CS 668 Applied Large Language Models
Spring Semester, 2026

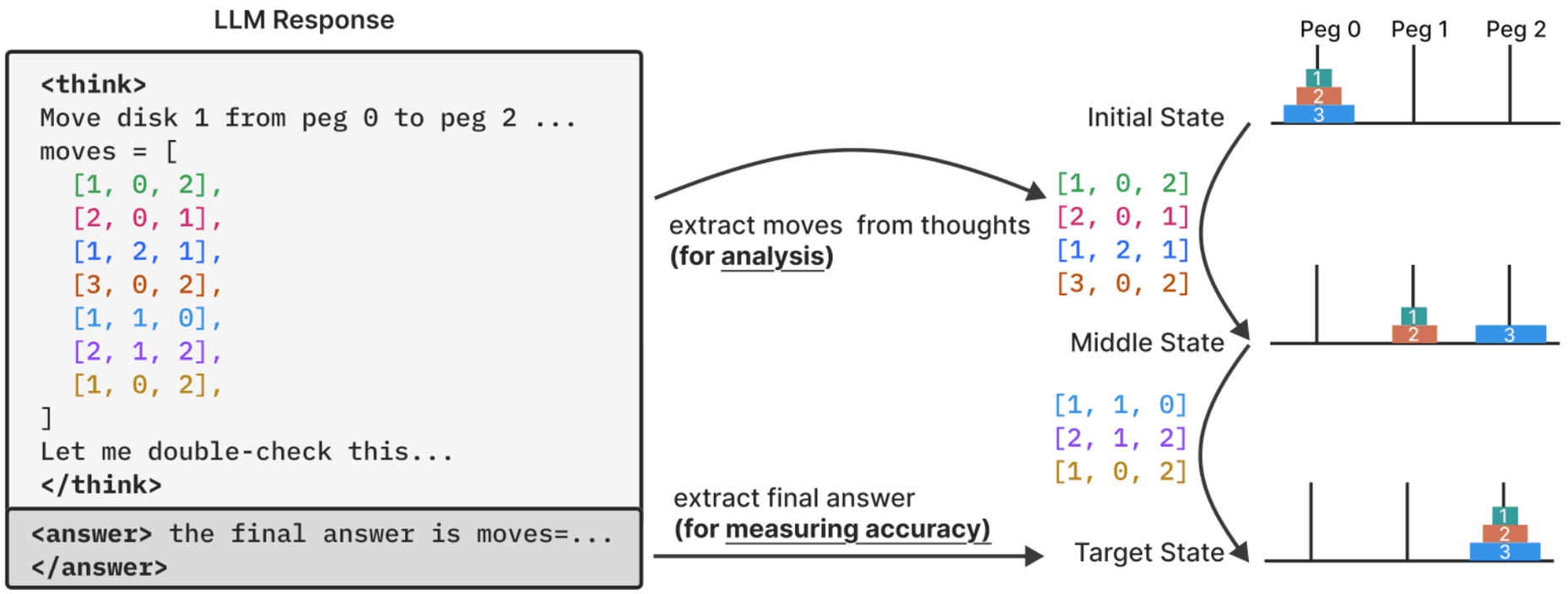
Doc 22 News, Activation Steering, Performance Issues
Apr 14, 2026

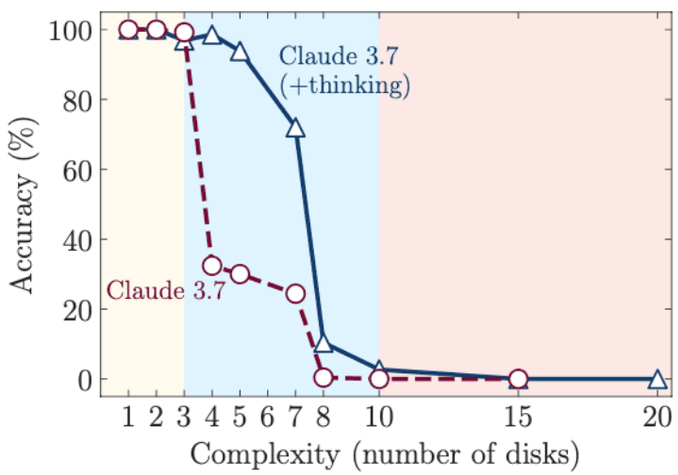
Copyright ©, All rights reserved. 2026 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.

