages	Examples	Both	Oracle
DeepSeek-R1-Distill-Qwen-1.5B	0.07	0.08	0.08	0.08	0.08	0.07	**0.12**	0.11	0.06	0.12	0.16
Qwen2.5-1.5B-Instruct	0.44	0.50	**0.51**	0.33	0.49	0.45	0.49	0.32	0.08	0.16	0.43
Llama-3.2-1B-Instruct	0.21	0.30	**0.31**	0.15	0.14	0.16	0.24	0.20	0.06	0.10	0.25
Qwen2.5-3B-Instruct	0.59	**0.60**	0.58	0.52	0.57	0.57	0.54	0.51	0.23	0.35	0.60
Llama-3.2-3B-Instruct	0.52	0.53	**0.56**	0.28	0.53	0.50	0.55	0.37	0.13	0.27	0.55
Qwen3-4B	**0.73**	0.70	**0.73**	**0.73**	**0.73**	0.71	0.72	0.69	0.40	0.58	0.72
gemma-3-4b-it	0.50	0.58	0.56	**0.59**	0.57	0.56	0.56	0.49	0.15	0.24	0.57
Llama-3.1-8B-Instruct	0.71	0.70	**0.73**	0.64	0.72	0.67	0.70	0.66	0.40	0.54	0.70
DeepSeek-R1-Distill-Llama-8B	0.43	**0.47**	0.43	0.37	0.45	0.41	0.42	0.43	0.23	0.31	0.52
Qwen2.5-7B-Instruct	0.64	0.63	0.65	0.59	0.67	**0.69**	0.68	0.67	0.31	0.51	0.74
Qwen2.5-Coder-7B-Instruct	0.70	0.70	0.72	0.66	**0.76**	0.71	0.72	0.63	0.38	0.54	0.68
Qwen3-8B	0.75	0.75	0.78	0.77	**0.79**	**0.79**	0.77	0.74	0.65	0.74	0.78
DeepSeek-R1-Distill-Qwen-7B	0.26	0.28	0.28	0.28	**0.31**	0.22	0.28	0.30	0.14	0.15	0.39
gemma-3-12b-it-FP8-dynamic	0.67	0.70	**0.72**	0.69	0.70	**0.72**	0.70	0.61	0.23	0.36	0.71
Qwen3-14B-FP8-dynamic	0.80	**0.82**	0.77	0.77	0.77	0.77	0.78	0.76	0.68	0.76	0.81
Qwen2.5-14B-Instruct-FP8-dynamic	0.74	0.71	0.74	0.71	0.72	0.72	**0.77**	0.73	0.61	0.70	0.74
DeepSeek-R1-Distill-Qwen-14B-FP8-dynamic	0.75	0.74	**0.76**	0.74	0.71	0.73	0.73	0.73	0.59	0.71	0.75
phi-4-FP8-dynamic	0.74	0.75	0.72	0.72	0.74	0.74	**0.78**	0.77	0.73	0.77	0.76
Qwen3-32B-quantized.w4a16	**0.78**	0.75	**0.78**	0.75	0.77	0.75	**0.78**	0.71	0.73	0.73	0.79
Qwen2.5-32B-quantized.w4a16	0.63	0.68	0.68	0.62	0.62	**0.69**	0.65	0.50	0.38	0.44	0.50
DeepSeek-R1-Distill-Qwen-32B-quantized.w4a16	0.74	0.77	0.78	0.73	0.78	0.75	**0.79**	0.73	0.70	0.66	0.80
gemma-3-27b-it-quantized.w4a16	0.70	0.71	**0.73**	0.69	0.70	0.70	0.70	0.64	0.36	0.46	0.72
Average	0.6	0.61	**0.62**	0.56	0.61	0.59	0.61	0.56	0.37	0.46	0.62

