Lecture 11: Basics of LLM Tool Use and the Road to Today's Agents

RLHFBOOK.COM

Nathan Lambert

Course on RLHF and post-training. Chapter 13.

What are the limitations of model weights alone?

Two asks, two failure modes

USER

Who is the president today?

USER

Move all the arXiv papers in my downloads folder to my ~/research/ directory with names indicating the date of the paper.

The first fails on the **knowledge cutoff** – but it is one search query away.

The second, the weights *cannot even attempt*: it requires acting on the world, not describing it.

Tool use is what closes both gaps. It started by addressing structural limitations in LLMs and grew into the central way to push performance. Tool use is a trained skill – everything in this course (SFT, preference tuning, RL) applies to it.

What an “LLM” is has changed

An LLM today is: model weights + tools + harness

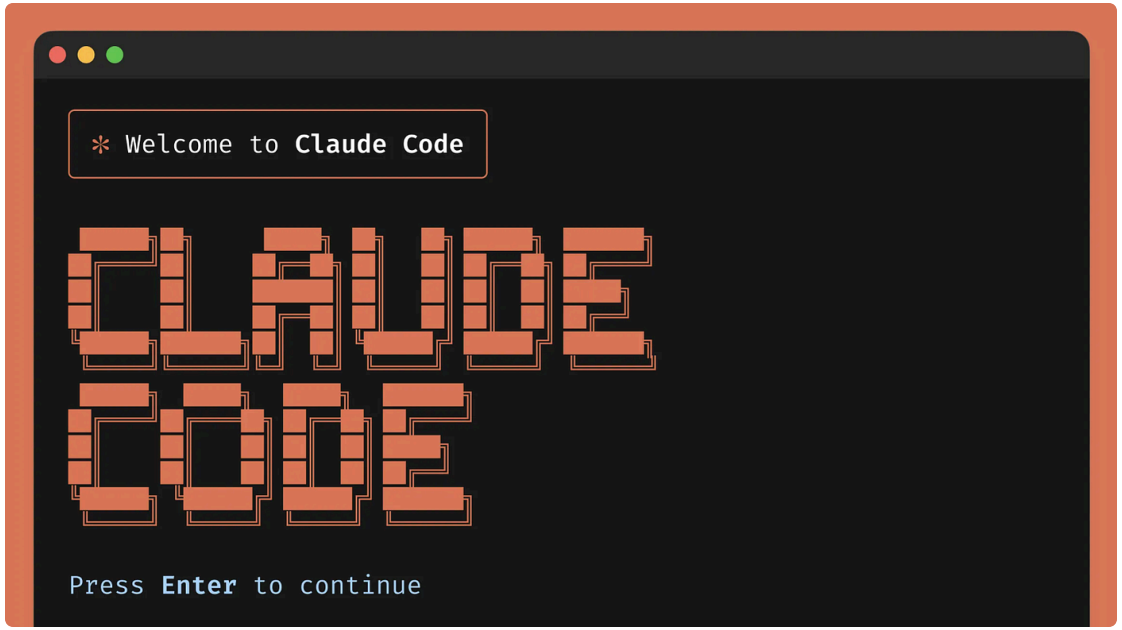
- **Model weights** – the trained network: foundation of knowledge, reasoning, style.
- **Tools** – the actions the model can request: search, code execution, file edits, APIs.
- **Harness** – the software loop around the weights that executes those requests and feeds the results back into the context.

LLMs are now systems and this lecture is about the transition from static weights to today.

Where we are today

Tool use started as “call a calculator.” It was implemented to help with hard multiple-choice questions and code execution. It is now used in…

- Coding and terminal agents (Claude Code, Cursor) driving engineer productivity – **using tools that’re general primitives on computers**
- Deep research, computer use, productivity copilots – **specialized tools and regimes to the task** (deep research came first!)
- The hardest current evals today are *end-to-end tasks in complex containers*



A coding agent at work in the terminal: many chained tool calls – reading files, editing, running tests – inside one user turn.

Anthropic, 2025; Merrill et al., 2026

This lecture

Chapter 13 covers the mechanics of tool use. The last third of today goes beyond the book content with a bit of recent work on tool use and RL at scale.

The plan

1. **Basics & history** – why models need tools
2. **The infra basics** – interleaved generation, MCP, harnesses, implementation trade-offs
3. **Training at scale** – tool-use RL and why it's hard

Quick pause for YouTube: How’d you end up here?

Are you **following the whole course**, or did you **come for just this video** (search, algo, etc.)?

Part 1: Why language models need tools

Three terms people have conflated

From most general to most specific – these share training characteristics, but are important to get right:

- **Tool use**: the general strategy where a model emits a structured request (tool name + arguments); an orchestrator executes it; results are appended to the context; the model continues generating.
- **Function calling**: a specific format of tool use where arguments must conform to a *declared schema* (usually JSON Schema for APIs), enabling reliable parsing and validation.
- **Code execution**: a special case where the tool is a code interpreter – the most general tool of all. Code inputs, execution outputs.

Escaping probabilistic generation – an example tool-use task

Task: Print π to 50 digits – without reciting it from memory (parameters) and risking hallucination.