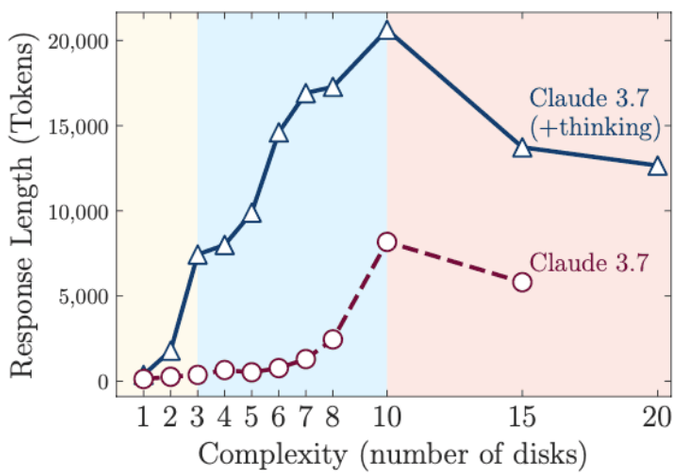
The Illusion of Thinking

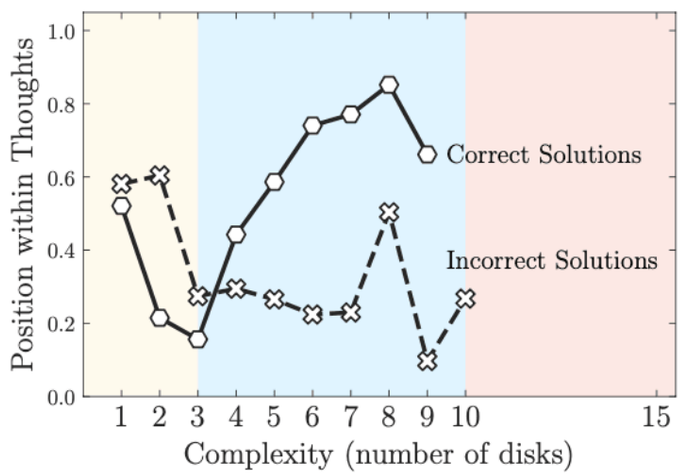
Parshin Shojaee*†, Iman Mirzadeh*, Keivan Alizadeh, Maxwell Horton, Samy Bengio, Mehrdad Farajtabar

https://machinelearning.apple.com/research/illusion-of-thinking









The Price Is Not Right

Neuro-Symbolic Methods Outperform VLAs on
Structured Long-Horizon Manipulation Tasks with Significantly Lower
Energy Consumption
https://arxiv.org/pdf/2602.19260
22 Feb 2026

Fine-tuned open-weight VLA model vs a neuro-symbolic architecture on Towers of Hanoi

Neuro-symbolic architecture
- Neural networks (for pattern recognition)
- Symbolic planner

Neuro-symbolic architecture
- 95% success

VLA
- 34% success
- 2 order of magnitude more energy

Bitter (AI) Lesson

Rich Sutton UT Austin, 2019

https://www.cs.utexas.edu/~eunsol/courses/data/bitter_lesson.pdf

Biggest lesson from 70 years of AI research

General methods that leverage computation are ultimately the most effective, and by a large margin

Chess

- Researchers tried using human knowledge to create chess programs
- Just using deep search was more effective

Go

- Researchers tried using human knowledge to create go programs
- Deep search and self-play were more effective

Speech Recognition

- Forget grammar etc, use Markov models, deep learning, LLMs

Computer Vision

- Deep learning

Bitter (AI) Lesson Rich Sutton UT Austin, 2019

Bitter lesson

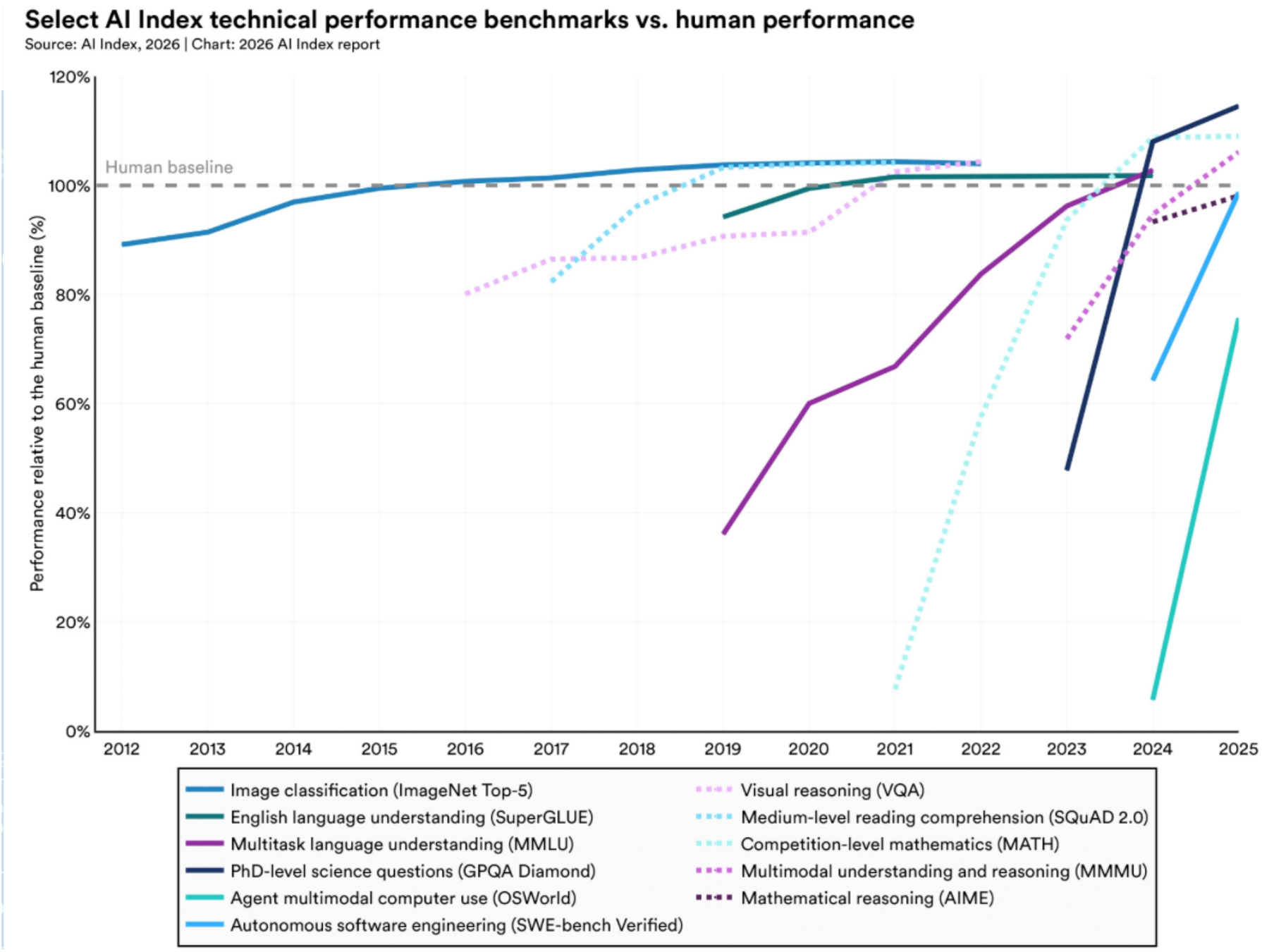
Building in how we think we think does not work in the long run