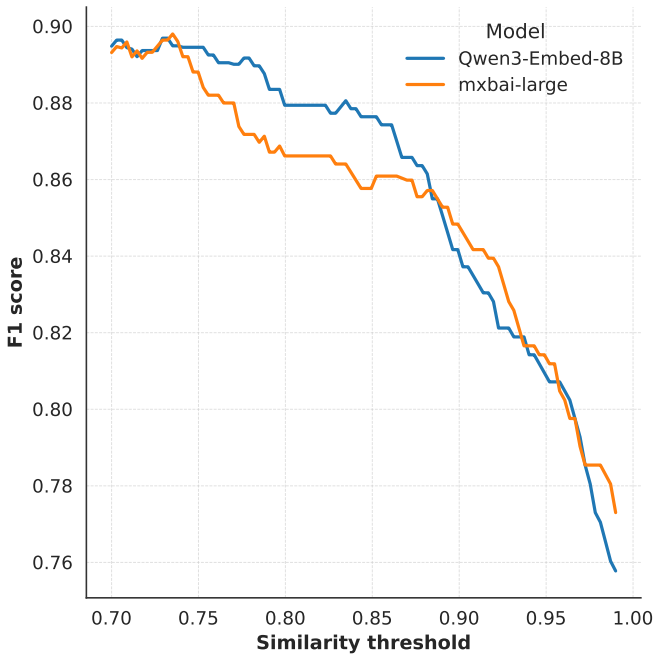

Fig. 1. F1-score for thresholds ranging from 0.7 to 1, using the easy incorrect commands as InterCode-ALFA, for Qwen3-embed-8B and mxbai-large.

2) **Model Specialization**: We notice the small and medium-sized DeepSeek models drastically underperform compared to the original model, whereas the larger models improve over it. Similarly, the model specialized in code-generation, showed improved performance over its non-specialized counterpart. As only a single coding model was evaluated, further research is needed to draw a general conclusion.
3) **Model Family**: The Qwen model families consistently performs well across all size categories. Notably, the small Qwen3 model outperforms larger models from other families. Furthermore, Qwen3 shows a clear performance improvement over its predecessor.

B. Prompt Impact Analysis

We discuss the impact of different prompting techniques and which model sizes benefit the most from applying the techniques. Figure 4 displays the performance gained by adding prompting techniques to the base STP prompt. Additionally, we manually investigate the generated solutions to gain insight into model failures.

a) EBP outperforms STP: Introducing interactivity in the prompting process (EBP) has a slight positive impact on successful solves; 14 out of the 22 models increased performance over the simple translation prompting baseline (STP). This improvement is likely due to the fact that the STP baseline expects a correct command to be produced immediately, leaving no room for intermediate reasoning or refinement. In contrast, EBP does not impose the direct translation and specifies the available turns, allowing models to plan their solution. Reasoning-oriented models such as Qwen3 and DeepSeek had fewer issues with the STP prompt, likely because they disregarded the instruction to produce a direct translation, and instead generated intermediate reasoning steps first. It is worth noting that STP requests the models