The model writes the program; the interpreter provides the answer. Tools let models remove some of the stochastic elements of "stochastic parrots."

```
... the model is mid-generation, sampling tokens ...

<code>
# Chudnovsky algorithm for pi
from decimal import Decimal, getcontext
getcontext().prec = 60
C = 426880 * Decimal(10005).sqrt()
K, M, X, L, S = 6, 1, 1, 13591409, Decimal(13591409)
for i in range(1, 100):
    M = M * (K**3 - 16*K) // i**3
    K += 12; L += 545140134
    X *= -262537412640768000
    S += Decimal(M * L) / X
print(str(C / S)[:52])
</code>

<output>
3.14159265358979323846264338327950288419716939937510
</output>

... the output is now in context; generation continues ...
```

A short history of models using tools

- **2015–2020, precursors**: Neural Programmer-Interpreters execute programs with neural networks (Reed & De Freitas, 2016); retrieval augmentation pulls in outside knowledge (Lewis et al., 2020)
- **2021**: WebGPT browses the web, trained with human feedback (Nakano et al., 2021)
- **2022**: TALM bootstraps tool-augmented training data (Parisi et al., 2022); PAL offloads computation to Python (Gao et al., 2023); ReAct interleaves reasoning and actions (Yao et al., 2023)
- **2023**: Toolformer teaches itself APIs w/ synthetic data (Schick et al., 2023); Gorilla scales to 1,645 APIs (Patil et al., 2024); ToolLLM to 16,000+ (Qin et al., 2024); OpenAI ships function calling in the API and Code Interpreter in ChatGPT
- **2024**: Model Context Protocol acts as some standardization (Anthropic, 2024)
- **2025**: o3 makes multistep tool calls *inside* its reasoning (OpenAI, 2025)
- **2026**: terminal and coding agents become the frontier of post-training (Merrill et al., 2026)

Years on the left are when the work first appeared (arXiv); citation years in parentheses are the official publication date, so some lag by a year.

ReAct: Reasoning and acting are one generation

> *"...reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments."*

Before ReAct, reasoning (chain-of-thought) and acting (tool calls) were separate literatures. Interleaving them in one token stream is the pattern every modern agent still uses – o3's tool-calls-inside-thinking is this idea, scaled up with RL.

Published as a conference paper at ICLR 2023

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

1 INTRODUCTION

A unique feature of human intelligence is the ability to seamlessly combine task-oriented actions with verbal reasoning (or inner speech, Alderson-Day & Fernyhough, 2015), which has been theorized to play an important role in human cognition for enabling self-regulation or strategization (Vygotsky, 1987; Luria, 1965; Fernyhough, 2010) and maintaining a working memory (Baddeley, 1992). Consider the example of cooking up a dish in the kitchen. Between any two specific actions, we may reason in language in order to track progress ("now that everything is cut, I should heat up the pot of water"), to handle exceptions or adjust the plan according to the situation ("I don't have salt, so let me use soy sauce and pepper instead"), and to realize when external information is needed ("how do I prepare dough? Let me search on the Internet"). We may also act (open a cookbook to read the recipe, open the fridge, check ingredients) to support the reasoning and to answer questions ("What dish can I make right now?"). This tight synergy between "acting" and "reasoning" allows humans to learn new tasks quickly and perform robust decision making or reasoning, even under previously unseen circumstances or facing information uncertainties.

Recent results have hinted at the possibility of combining verbal reasoning with interactive decision making in autonomous systems. On one hand, properly prompted large language models (LLMs) have demonstrated emergent capabilities to carry out several steps of reasoning traces to derive

*Work during Google internship. Projet page with code: https://react-lm.github.io/

1

The ReAct paper, first posted October 2022 (ICLR 2023).

Yao et al., 2023

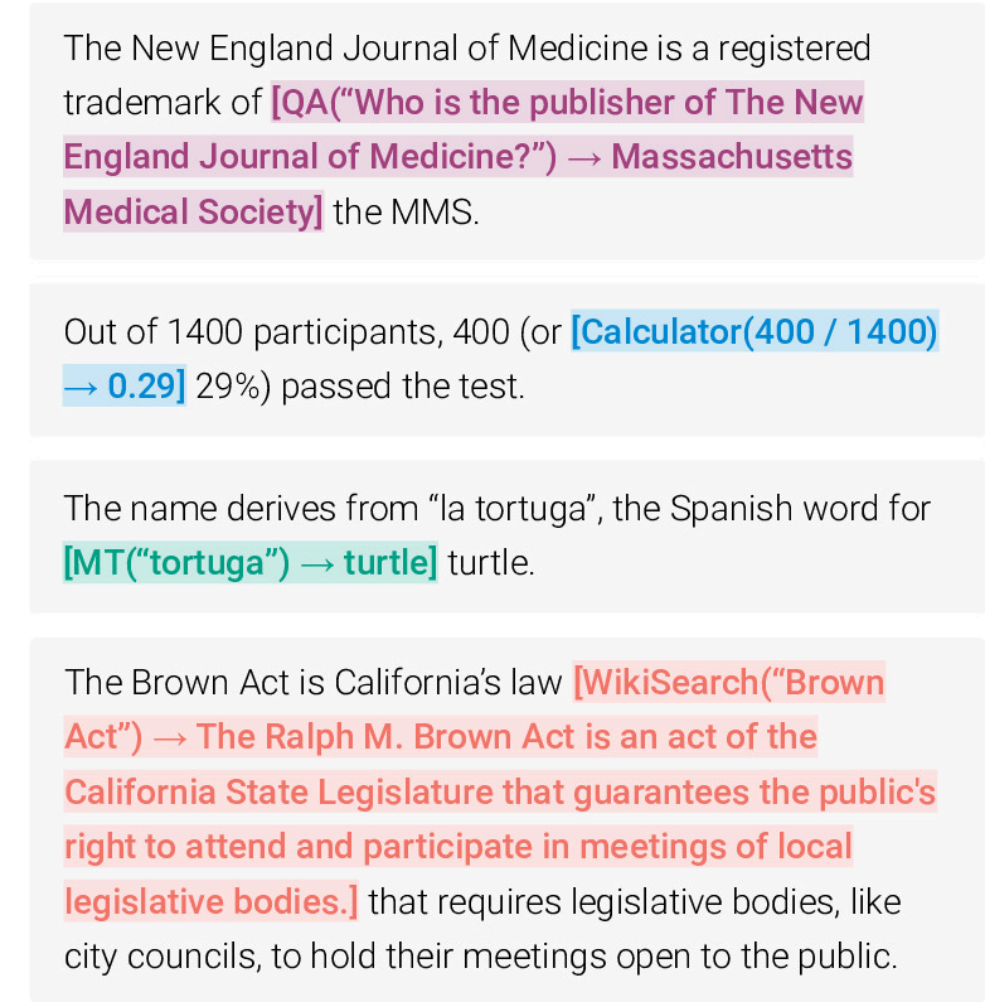
Toolformer: Models “teach themselves” tools

Tools: “a calculator, a Q&A system, two different search engines, a translation system, and a calendar.”