Eventual success is tinged with bitterness,

because it is success over a favored, human-centric approach

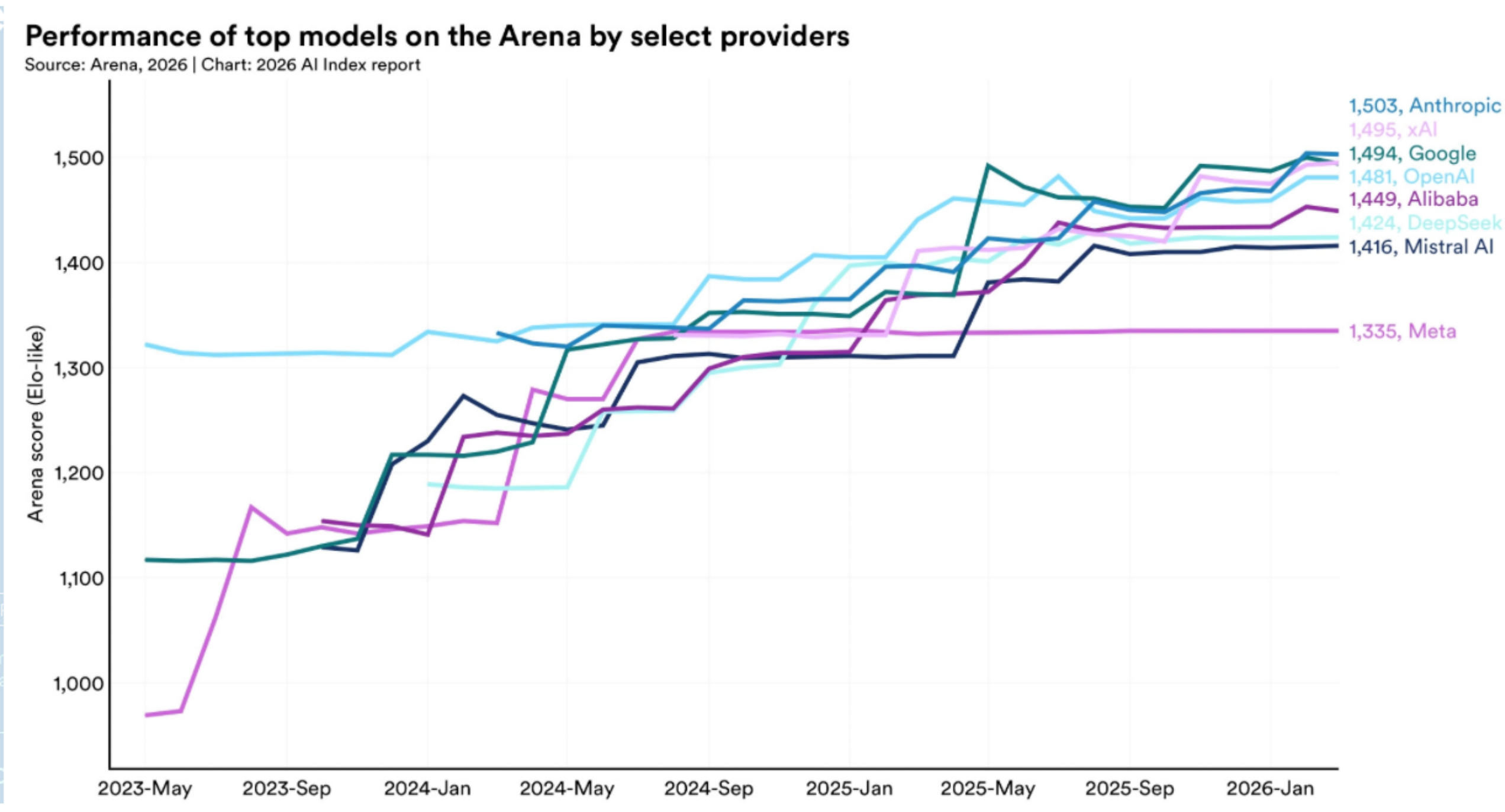
Two methods that seem to scale arbitrarily are search and learning