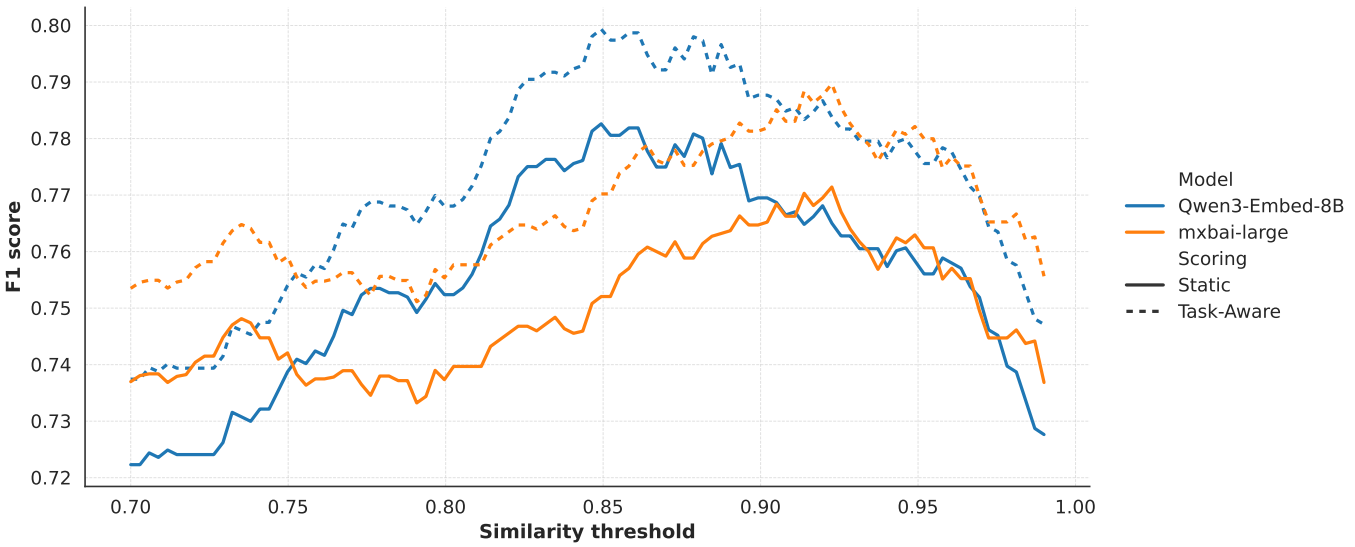

Fig. 2. Comparison of task-aware and static scoring for Qwen3-embed-8B and mxbai-large, across similarity thresholds ranging from 0.7 to 1 using difficult incorrect commands.

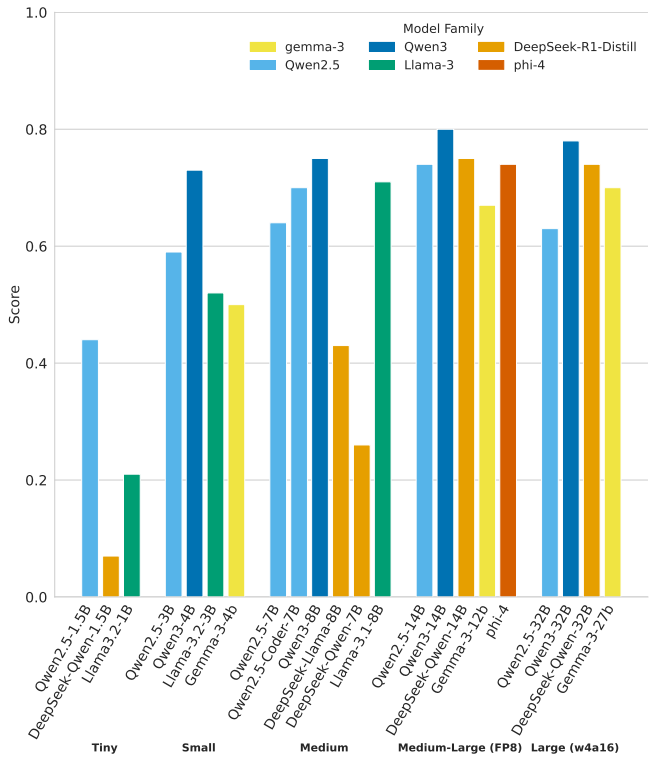

Fig. 3. Model STP scores grouped by size and colored per model family.

to directly provide a Bash command; however, this is not enforced, models can ignore it. Surprisingly, Qwen3-4b is one of the few models that noticeably performs better with the baseline prompt. We manually investigated the reasoning generated by Qwen3-4b for both STP and EBP, and found that the model had difficulties with the request to write commands instead of multi-line scripts. When using the EBP prompt, the model stated multiple times that scripting is required, whereas the same task solved with STP did not have this issue.

b) Positive Impact of Role-Play: This technique aims to improve performance by explicitly instructing the model to adopt the role of a Linux expert. As shown in Figure 4, the technique enhances task-solving ability across most models. Role-playing was particularly beneficial for tiny and small models. The impact is most pronounced for the Llama family, which consistently obtains the highest score under this prompting technique. Interestingly, the DeepSeek distilled variant based on the Llama model does not improve in performance using Role-Play.

c) Chain-of-Thought: Overall, we observe a negative performance impact when models are asked to provide a chain-of-thought. With the Llama model family scoring significantly worse compared to the baseline. Model size did not improve performance relative to the base prompt; 16 out of the 22 models showed a decreased performance. An increase in model size did reduce the performance decline. An exception to this is Gemma3-4b, which improved by 9% over the base prompt by generating a chain-of-thought.