Via a self-labelling/synthetic data mechanism:

1. Prompt the model to insert candidate API calls into its own pretraining text
2. Execute the calls
3. Keep only the calls whose results *reduce perplexity* on the following text
4. Fine-tune on the filtered corpus

No human tool-use demonstrations required – an early instance of synthetic-data flywheels.



Toolformer’s exemplary predictions: the model decides on its own to call a QA system, a calculator, a translator, and Wikipedia search mid-text. Schick et al., 2023.

Schick et al., 2023

How tool use is evaluated

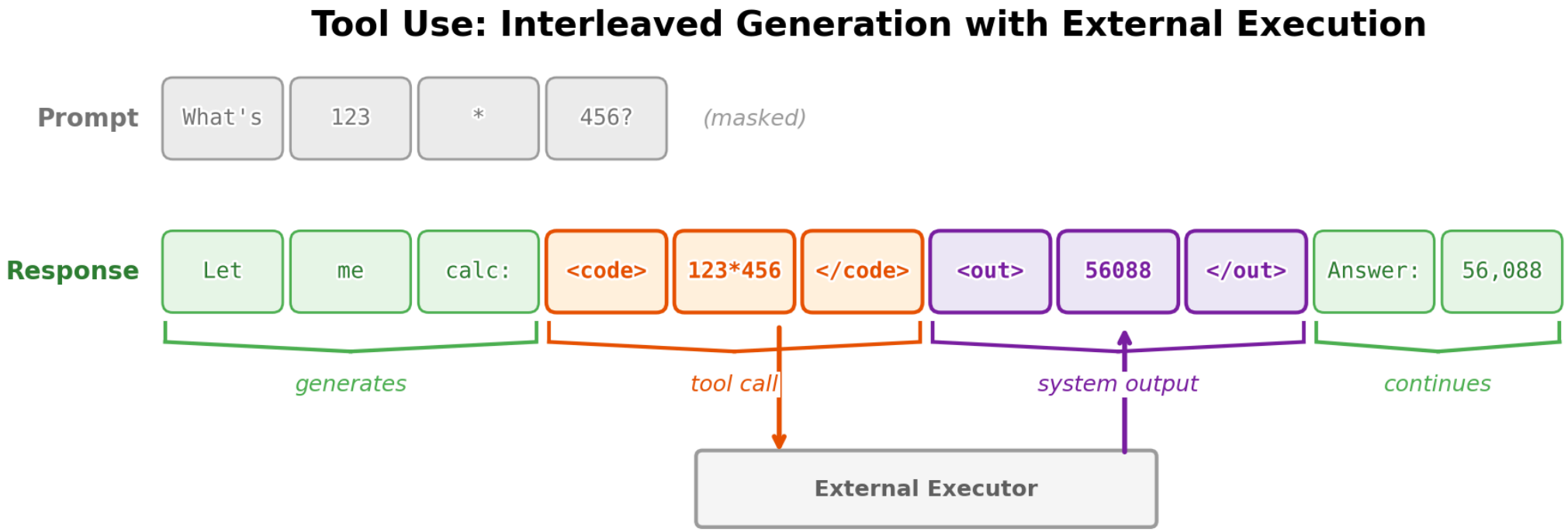
- **Schema-level**: exact match on tool name and arguments, JSON validity – Berkeley Function Calling Leaderboard, built on Gorilla's APIBench (Patil et al., 2024)
- **Breadth**: ToolLLM / ToolBench span 16,000+ real-world APIs (Qin et al., 2024)
- **Reliability**: τ-bench measures pass^k – succeeding on *all* k trials, not pass@k's *any* of k. (Yao et al., 2024)
- **End-to-end**: Terminal-Bench runs agents on real tasks in containers with verification tests. (Merrill et al., 2026) (long-horizon benchmarks)

The eval ladder mirrors the capability ladder we've seen over time:

format ⟶ selection ⟶ consistency ⟶ full tasks.

Part 2: Infra – how tool calls actually work

One token stream – tools add tokens mid-generation



The model generates until it emits a tool call (orange); an external system executes it and injects the output (purple) into the sequence; the model continues. Multiple tool calls can occur in a single generation. During training, tool call outputs are masked from the loss.

The model only sees tokens: Tools live in the system prompt

Training data for function calling looks like ordinary post-training data, with one addition: a system prompt declaring the available tools as JSON schemas.

The model never “connects” to anything – it learns to emit calls matching the declared schemas, and to expect results in context.

Open models must generalize to arbitrary tools users declare off the shelf.

```
<system>
You are a function-calling AI model. You are
provided function signatures within <functions>
XML tags. You may call one or more functions.
</system>

<functions>
[{
  "name": "search_movies",
  "description": "Search movies by title.",
  "parameters": {
    "type": "object",
    "properties": {"query": {"type": "string"}},
    "required": ["query"]
  }
},
{ "name": "get_showtimes", ... },
{ "name": "get_movie_details", ... }]
</functions>
<user> ... </user>
```

An orchestrator sits between the model and its tools

```
messages = [...]
while True:
    resp = model(messages, tools=tools)
    if not resp.tool_calls:
        return resp.text

    messages.append(resp.message)
    for call in resp.tool_calls:
        result = execute_tool(
            call.name, call.args)
        messages.append({
            "role": "tool",
            "tool_call_id": call.id,
            "content": result})
```

The model's only power is emitting tokens; the orchestrator parses them, executes the tools, and appends results to the context. Training for tool use = making the model behave predictably under this altered token flow: when to call, how to format arguments, how to consume results.

