Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Data Science Lab Fall 25

ETH AI CENTER

Evaluation of a Swiss LLM-based Coding Assistant

Sinuo Liu [1], Anna Kravchenko [1], Imanol Schlag[2], Robert Matthew Smith[3]
[1]ETH Zurich, D-INFK [2]Apertus [3]ETH AI Center, ETH Zurich

1. Problem Statement

We study whether Swiss-hosted open LLMs can act as reliable coding assistants when accessed via terminal-based tools (interactive CLIs and scripted benchmarks).

Our main question is **how the Apertus models compare to other open models (e.g., Qwen, DeepSeek, Kimi, Mistral) on practical coding tasks** in terms of quality, stability, latency, and reproducibility.

We also examine the impact of infrastructure limits (SwissAI endpoints on Clariden/Alps cluster and OpenRouter).

We focus on building an end-to-end evaluation pipeline that runs locally, talks to SwissAI/CSCS endpoints, and is designed to scale to the Clariden/Alps infrastructure for larger experiments.

2. Related work

- **Apertus.** Open Swiss LLM family with released weights, code, and evals for multilingual, compliance-sensitive use [1], spun up on Clariden/Alps cluster [2].
- **Agent benchmarks.** *Terminal-Bench* tests terminal agents [3]; *SWE-bench* and *SWE-agent* benchmark repo-level coding agents on real GitHub issues [4;5].
- **CLI tooling.** *OpenCode* is a terminal agent UI [6]; OpenAI-style gateways (*SwissAI/CSCS* [7], *PublicAI* [8], *OpenRouter*[9]) and *LLM-Benchmark-CLI* provide simple CLI access to many models [10].

3. Data

- **Models.** SwissAI/CSCS-hosted Apertus-8B/70B and baselines Qwen3-80B [11], Mistral-7B [12], DeepSeek-V3.1 [13], Kimi-K2 [14], all served via OpenAI-compatible SwissAI endpoints on Clariden/Alps.
- **Benchmark suite (quantitative).** `terminal-bench-core==0.1.1`: 80 containerized terminal tasks with pass/fail labels approximating realistic developer actions.
- **Interactive prompt set (qualitative).** Small hand-written set of CLI coding prompts run via OpenCode to probe usability and latency in an interactive terminal workflow.

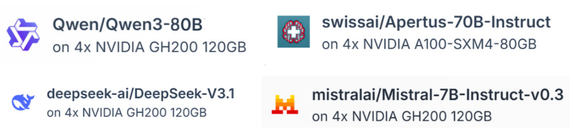

Figure 1: A few of used models

4. Methods

- **Infrastructure.** Set up a local and Clariden/Alps environment with Docker Desktop to run terminal-bench containers and expose OpenAI-compatible SwissAI endpoints.
- **Model deployment.** We used SwissAI/CSCS to spin up Apertus-8B/70B, Qwen3-80B, DeepSeek-V3.1, Mistral-7B, and Kimi-K2 models on Clariden/Alps, exposed via the SwissAI API.
- **CLI latency & usability.** Built a client that sends coding prompts, measures wall-clock latency, and logs per-model summaries.
- **Task-level benchmarking.** Ran Terminal-Bench (core v0.1.1) on SwissAI models over 80 tasks, recording resolved vs. unresolved trials and accuracy, and inspecting container logs for typical failure modes.

See Figure 2 for an overview of the pipeline.

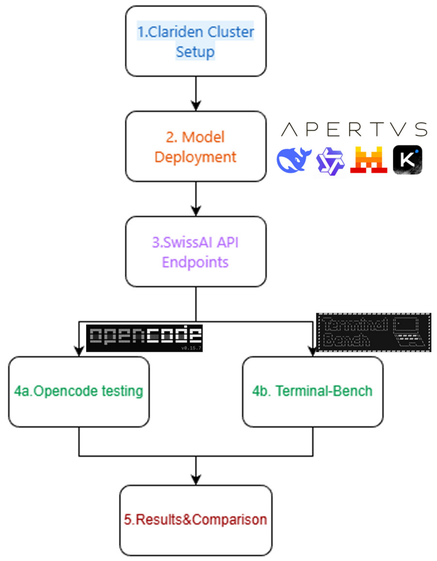

Figure 2: Working flow

5. Results

Table 1 shows a non-interactive latency benchmark on a fixed Fibonacci prompt. Apertus-8B is the fastest model (0.81 s on average), but typically produces very short answers; Qwen3-32B is slower (3.74 s) and returns more verbose, fully documented solutions, while Apertus-70B is the slowest (6.85 s). Even on this toy task, we already see a trade-off between latency and answer length/verbosity that we will explore further in our interactive CLI experiments.