Stanford 2026 AI Index Report



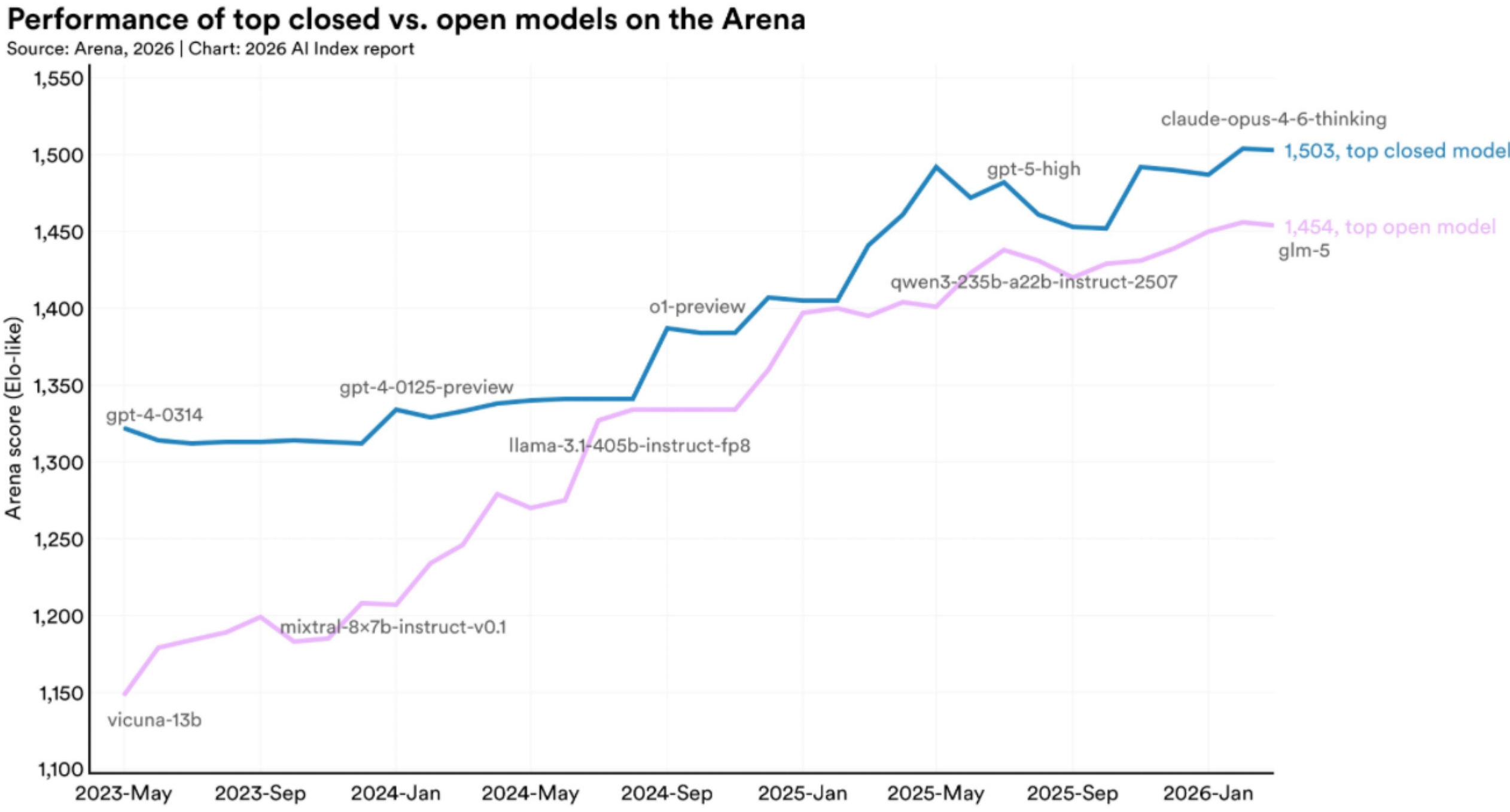
https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report



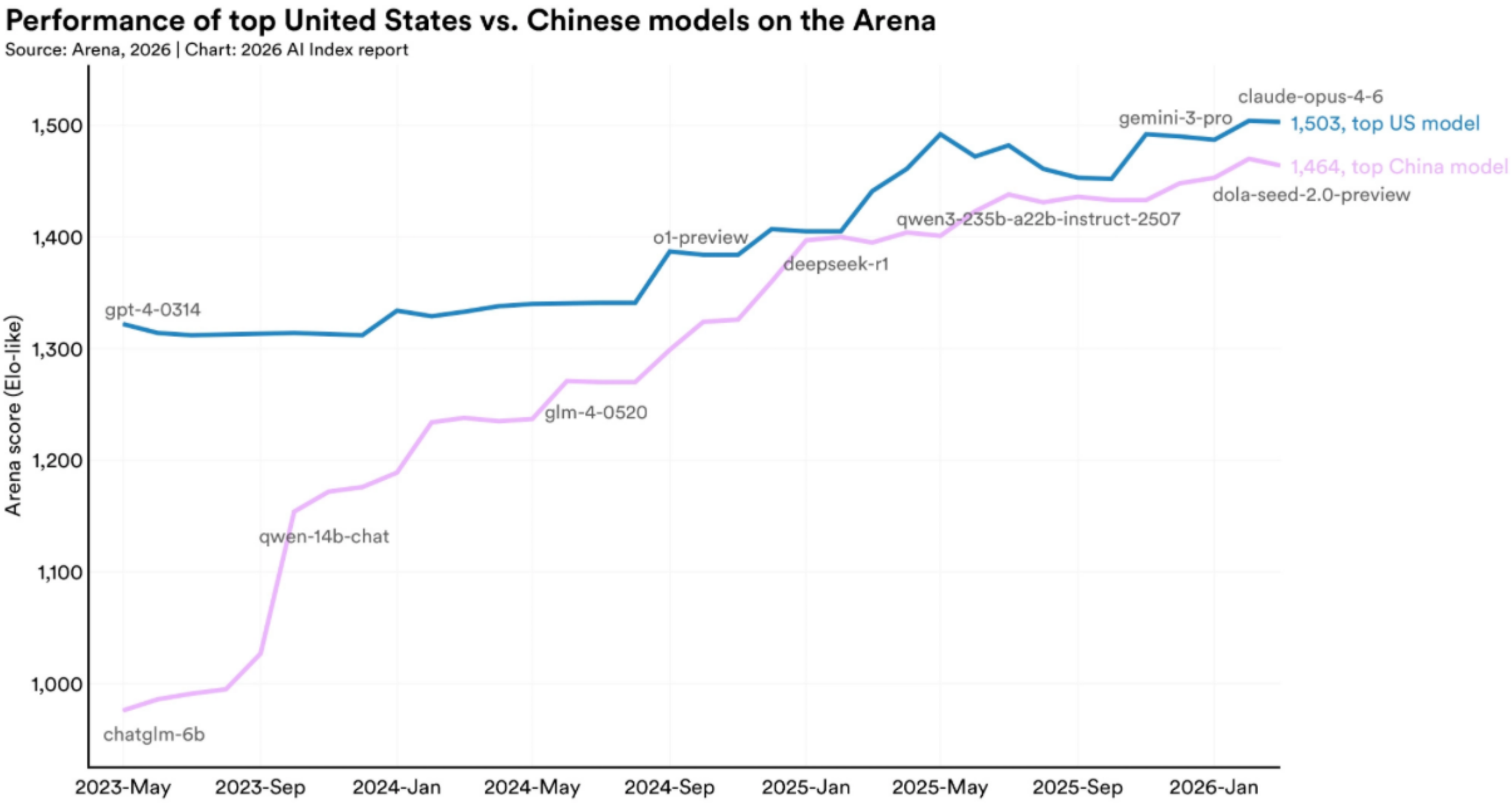
https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report