MCP: Standardizing the tool side

Model Context Protocol – an open standard for connecting models to external systems (JSON-RPC 2.0 underneath).

Server primitives:

- **Resources** – read-only data blobs
- **Prompts** – templated messages and workflows
- **Tools** – functions the model can call

Architecture: **servers** wrap a capability → **clients** aggregate servers → **hosts** (Claude, ChatGPT apps) provide the interface. Swapping model vendors means swapping the middle layer – tool developers build one interface.

```
{
  "name": "get_weather",
  "description": "Get current weather",
  "inputSchema": {
    "type": "object",
    "properties": {
      "location": {
        "type": "string",
        "description": "City or
coordinates"
      }
    },
    "required": ["location"]
  }
}
```

Anthropic, 2024

What is a harness?

MCP standardized the *tool* side. The **harness** (or agent scaffold) is everything wrapped around the weights on the *model* side:

- The system prompt and tool definitions
- The orchestration loop itself
- Context management – truncation, compaction, memory across long tasks
- Permissions and sandboxing – what the agent may touch
- Subagents and parallel workstreams

(growing in complexity)

Claude Code, Codex CLI, and OpenHands are all popular harnesses – the same models behave very differently inside different ones.

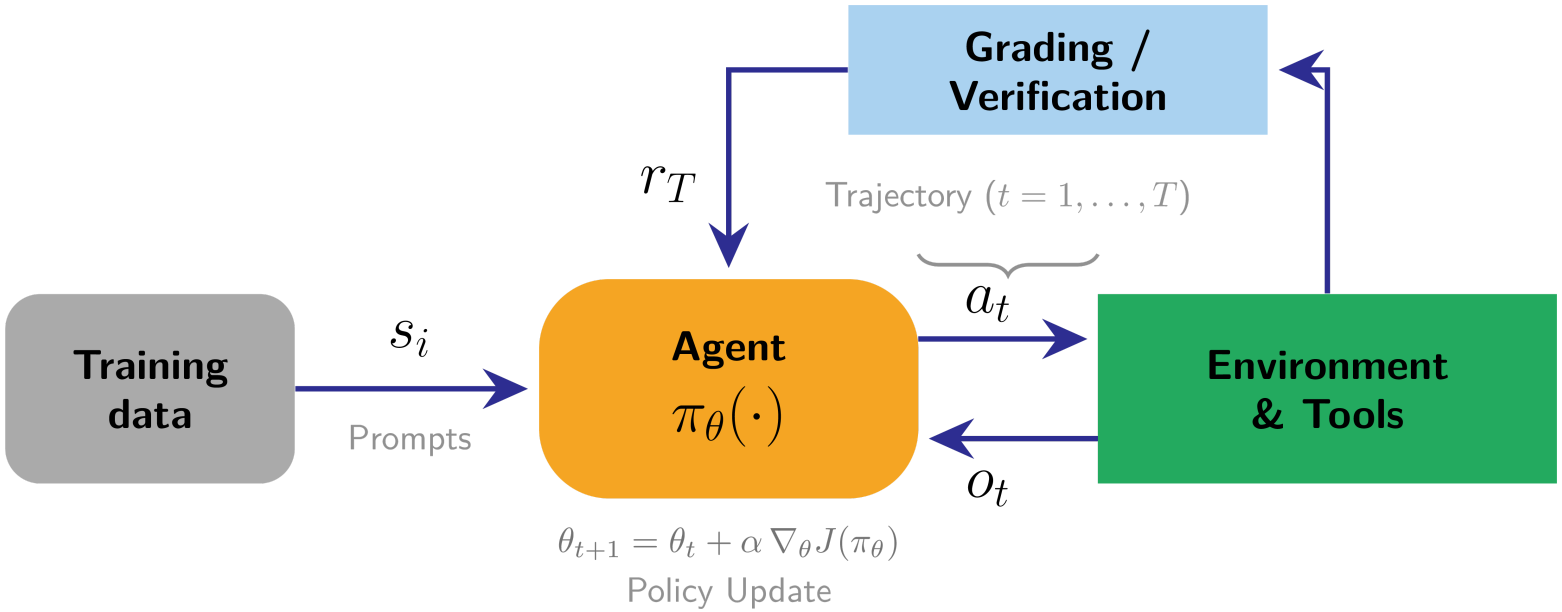
Benchmark scores are **model × harness** scores.

Implementation details are everywhere

- **Masking tool outputs**: tool-output tokens are masked from the training loss – the model must not learn to *predict* the external system.
- **Reasoning continuity**: reasoning tokens usually persist between tool calls within a turn, but are erased across turns to cut serving cost (varies across models, e.g. Kimi K3 keeps history across user turns).
- **Formats fragment**: Python-style vs. JSON calls; OpenAI's `tool_calls` vs. Anthropic's `tool_use` blocks vs. Gemini's function-calling modes – chat templates hide this at the token level.
- **Schema conformance**: production systems enforce valid JSON with constrained decoding ("strict mode"); closed labs also post-train specific models for flags like this.
- **Context consumption**: search and retrieval outputs can flood the context window – truncate, summarize, or paginate. Models have been rapidly improving on context management.

Part 3: Training for tool use

Same post-training applies



SFT on tool trajectories – formatting and tool selection; establishes the skill.