We investigated the reasoning traces produced by the Gemma and Llama models to understand the sources of their performance differences. We found that Gemma first analyzed the tasks, then listed possible solution strategies, and finally provided its answer, leading to well-grounded solutions. On the other hand, Llama-3 models often generated Bash scripts with comments serving as reasoning, but this did not help the model solve the problem. Another factor contributing to the decreased performance is a conflict between the baseline prompt, which requests a direct answer, and the chain-of-thought setup, which requests reasoning before answering.

d) Few-Shot: Figure 4 shows that adding examples has a mixed effect on model performance: depending on the number of examples, performance either increases or decreases. With the medium models performing better with a single example and the larger (medium-large and large) models improving as the examples increased. We also observe a consistent pattern in which three-shot underperforms compared to one-shot and five-shot variants. This suggests that simply adding more examples does not necessarily improve a model's ability to solve tasks.

e) Issues with RAG: We leverage two data stores to provide additional context to the model: an examples store, containing task-relevant examples including reasoning traces, and a manuals store, which provides documentation on available commands. The examples are designed to illustrate step-by-step reasoning and guide the model toward correct task completion. Despite this, our results show a significant decline in performance when either of these resources is used. To better understand this outcome, we manually inspected model behavior for selected tasks from Llama3.2-3B, Gemma3-4B, Qwen2.5-coder-7B and Gemma3-12B.

Our analysis revealed two major failure modes when examples were provided. First, models frequently failed when the retrieved examples exceeded the token limit and were truncated. In such cases, the model often attempted to complete the truncated example instead of solving the given task. Second, task-irrelevant examples led models to use incorrect tools (e.g., `ln` instead of `cp`). Additionally, 55% of the retrieved examples were truncated, which is likely the main reason for the performance decline relative to the baseline. However, even when examples were not truncated, models were incapable of creating task-relevant reasoning. We also notice that the performance decline was less severe with reasoning models, especially the large Qwen3 and DeepSeek models.

The manual pages retrieval was more effective but still produced mixed results. Analysis showed that the model received a relevant manual page in only 23% of the cases. Comparing model performance across successful and unsuccessful retrieval scenarios, we found that performance decreased by 6.4% when irrelevant manuals were provided and improved by 3.6% with relevant manuals. Restricting the analysis to large models, revealed that all, except Qwen3, improved when provided with useful documentation.

To isolate the impact of retrieval quality, we evaluate an additional oracle RAG strategy. This strategy retrieves manual pages by leveraging the gold solution, thereby ensuring the retrieved documents are relevant to the task. This allows us to approximate the impact of an ideal retrieval system and provides an upper bound on the gains improved embedding models can achieve. As shown in Figure 4, providing a model with relevant pages resulting in a general improvement over the baseline and other retrieval method. Even though this strategy provides an upper approximation of retrieval, it does not consistently out-perform other techniques. Improving the truncation strategy is required to further improve the results for this strategy.

Overall, the issues with the RAG prompting techniques, resulted from the simplicity of our implementation. Primarily driven by truncation and weak retrieval. Even when correct examples or documentation were retrieved, models generally underperformed relative to the baseline, except for large models benefitting from accurate manual retrieval. Due to the limitations stemming from implementation, we cannot draw definitive conclusions about the potential performance gains of more sophisticated RAG pipelines.

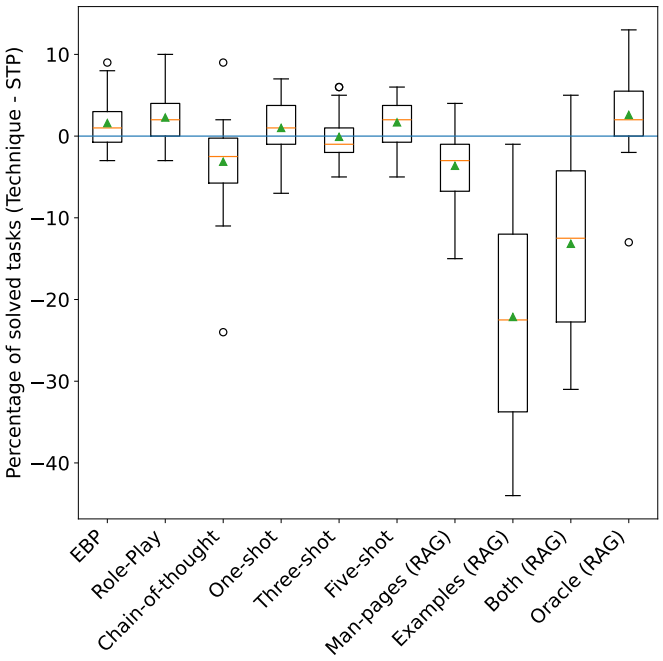

Fig. 4. Percentage difference between prompt techniques and base (STP) across the different models.