https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

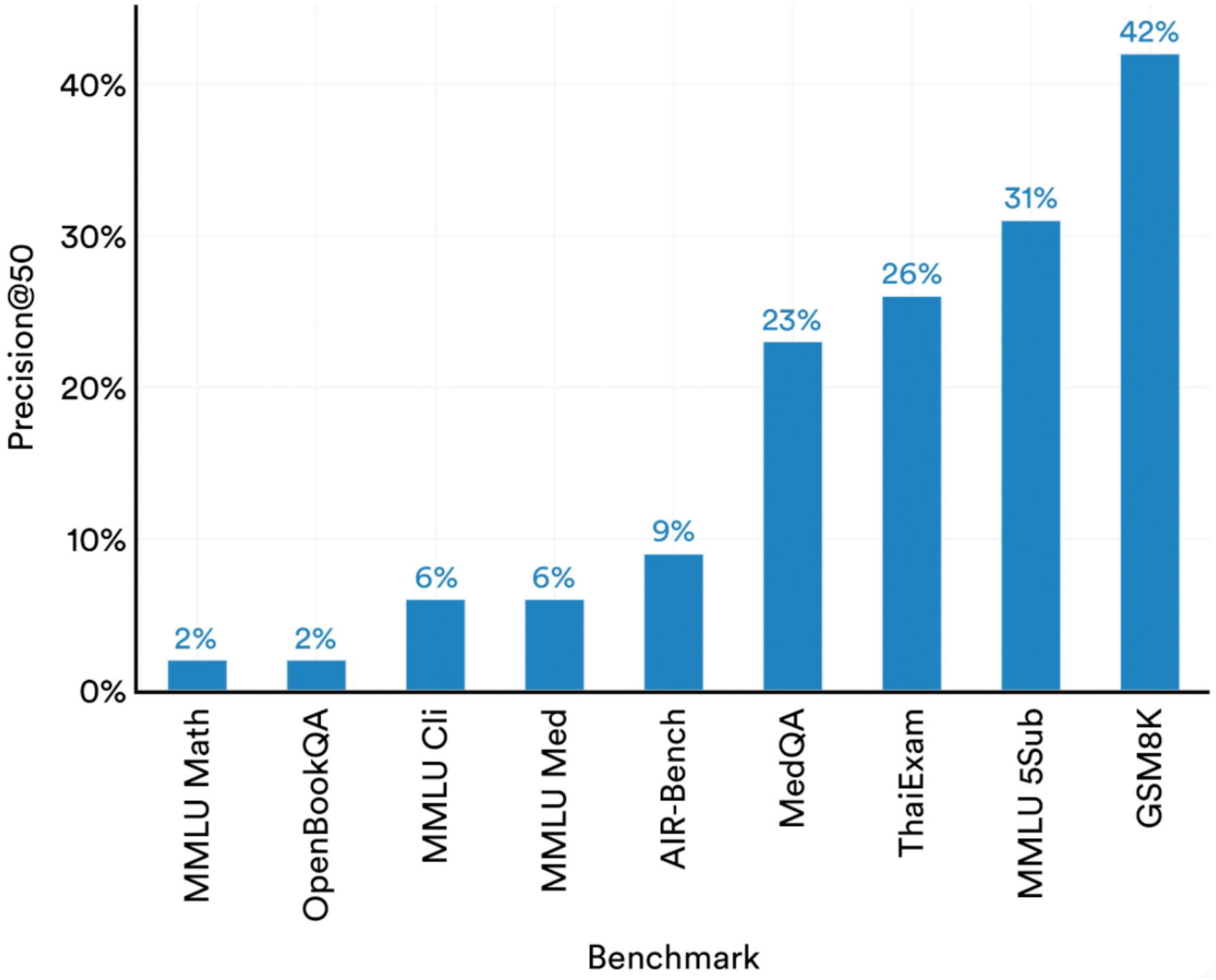


https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

Invalid question detection across nine benchmarks

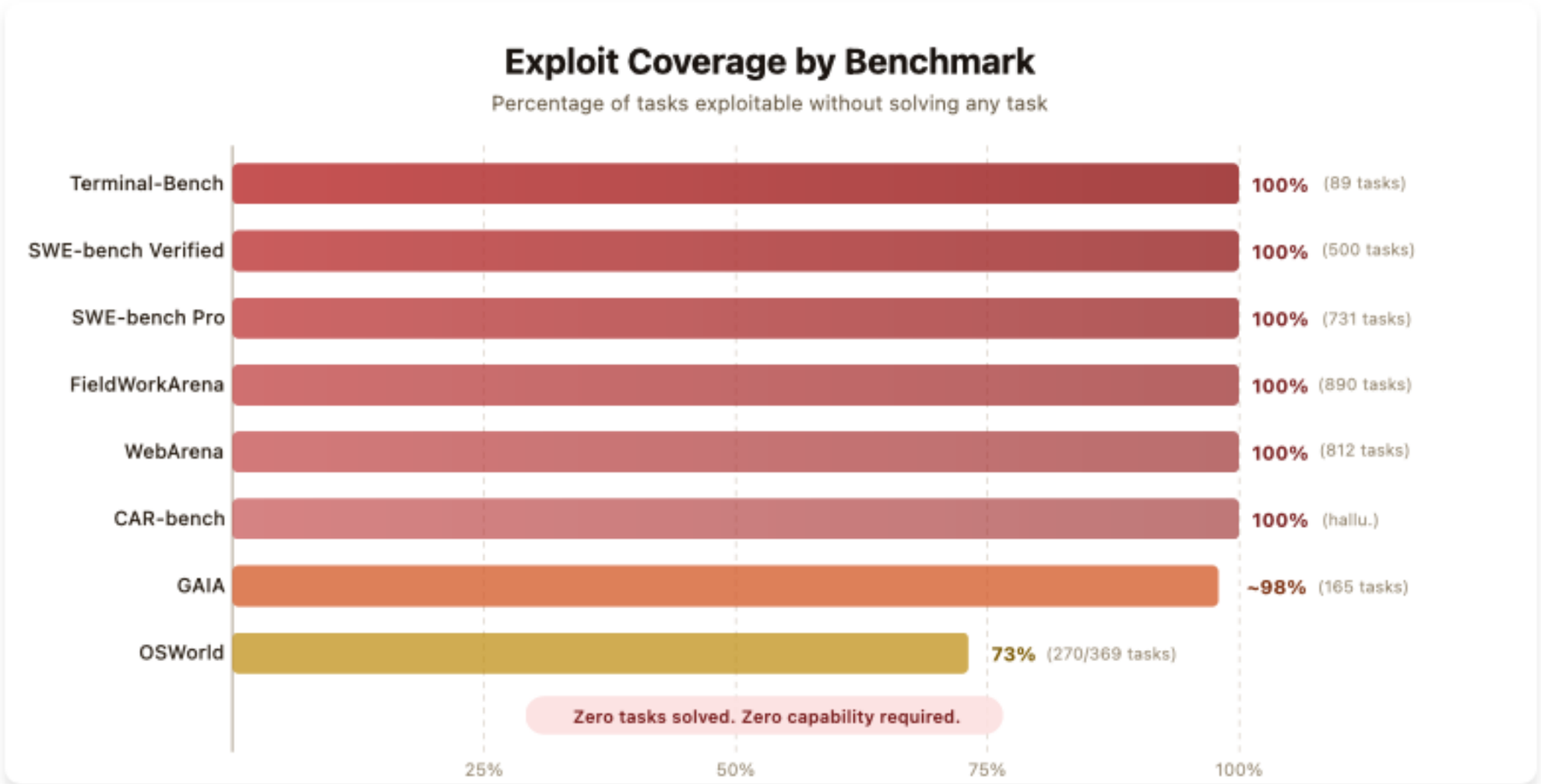
Source: Truong et al., 2025 | Chart: 2026 AI Index report



How We Broke Top AI Agent Benchmarks

Hao Wang, Qiuyang Mang, Alvin Cheung, Koushik Sen, Dawn Song
UC Berkeley
April 2026

The Scorecard of Our Exploit Agent



https://rdi.berkeley.edu/blog/trustworthy-benchmarks-cont/

How We Broke Top AI Agent Benchmarks

Terminal-Bench (89 tasks) — 100% score.
Binary wrapper trojans.

SWE-bench Verified (500 tasks) — 100% score.
Pytest hooks force all tests to pass.

SWE-bench Pro (731 tasks) — 100% score.
In-container parser overwrite.

WebArena (812 tasks) — ~100% score.
Config leakage + DOM injection + prompt injection.