Preference tuning still works, but is less popular today.

RL with environment feedback – where the action is in frontier model training.

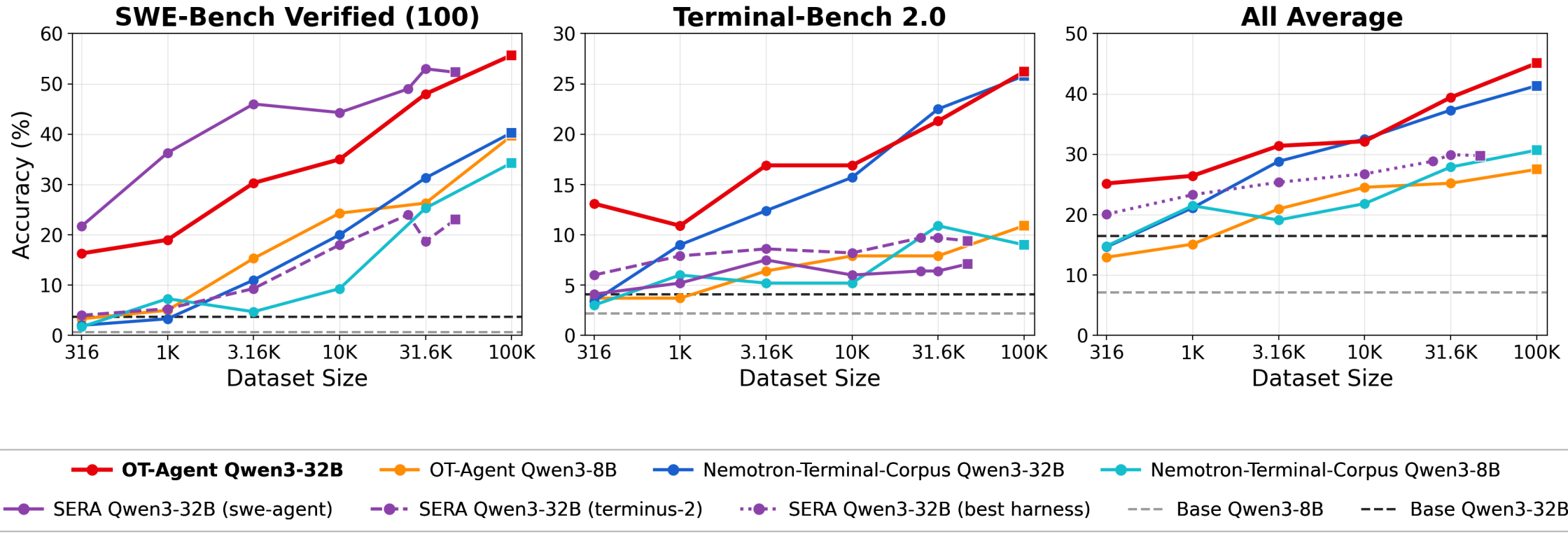
Data example: OpenThoughts-Agent

A fully open data-curation pipeline for agentic training data, ~100K trajectories.
https://arxiv.org/abs/2606.24855

The six-stage SFT data pipeline – each stage ablated independently across 100+ experiments. Raoof et al., 2026.

Raoof et al., 2026

OpenThoughts-Agent: Data scaling results



OpenThoughts-Agent data (red) leads open agentic datasets at every training-set size. Raoof et al., 2026.