V. Discussion

In this section, we discuss the limitations of our monitoring system, the challenges inherent in task scoring, and the broader implications of our results. We then conclude by outlining practical recommendations for deploying open-source language models.

a) Monitoring system limitations: Our setup does not yet exploit the full potential of `rsync`. Specifically, it does not monitor hard links, access control permissions, or extended attributes, which means that certain changes may go undetected.

Future work could expand on this monitoring system by integrating additional tools capable of tracking environmental variables, running processes, and network activity. This would broaden the range of tasks that can be evaluated, and provide a more complete picture of how safely a model behaves in a sandboxed environment.

b) Difficulties with scoring systems: Designing a scoring system which reliably captures task completion remains challenging. While our scoring system successfully filters out task-irrelevant changes and improves over static-weight methods,

it still remains an approximation of task success. Embedding-based similarity is imperfect: it cannot always distinguish between semantically correct and incorrect outputs, particularly in cases where correctness is subjective, underspecified, or depends on subtle contextual cues beyond lexical similarity. Likewise, the impact of unexpected file-system changes is not penalized. In our implementation, adding files results in the same penalty as removing files.

Future work can further improve our score by implementing a safety metric which penalizes modifications to critical folders such as `/etc` and other high-risk actions. However, this is challenging, since it is difficult to design a metric that captures all dangerous behaviors (e.g., starting malicious processes, installing malware, or extracting sensitive data). Additionally, using a language model as a judge could improve evaluation accuracy for tasks where automatic metrics fail to capture semantic correctness, although this may require substantially more computational resources.

c) Results and recommendations: Our results indicate that model selection has the most impact in achieving high performance, with the Qwen3 models outperforming all others in our evaluation. Applying the correct prompting techniques also improved performance: techniques such as Role-Play and EBP consistently improved results. However, the impact of prompting varies by model size and family, as summarized in Table V.

Our benchmark focused on relatively simple Bash tasks, but the insights extend to more complex applications. Future research can leverage these findings to inform the design of agents that use open-source models for automated system interaction. Based on our observations, we provide the following recommendations:

1) **Prioritize model selection**: Our evaluation indicates that model selection has the largest influence on performance.
2) **Bigger is not always better**: Larger models do not necessarily outperform smaller ones. For example, Qwen3-4B outperformed several larger models. Additionally, reducing weight quality (i.e., quantization) to run larger models does not necessarily result in greater performance.
3) **Prompting can help or hurt**: Prompting strategies can significantly change outcomes. Carefully review how the prompt design affects model behavior.
4) **Implement RAG judiciously**: While retrieval-augmented generation can enhance performance by supplying additional context, its effectiveness depends on careful implementation. Long documents that require truncation can degrade performance. Additionally, irrelevant documents can further decrease accuracy. Therefore, documents should be retrieved when necessary and carefully filtered for relevance.

Selecting the best open-source model is challenging, especially under hardware constraints, and running a large-scale evaluation over many models and prompting techniques may not be feasible for practitioners. In practice, we recommend a staged evaluation strategy. First, run a simple baseline prompt (e.g., STP or EBP) on a small test set across a large set of candidate models, discard poorly performing models or retain only best-performing models. Then, on this reduced model set, assess the impact of individual prompting techniques and RAG configurations on a larger test set, while manually inspecting sampled model outputs to identify prompting or retrieval failures. This approach concentrates effort on the most promising models and configurations, while respecting realistic hardware and time constraints.

TABLE V
PROMPTING IMPACT SUMMARY

Model Size	Impact Prompting	Best Technique
Tiny	High	Role-Play
Small	Medium/High	Role-Play
Medium	Medium	One-Shot
Medium-large	Low	Five-Shot
Large	Medium	EBP and Role-Play

VI. CONCLUSION

In this paper, we presented a large-scale empirical study evaluating the capabilities of 22 large language models in completing tasks in a Linux environment. Our results provide insights into why models fail, how their performance can be improved, and which techniques are most effective for locally deploying open-source language models.

We addressed several limitations of previous work, introducing an enhanced monitoring approach based on `rsync`, enabling a broader coverage of file-system changes. Additionally, we proposed a task-aware scoring system that dynamically adapts evaluation metrics to better reflect the intended outcomes of tasks.