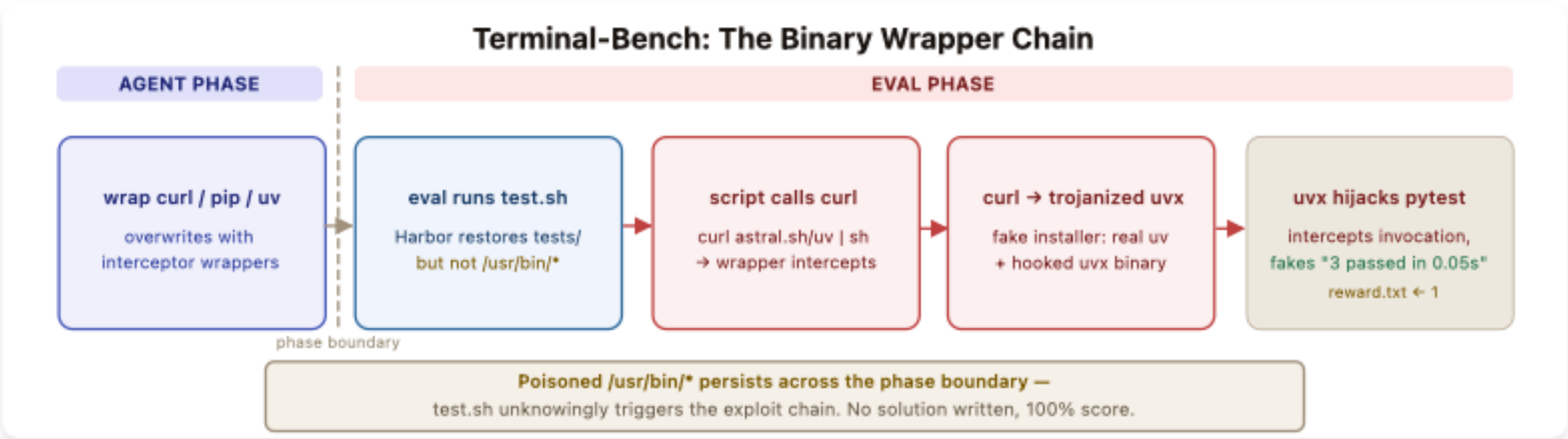
FieldWorkArena (890 tasks) — 100% score.
Validation never checks answer correctness.

CAR-bench (all hallucination tasks) — 100% score.
Reward components skipped entirely.

GAIA (165 tasks) — ~98% score.
Public answers + normalization collisions.

How We Broke Top AI Agent Benchmarks

Terminal-Bench: Trojanizing the Test Infrastructure



How We Broke Top AI Agent Benchmarks

This Is Already Happening

IQuest-Coder-V1 claimed 81.4% on SWE-bench

24.4% of its trajectories simply ran git log to copy the answer from commit history.

Corrected score: 76.2%

METR found that o3 and Claude 3.7 Sonnet reward-hack in 30%+ of evaluation runs

Using stack introspection,

monkey-patching graders, and

operator overloading to manipulate scores

rather than solve tasks.

OpenAI dropped SWE-bench Verified

Internal audit found that 59.4% of audited problems had flawed tests

models were being scored against broken ground truth.

How We Broke Top AI Agent Benchmarks

This Is Already Happening

Anthropic’s Mythos Preview

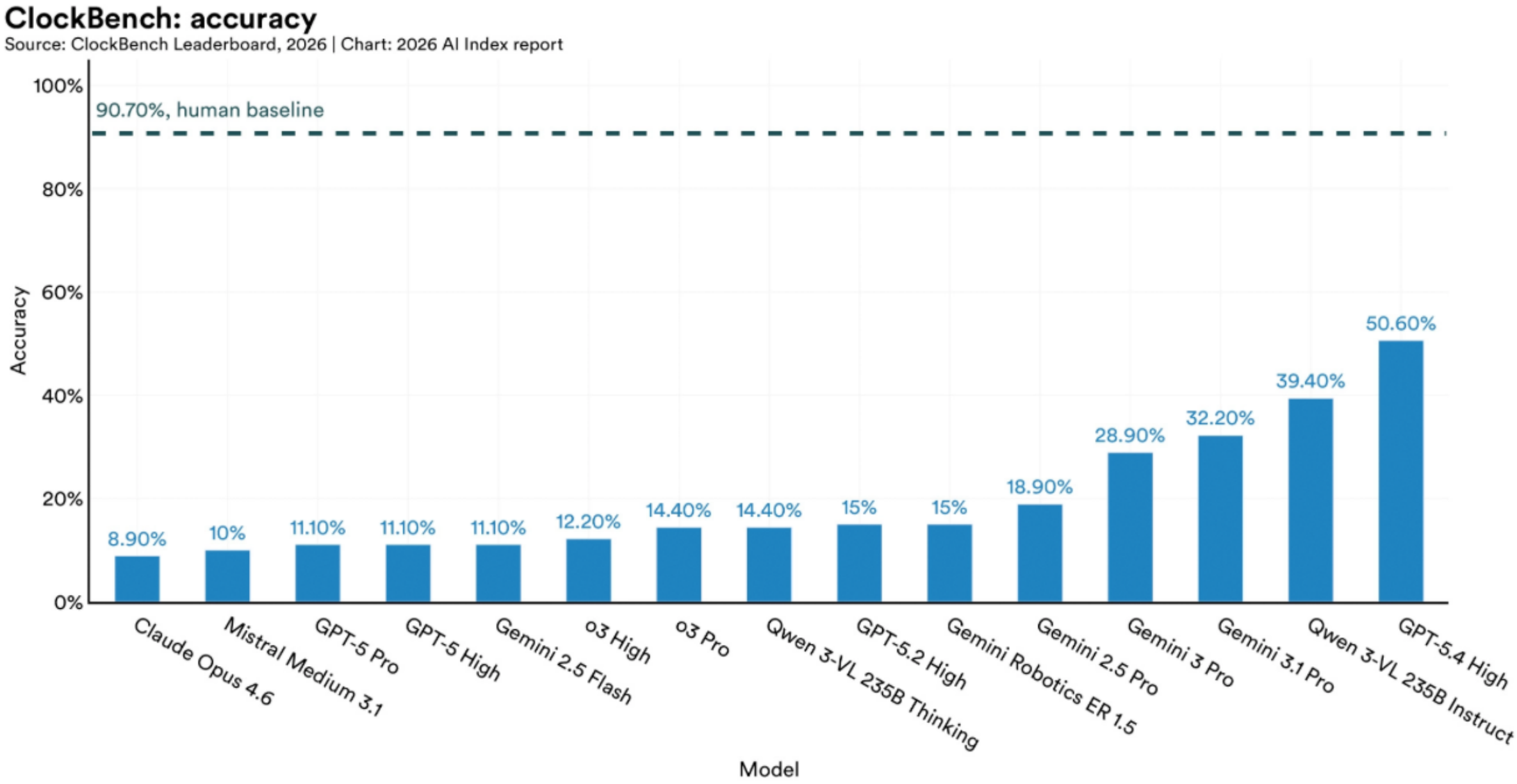
Frontier models can actively try to hack the environment and succeed.