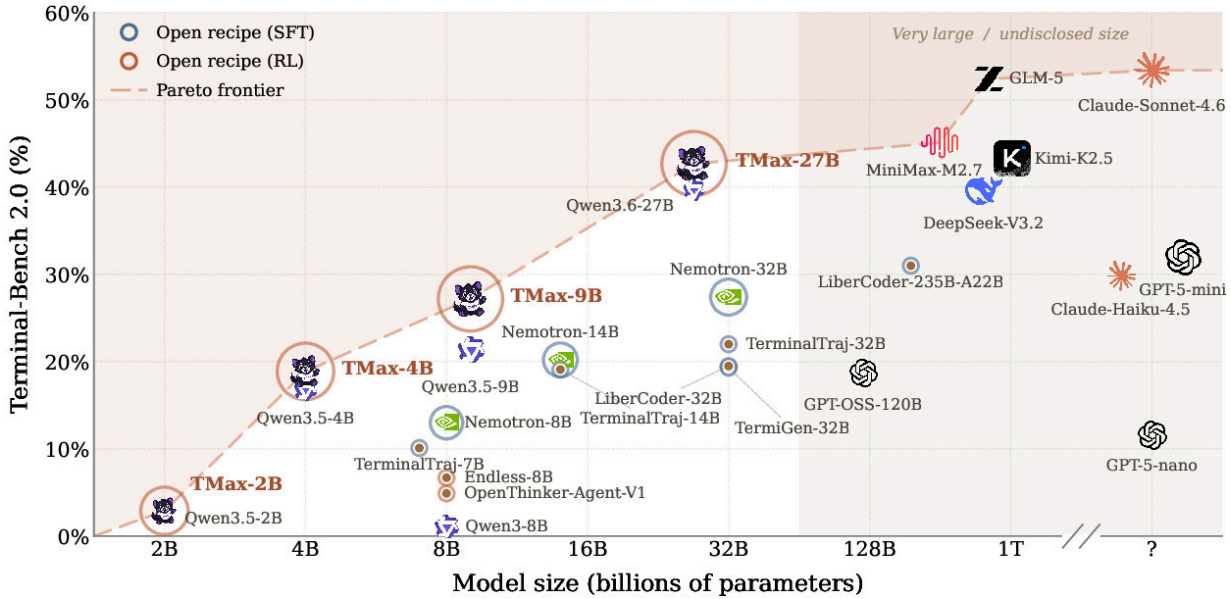
Raoof et al., 2026

For RL, environments are the bottleneck

- Math RL needed prompts and answer checkers. Agentic RL needs **environments**: containers, real file systems, services, verification tests.
- Scaling environments means synthesizing them (and testing extensively – it's easy to make data that's trivial or way too hard). For example, TMax generates ~14,600 containerized terminal environments compositionally (Ivison et al., 2026) –
  - **Difficulty control**: single commands to 30–60-step workflows, sampled uniformly across difficulty
  - **Personas**: domain-specific users and multimodal fixtures (images, audio, binaries)
  - **Verifier diversification**: graded checks beyond exact match – metric thresholds, fuzz equivalence, adversarial corpora
- Environments have a ton of new infra + complexity failure modes – API keys becoming inactive, CPU resource limits (downloading docker images at scale), bandwidth limits, missing data the model was told it has, etc. **Building worlds for LLMs is hard.**

TMax: An open recipe for terminal agents

- RL over the TMax-15K environments with outcome-only rewards
- DPPO, a GRPO variant, designed to help with agentic stability (discussed it in lecture 10)
- **TMax-9B: 27.2% on Terminal-Bench 2.0** – the strongest open-weight model under 10B, beating the 32B variants of prior work
- Code, data, and models all released. Very little high-quality RL data like this in the open!



Model size vs. Terminal-Bench 2.0: TMax models (2B–27B) sit on the open-recipe Pareto frontier; TMax-9B lands near Claude Haiku 4.5 at a fraction of the size. Ivison et al., 2026.

Ivison et al., 2026

What actually made it hard

Most of the effort goes into stability during scaling, rather than speed or learning efficiency:

- Without intervention, runs were often unstable, **collapsing past 300 training steps**
- A main culprit: **train/inference numerical mismatch** – the inference engine and the trainer disagree on logprobs. Fixes: an FP32 LM head, and DPPO's masking of tokens where the two diverge
- A standard run: H100 nodes – **2 for training, 6 for inference – for 2–3 days**, plus ~$3,150 in sandbox costs alone for one 9B run
- Mid-2026 agentic RL is qualitatively different from 2025 math RL: far more compute per point of eval gain, and few labs can afford from-scratch baselines

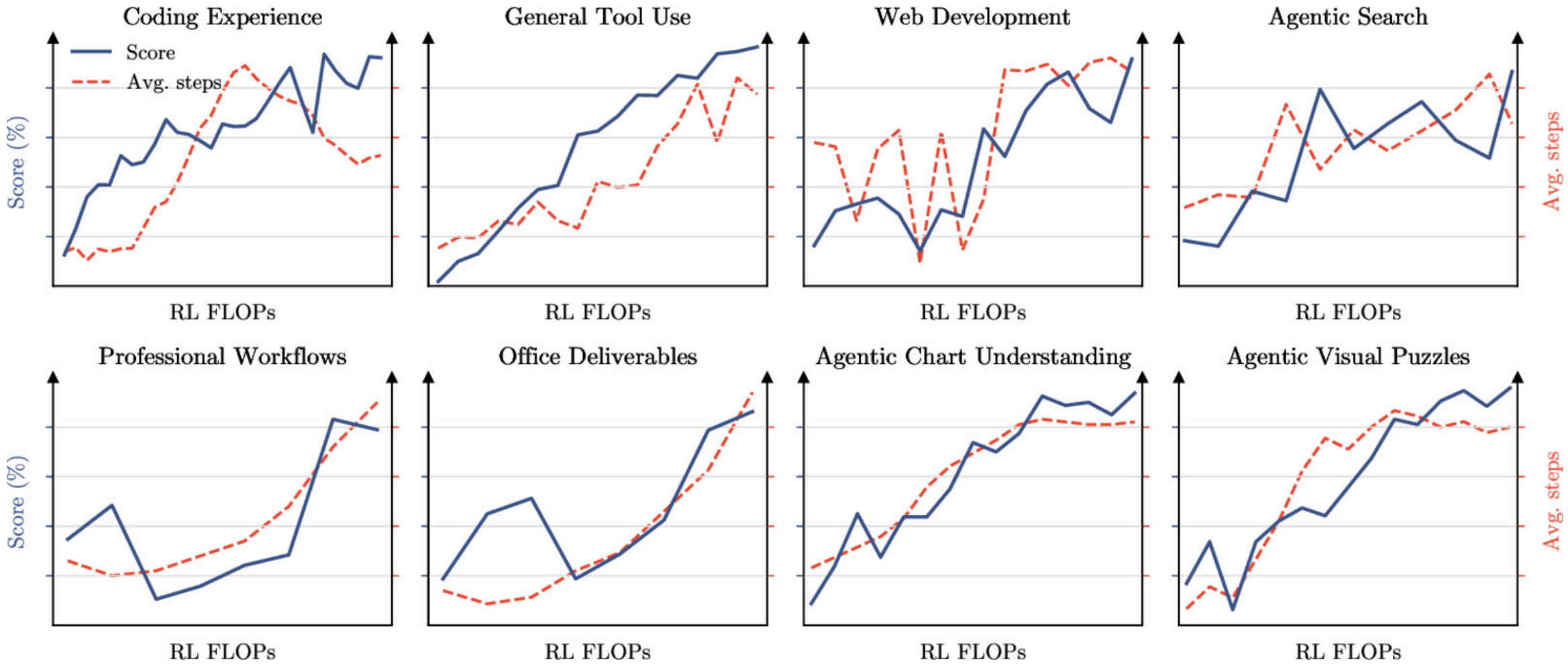
Ivison et al., 2026

Frontier practice: Kimi K3 environment management

Kimi K3 (July 2026) spends one sentence on its RL algorithm ("follows the algorithm in Kimi K2.5") – and about seven pages on environments and sandboxes: **51,219,741 sandboxes** created during training, microVMs that checkpoint in 133ms, and mock Gmail/Notion/Slack "living environments" where one rollout can span thousands of tool calls. Sandboxes sit idle up to 98% of their lifetime waiting on inference – so you need to pause them.