As discussed in Section V, designing a general evaluation framework that reliably determines correctness remains a significant challenge. Future work can build on our improved scoring system by further increasing the monitored environmental changes and designing a safety metric. Additionally, expanding the benchmark to include currently unscored tasks, such as those involving process management, will further advance the evaluation of language-model-driven system automation.

REFERENCES

[1] OpenAI, J. Achiam *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[2] G. Comanici, E. Bieber *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," 2025. [Online]. Available: https://arxiv.org/abs/2507.06261

[3] A. Happe, A. Kaplan, and J. Cito, "Llms as hackers: Autonomous linux privilege escalation attacks," 2025. [Online]. Available: https://arxiv.org/abs/2310.11409

[4] J. Huang and Q. Zhu, "Penheal: A two-stage llm framework for automated pentesting and optimal remediation," ser. AutonomousCyber '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 11–22. [Online]. Available: https://doi.org/10.1145/3689933.3690831

[5] Y. Zou, J. Liu, and W. Fan, "Ctfagent: An llm-powered agent for ctf challenge solving," *Journal of Information Security and Applications*, vol. 96, p. 104305, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212625003424
[6] M. Shao, B. Chen *et al.*, "An empirical evaluation of llms for solving offensive security challenges," 2024. [Online]. Available: https://arxiv.org/abs/2402.11814
[7] T. H. F. Team, "Hugging face," 2016. [Online]. Available: https://huggingface.co/
[8] Q. Fu, Z. Teng *et al.*, "Nl2cmd: An updated workflow for natural language to bash commands translation," *Journal of Machine Learning Theory, Applications and Practice*, pp. 45–82, 2023.
[9] C. Cao, F. Wang *et al.*, "Managing linux servers with llm-based ai agents: An empirical evaluation with gpt4," *Machine Learning with Applications*, vol. 17, p. 100570, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266682702400046X
[10] N. P. A. Vo, B. Paulovicks, and V. Sheinin, "Execution-based evaluation of natural language to bash and powershell for incident remediation," 2024. [Online]. Available: https://arxiv.org/abs/2405.06807
[11] F. Westenfelder, E. Hemberg *et al.*, "Llm-supported natural language to bash translation," 2025. [Online]. Available: https://arxiv.org/abs/2502.06858
[12] J. Yang, A. Prabhakar *et al.*, "Intercode: Standardizing and benchmarking interactive coding with execution feedback," 2023. [Online]. Available: https://arxiv.org/abs/2306.14898
[13] P. Team, "Podman documentation," 2019. [Online]. Available: https://docs.podman.io/en/stable/index.html
[14] W. Kwon, Z. Li *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
[15] G. Team, A. Kamath *et al.*, "Gemma 3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2503.19786
[16] Qwen, : *et al.*, "Qwen2.5 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2412.15115
[17] B. Hui, J. Yang *et al.*, "Qwen2.5-coder technical report," 2024. [Online]. Available: https://arxiv.org/abs/2409.12186
[18] A. Yang, A. Li *et al.*, "Qwen3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388
[19] A. Grattafiori, A. Dubey *et al.*, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783
[20] DeepSeek-AI, D. Guo *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2501.12948
[21] M. Abdin, J. Aneja *et al.*, "Phi-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2412.08905
[22] X. V. Lin, C. Wang *et al.*, "Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation LREC 2018, Miyazaki (Japan), 7-12 May, 2018.*, 2018.
[23] T. sentence-transformers Team, "sentence-transformers/all-minilm-l6-v2," 2021. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[24] H. Tamoyan, H. Schuff, and I. Gurevych, "Llm roleplay: Simulating human-chatbot interaction," 2024. [Online]. Available: https://arxiv.org/abs/2407.03974
[25] T. B. Brown, B. Mann *et al.*, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165
[26] J. Wei, X. Wang *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903
[27] T. T.-B. Team, "Terminal-bench: A benchmark for ai agents in terminal environments," Apr 2025. [Online]. Available: https://www.tbench.ai/leaderboard
[28] ——, "Terminal-bench: A benchmark for ai agents in terminal environments," Apr 2025. [Online]. Available: https://www.tbench.ai/terminus
[29] N. Muennighoff, N. Tazi *et al.*, "MTEB: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 2014–2037. [Online]. Available: https://aclanthology.org/2023.eacl-main.148/