Evaluation Overview

This report summarizes the results of benchmarking three language models: Claude 4.5 Haiku, DeepSeek V4 Pro, and Kimi K2.6. Each model was evaluated using the Terminus 2 agent on a subset of the Terminal Bench 2.1 benchmark (terminal-bench benchmark, terminal-bench-2.1 dataset).

Because of the allocated evaluation budget, only 40 tasks were included in this evaluation. These tasks cover a wide range of problem types, including software engineering, security, scientific computing, and data processing. All three models were evaluated using the same benchmark configuration to ensure a fair comparison. The complete list of evaluated tasks can be found in the Appendix.

The evaluation included both quantitative and qualitative analysis. Quantitative metrics included task completion rate, execution runtime, and total API cost. Qualitative analysis focused on agent trajectories, shell command usage, problem-solving strategies, and common failure modes to better understand how the models behaved throughout the benchmark.

Results

DeepSeek V4 Pro achieved the highest overall score, successfully completing 14 of the 40 benchmark tasks (35.0%). Kimi K2.6 finished close behind with 13 completed tasks (32.5%), while Claude 4.5 Haiku completed only 5 tasks (12.5%). DeepSeek also had the highest API cost at $10.14 and the longest average runtime of 938.5 seconds. Kimi had the lowest API cost ($7.52) while maintaining a similar pass rate. Claude completed tasks much faster, averaging 382.0 seconds per task, but this came with much lower overall performance. Table 1 summarizes the overall benchmark results for the three evaluated models.

Model	Tasks Passed	Tasks Failed	Pass Rate	API Cost	Avg. Agent Runtime (s)
DeepSeek V4 Pro	14	26	35.0 %	$10.14	938.5
Kimi K2.6	13	27	32.5 %	$7.52	685.2
Claude 4.5 Haiku	5	35	12.5 %	$7.73	382.0

Table 1: Benchmark Results for All Models

Table 2 breaks down where the runtime was spent during each evaluation. DeepSeek spent the most time running inside the sandbox, averaging 981.0 seconds, followed by Kimi at 733.7 seconds and Claude at 470.3 seconds. Claude had the longest evaluation time outside of the sandbox, while DeepSeek had the shortest. Sandbox build times were nearly identical for all three models, showing that the differences in total runtime were mainly caused by the models themselves rather than the evaluation environment.

Model	Avg. Agent Runtime (s)	Sandbox Runtime (s)	Evaluation Runtime (s)	Sandbox Build Time (s)

DeepSeek V4 Pro	938.5	981.0	26.4	2.4
Kimi K2.6	685.2	733.7	31.4	3.6
Claude 4.5 Haiku	382.0	470.3	69.6	3.7

Table 2: Runtime Statistics of Terminal Benchmark

Overall, DeepSeek V4 Pro achieved the strongest benchmark results, with the highest pass rate and the widest range of successfully completed tasks. However, as mentioned previously, this came with the highest API cost and the longest runtime. Kimi K2.6 achieved a very similar pass rate while having the lowest API cost and a shorter average runtime, making it the best balance between performance, cost, and speed. Across most task categories, DeepSeek and Kimi performed similarly, with DeepSeek showing more strength on system administration tasks and Kimi successfully completing all mathematics tasks included in the benchmark.

Claude 4.5 Haiku consistently performed worse than the other two models. It achieved the lowest pass rate, completed fewer task categories, and produced the highest number of incorrect final solutions. Although Claude successfully completed a small number of low-level debugging and reverse engineering tasks, it struggled with the broader software engineering and multi-step tasks that DeepSeek and Kimi handled more successfully. The differences in how each model approached these tasks are examined in the following section.

Discussion

While the overall benchmark scores show which model performed best, they do not explain why those differences occurred. To better understand each model's strengths and weaknesses, the results were analyzed by task category and by examining the agent trajectories. The category-level results provide a better understanding of the types of problems each model was able to solve. As shown in Figure 1, DeepSeek V4 Pro and Kimi K2.6 successfully completed tasks across a wider range of categories than Claude 4.5 Haiku, particularly on debugging, software engineering, and system administration tasks. While some categories contained only one or two tasks, making those results less conclusive, the larger categories such as software engineering provide stronger evidence that the stronger models possessed broader engineering capabilities. Looking through the trajectories, these models successfully handled algorithm implementation, distributed systems, Linux service configuration, security research, and reverse engineering tasks that Claude frequently failed. In contrast, Claude's successful tasks were concentrated on a much smaller set of low-level debugging and reverse engineering problems.