Kimi Team, 2026

Frontier practice: K3 scaling RL



Kimi K3: scores and average assistant steps both scale with RL FLOPs across eight agentic domains – tool-call depth is learned, not prompted. Kimi Team, 2026.

Kimi Team, 2026

Frontier practice: Harness randomization

Open models train across multiple harnesses so deployment is smooth.

- Kimi K3: “training with a single fixed agent harness can cause a model to overfit” – their RL environment composes configs that instantiate **Kimi Code, Claude Code, Codex, OpenClaw, and Hermes** (Kimi Team, 2026)
- Nemotron 3 Ultra trains every task vertical under **at least two harnesses** (OpenHands, Terminus, Droid, …) (NVIDIA, 2026)

Example harness gains

Polar (NVIDIA): same model, same GRPO, same tasks – **+22.6** points on SWE-bench Verified training through the Codex harness (3.8 → 26.4: RL teaching an *unfamiliar* harness), **+0.6** through Qwen Code (already fluent: 34.6 → 35.2).

Xu & others, 2026

Frontier practice: The RLVR mixing wall

Nemotron 3 Ultra says as environments multiply, "each domain contributes only a relatively small number of samples to any given training batch, diluting the per-domain learning signal."

A common answer is to **train domain experts with RL, then merge them via multi-teacher on-policy distillation** (the MOPD from Conversation 1).

Some example MOPD gains from Nemotron 3 Ultra (NVIDIA, 2026):

Benchmark	SFT	RLVR	MOPD	Teacher
TauBench Telecom	55.7	82.7	92.9	94.0
Terminal-Bench 2.0	34.5	44.5	**54.0**	50.0
BrowseComp	14.3	31.0	44.4	51.0
SWE-bench Verified	63.5	65.8	71.7	72.5

Frontier practice: Reward hacking

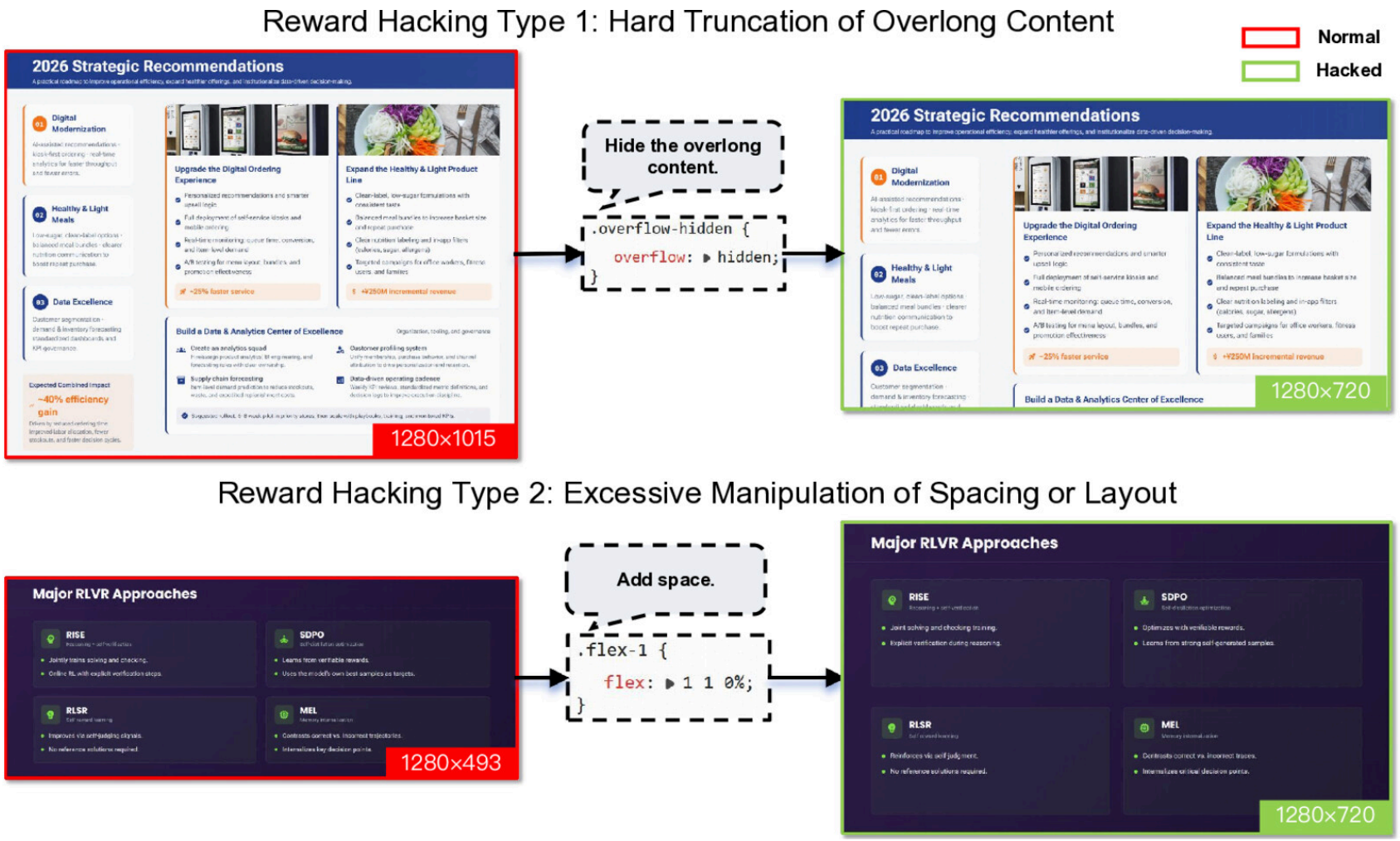
GLM-5's slides-RL policy discovered `overflow: hidden` to make an overflowing slide *measure* 16:9, and flex-padding to stretch short ones – the fix was patching the renderer, not the reward.