Model	Runs	Mean latency [s]
Qwen3-32B	50	3.74
Apertus-70B	50	6.85
Apertus-8B	50	0.81

Table 1: Latency benchmark (non-interactive Fibonacci task, 50 runs per model)

Table 2 shows the Terminal-Bench evaluation results across six models on 80 terminal tasks. DeepSeek-V3.1 (685B parameters) achieved the highest accuracy at 27.50%, followed by Kimi-K2-Instruct (1T parameters) at 23.75%. Qwen3-80B reached 13.75%, while the Swiss-hosted Apertus-70B and Apertus-8B showed 3.75% and 2.50% respectively. Mistral-7B achieved the lowest at 1.25%.

Model	Resolved Trials	Unresolved Trials	Accuracy
DeepSeek-V3.1	22	58	27.50%
Kimi-K2-Instruct	19	61	23.75%
Qwen3-80B	11	69	13.75%
Apertus-70B	3	77	3.75%
Apertus-8B	2	78	2.50%
Mistral-7B	1	79	1.25%

Table 2: Terminal-Bench Result

6. Conclusion

We built and used a terminal-focused evaluation pipeline that links Apertus and other strong open models to OpenCode and Terminal-Bench via an OpenAI-compatible SwissAI API.

All models could complete some realistic terminal tasks, while larger models such as DeepSeek-V3.1 and Kimi-K2 achieve higher accuracy on Terminal-Bench tasks compared to smaller.

A latency experiment showed that Apertus-8B responds fastest, while Qwen3-32B and especially Apertus-70B are slower. However, Qwen tends to produce longer, more didactic answers, whereas Apertus often replies with short code snippets, so latency partly reflects response style rather than serving performance alone.

From an infrastructure perspective, using SwissAI/CSCS requires models to be spun up and managed on Clariden/Alps before they can be used by OpenCode or Terminal-Bench, making operational constraints an integral part of the evaluation.

7. Future Work

- **Extend benchmarks.** Run Terminal-Bench on additional models and tasks. Investigating the causes of unresolved trials, specifically focusing on max_tokens limitation and API RetryError exceptions encountered. Aim: to evaluate the performance of a broader range of models with comparable parameter counts.
- **Richer latency study.** Repeat the non-interactive latency benchmark for more models and more diverse prompts to get more serving performance from answer length and verbosity.
- **Document the pipeline.** Write a concise pipeline document covering SwissAI setup, model spin-up, latency benchmarking, and Terminal-Bench usage to make our experiments easy to **reproduce** and **extend**.

References

[1] Project Apertus et al. Apertus: Democratizing open and compliant llms for global language environments, 2025.
[2] Maxime Martinasso, Mark Klein, and Thomas C. Schulthess. Alps, a versatile research infrastructure, 2025. Cray User Group (CUG) 2025 Best Paper Award.
[3] The terminal-bench Team. terminal-bench: a benchmark for ai agents in terminal environments, 2025. Accessed: 2025-5-19.
[4] Carlos E. Jimenez et al. Swe-bench: Can language models resolve real-world software engineering problems?, 2023.
[5] SWE-agent Team. Swe-agent: turning language models into software engineering agents on real github repositories, 2024.
[6] SST. Opencode: the open source ai coding agent, 2025.
[7] Swiss AI Research Platform. Swissai/cscs: Swiss ai research platform for large language models, 2025.
[8] Public AI. Public ai inference utility: a public access point for sovereign ai models, 2025.
[9] OpenRouter. Openrouter: one api for any model, 2025.
[10] LLM-Benchmark-CLI Team. Llm-benchmark-cli: a command-line interface for benchmarking language models, 2025.
[11] An Yang et al. Qwen3 technical report, 2025.
[12] Albert Q. Jiang et al. Mistral 7b, 2023.
[13] DeepSeek-AI et al. Deepseek-v3 technical report, 2025.
[14] Moonshot AI. Kimi-k2-instruct (revision 2f7e011), 2025.
[15] Amy Brand, Liz Allen, Micah Altman, Marjorie Hlava, and Jo Scott. Beyond authorship: Attribution, contribution, collaboration, and credit. *Learned Publishing*, 28(2):151–155, 2015.

Contributions

Author contributions using the CRediT framework [15]:
AK, SL: Conceptualization, Methodology, Software and infrastructure setup, Investigation, Analysis, Project administration, Visualization, Writing
IS: Conceptualization, Methodology, Resources.
RS: Supervision, Methodology, Technical Support, Resources.