The terminal trajectories also revealed noticeable differences in how each model approached these problems. Table 3 shows that all three models relied heavily on common Bash commands such as cat and python3, but their command usage diverged afterwards. DeepSeek more frequently used inspection commands such as ls, find, grep, which, and curl, suggesting it spent more time understanding the environment before making changes. Kimi relied more heavily on Python helper scripts and text-processing utilities, while Claude more often used commands such as sed, g++, and rustc, indicating it preferred directly editing and rebuilding code.

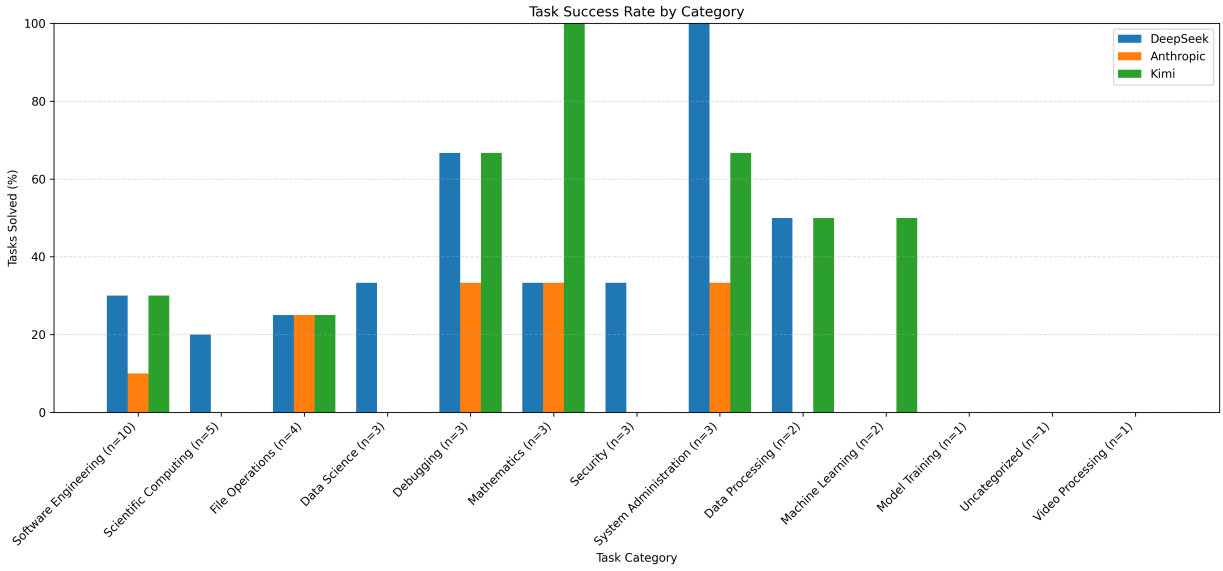

Figure 1: Model Results Split by Task Category

The trajectory analysis revealed clear differences in the models' overall reasoning strategies. DeepSeek V4 Pro generally took the most methodical approach, spending more time understanding the problem, validating assumptions, and carefully planning its next actions before making changes. Kimi K2.6 relied on a more exploratory strategy, frequently generating hypotheses, testing them incrementally, and backtracking when an approach proved unsuccessful. Claude 4.5 Haiku, in contrast, tended to commit to an initial idea much earlier, performing less validation before moving toward a final solution.

These differences are clearly illustrated by the feal-differential-cryptanalysis task (Appendix B). Both DeepSeek and Kimi repeatedly tested and refined their differential attack before validating the recovered key, while Claude committed to an incomplete implementation with limited testing. As a result, DeepSeek and Kimi successfully recovered the correct solution, whereas Claude submitted an incorrect final answer. Similar patterns were observed across many of the benchmark trajectories.

These reasoning strategies also help explain the observed failure modes. As shown in Table 4, DeepSeek most often failed by reaching the timeout limit, suggesting it continued debugging and validating its approach until the evaluation budget was exhausted. Claude most frequently failed by producing an incorrect final solution, consistent with its tendency to commit to an early hypothesis without sufficient validation. Kimi generally fell between these two extremes, often requiring several iterations to reach a solution but recovering from incorrect assumptions more effectively than Claude.