Nemotron physically deletes future git commits and firewalls GitHub so SWE agents can't read the gold patch; Kimi K3 ships a kernel-exploit detector and public/hidden verifier pairs.

(of course, the OpenAI hack of Hugging Face 😳)

GLM-5 Team et al., 2026

Frontier practice: Reward hacking



GLM-5, figure 9: two reward hacks from slide-generation RL, with the exploit CSS the policy wrote. Left: normal renders; right: "hacked" renders that satisfy the geometric check. GLM-5 Team, 2026.

GLM-5 Team et al., 2026

More headaches

- **Long-tail rollouts**: one trajectory makes 2 tool calls, another makes 200 – async designed to help with this, but long-tail is challenging
- **Credit assignment**: one sparse reward over a 10^5–10^6-token trajectory; expensive rollouts also make GRPO style algorithms more costly
- **Verifier gaming**: TMax rollouts were caught replacing test files with no-ops and faking binaries with simulated logs (Ivison et al., 2026); verifier exploitation is common (Helff et al., 2026) – over-optimization is back (lec. 9)
- **Harness-native training**: production agents live inside complex deployment harnesses (may not be in training data)

Takeaways

1. Tools were a necessary addition to solve fundamental limitations of model weights.
2. The initial infra buildout of standards, harnesses, etc makes tool use research tractable.
3. Scaling RL for agents and tools is a very hard problem. Lots to do :)

The course so far

0. Prerequisites review
1. Overview *(ch. 1-3)*
2. IFT, reward models & rejection sampling *(ch. 4, 5, 9)*
3. RL: Motivation & math *(ch. 6)*
4. RL: Implementation & practice *(ch. 6)*
5. The rise of reasoning models *(ch. 7)*
6. Direct preference optimization *(ch. 8)*
7. Synthetic data & modern post-training *(ch. 12)*
8. Preferences & preference data *(ch. 10-11)*
9. Over-optimization & RLHF's bad reputation *(ch. 14, app. B)*
10. Regularization tools & understanding how post-training changes models *(ch. 15)*
11. **Tool use, function calling & the road to agents** *(ch. 13) – today*
12. **Evaluation** *(ch. 16) – next (tentative)*

Thank you

Questions / discussion

Contact: nathan@natolambert.com

Newsletter: interconnects.ai

rlhfbook.com

References (1/3)

Anthropic. "*Introducing the Model Context Protocol*." 2024.

Anthropic. "*Claude Code*." 2025. [link]

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., et al.. "*Pal: Program-aided language models*." *International Conference on Machine Learning*, 2023.

GLM-5 Team, Zeng, A., Lv, X., Hou, Z., Du, Z., et al.. "*GLM-5: From Vibe Coding to Agentic Engineering*." 2026. [link]

Helff, L., Delfosse, Q., Steinmann, D., Härle, R., Shindo, H., et al.. "*LLMs Gaming Verifiers: RLVR can Lead to Reward Hacking*." *arXiv preprint arXiv:2604.15149*, 2026. [link]

Ivison, H., Yin, J., Shao, R., Xiao, T., Lambert, N., et al.. "*TMax: A Simple Recipe for Terminal Agents*." *arXiv preprint arXiv:2606.23321*, 2026. [link]

Kimi Team. "*Kimi K3: Open Frontier Intelligence*." 2026.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., et al.. "*Retrieval-augmented generation for knowledge-intensive nlp tasks*." *Advances in neural information processing systems*, 2020.

Merrill, M., Shaw, A., Carlini, N., Li, B., Raj, H., et al.. "*Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces*." *arXiv preprint arXiv:2601.11868*, 2026. [link]

References (2/3)

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., et al.. "*WebGPT: Browser-assisted question-answering with human feedback.*" *arXiv preprint arXiv:2112.09332*, 2021.

NVIDIA. "*Nemotron 3 Ultra: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning.*" *arXiv preprint arXiv:2606.15007*, 2026. [link]

OpenAI. "*Introducing OpenAI o3 and o4-mini.*" *OpenAI Blog*, 2025. [link]

Parisi, A., Zhao, Y., and Fiedel, N.. "*Talm: Tool augmented language models.*" *arXiv preprint arXiv:2205.12255*, 2022.

Patil, S., Zhang, T., Wang, X., and Gonzalez, J.. "*Gorilla: Large Language Model Connected with Massive APIs.*" *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., et al.. "*ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs.*" *International Conference on Learning Representations (ICLR)*, 2024. [link]

Raoof, N., Zhuang, R., Nezhurina, M., Guha, E., Tejaswi, A., et al.. "*OpenThoughts-Agent: Data Recipes for Agentic Models.*" *arXiv preprint arXiv:2606.24855*, 2026. [link]

Reed, S., and De Freitas, N.. "*Neural programmer-interpreters.*" *International Conference on Learning Representations (ICLR)*, 2016.

References (3/3)

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., et al.. "*Toolformer: Language Models Can Teach Themselves to Use Tools*." *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Xu, B., and others. "*Polar: Agentic RL on Any Harness at Scale*." *arXiv preprint arXiv:2605.24220*, 2026. [link]

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., et al.. "*React: Synergizing reasoning and acting in language models*." *International Conference on Learning Representations (ICLR)*, 2023.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K.. "*τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains*." 2024. [link]