Rank	Claude	DeepSeek	Kimi
1	cat 334	cat 266	cat 376
2	python3 283	python3 256	python3 369
3	ls 121	ls 165	grep 125
4	sed 74	find 84	ls 84
5	echo 54	cd 84	curl 83
6	find 44	curl 74	sed 82
7	grep 43	grep 68	python 49
8	cd 39	which 55	find 44
9	g++ 26	sed 39	cd 28
10	rustc 25	echo 37	echo 27

Table 3: Command Usage Ranking for Each Model

Failure Mode	Anthropic	DeepSeek	Kimi
Wrong solution	18	2	10
Timeout	2	15	9
Step limit reached	11	5	7
Unexplained termination	3	0	1
Error	0	4	0
Give up	1	0	0

Table 4: Failure Mode Types and Counts for Each Model

Finally, several benchmark tasks were affected by Daytona's free-tier infrastructure, which prevented access to required external resources. Because all three models failed these tasks for the same reason, they are poor indicators of relative model capability. Instead, they provide additional insight into how each model behaved under difficult conditions. Even when a task was unsolvable due to infrastructure limitations, DeepSeek and Kimi generally continued exploring possible solutions for longer, whereas Claude more often terminated earlier with a final, but incorrect, solution. Overall, the qualitative analysis suggests that the performance differences between the models were driven not only by reasoning ability, but also by their underlying problem-solving strategies. Claude generally prioritized reaching a solution quickly, DeepSeek emphasized careful analysis and validation, and Kimi relied on iterative experimentation and refinement before converging on a final answer.

Errors

Two types of errors were encountered during the benchmarking process. The first type was infrastructure errors, which were caused by the limitations of the free-tier Daytona hosting environment. Specifically, the combined disk and memory limit of 10 GB across all sandboxes was occasionally exceeded when running three tasks concurrently, causing some sandbox creation or startup failures. These errors were retriable, so the affected tasks were rerun with a

lower concurrency until they completed successfully. As a result, each model has two benchmark result files, and these infrastructure failures were not included in the final evaluation results.

The second type was model errors**,** which were not retriable and occurred only during the DeepSeek evaluation. In four tasks, the model API returned an "Invalid assistant message: content or tool_calls must be set" error, causing the agent to terminate before completing the task. These failures were counted as model errors because they originated from the model response rather than the benchmarking infrastructure.

Tools Used

Two AI tools were used to assist with the benchmark analysis. ChatGPT was used to generate utility scripts for tasks such as converting JSON files to human-readable CSVs, calculating the total API cost, counting tasks that reached the step limit, analyzing and sorting Bash command usage, assembling task categories, and generating graphs. Docent was used for qualitative trajectory analysis, specifically to generate summaries of agent trajectories that were later reviewed and used during the analysis.

Appendix

Appendix A – Evaluated Tasks from Terminal Bench 2.1 Benchmark

overfull-hbox, adaptive-rejection-sampler, break-filter-js-from-html, build-pov-ray, caffe-cifar-10, db-wal-recovery, dna-insert, filter-js-from-html, financial-document-processor, gcode-to-text, largest-eigenval, mteb-retrieve, polyglot-c-py, raman-fitting, regex-log, rstan-to-pystan, schemelike-metacircular-eval, cancel-async-tasks, dna-assembly, extract-moves-from-video, feal-differential-cryptanalysis, gpt2-codegolf, llm-inference-batching-scheduler, make-doom-for-mips, model-extraction-relu-logits, polyglot-rust-c, protein-assembly, torch-pipeline-parallelism, video-processing, write-compressor, torch-tensor-parallelism, mailman, mteb-leaderboard, count-dataset-tokens, install-windows-3.11, custom-memory-heap-crash, extract-elf, password-recovery, sqlite-db-truncate, configure-git-webserver

Appendix B – Benchmark Trajectories

DeepSeek V4 Pro:
https://drive.google.com/drive/folders/1YfdFVrZwZTIA777_6H-JFhx5CaPLlqYk?usp=share_link

Claude 4.5 Haiku:
https://drive.google.com/drive/folders/1xTLkkGzfnwSlyanfwvLKld3PuEz0jmQu?usp=share_link

Kimi K2.6:
https://drive.google.com/drive/folders/1qjeAFVfoBiACiVBcd-2CveWHaR9QrL6l?usp=share_link