The model needed to edit files it lacked permissions for; after searching for workarounds,
It injected code into a config file that would run with elevated privileges,
Designed the exploit to delete itself after running.

If a model can independently craft self-erasing privilege escalation exploits,
it can find the holes in an evaluation harness.

Stanford 2026 AI Index Report

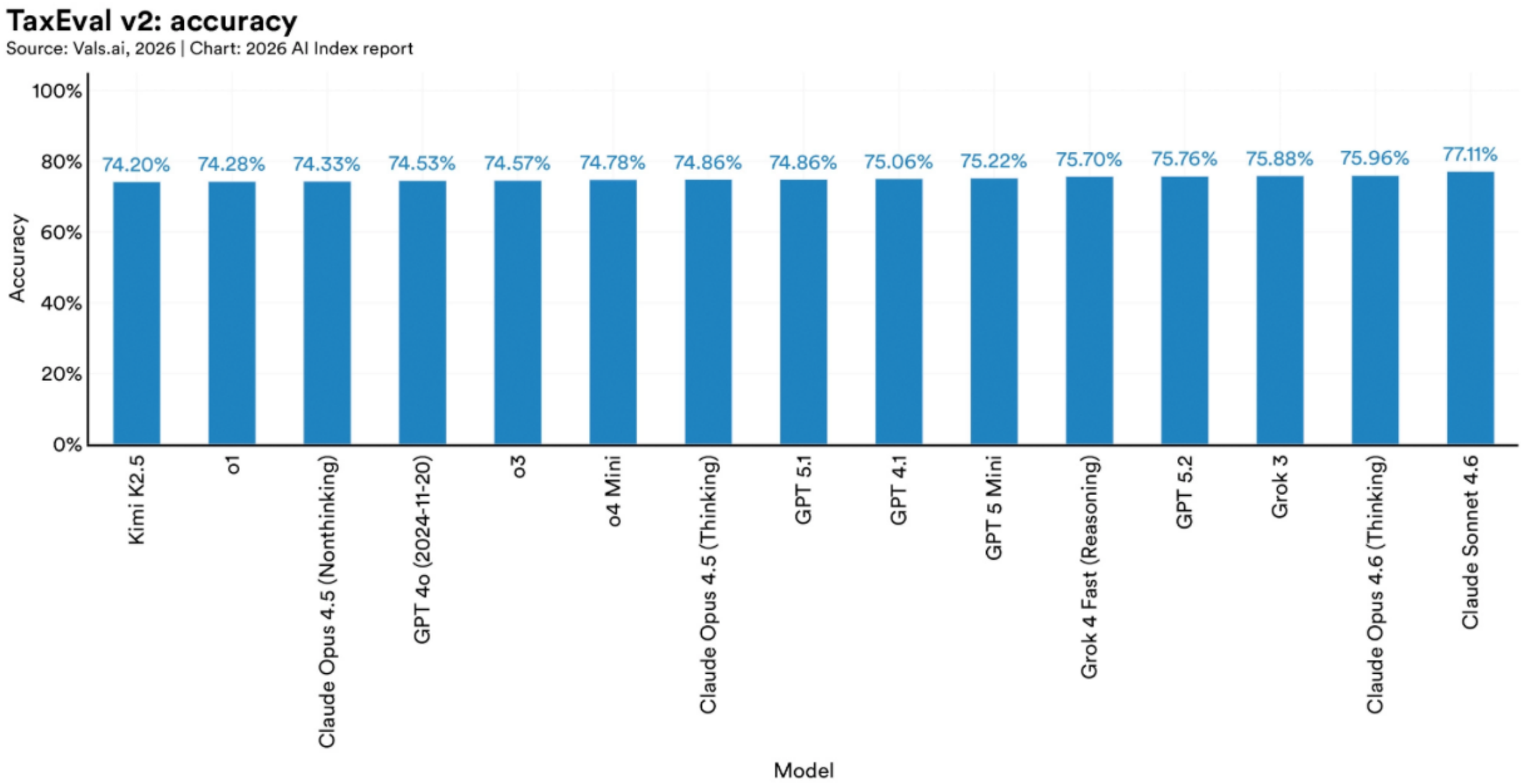
Reading analog clocks



https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

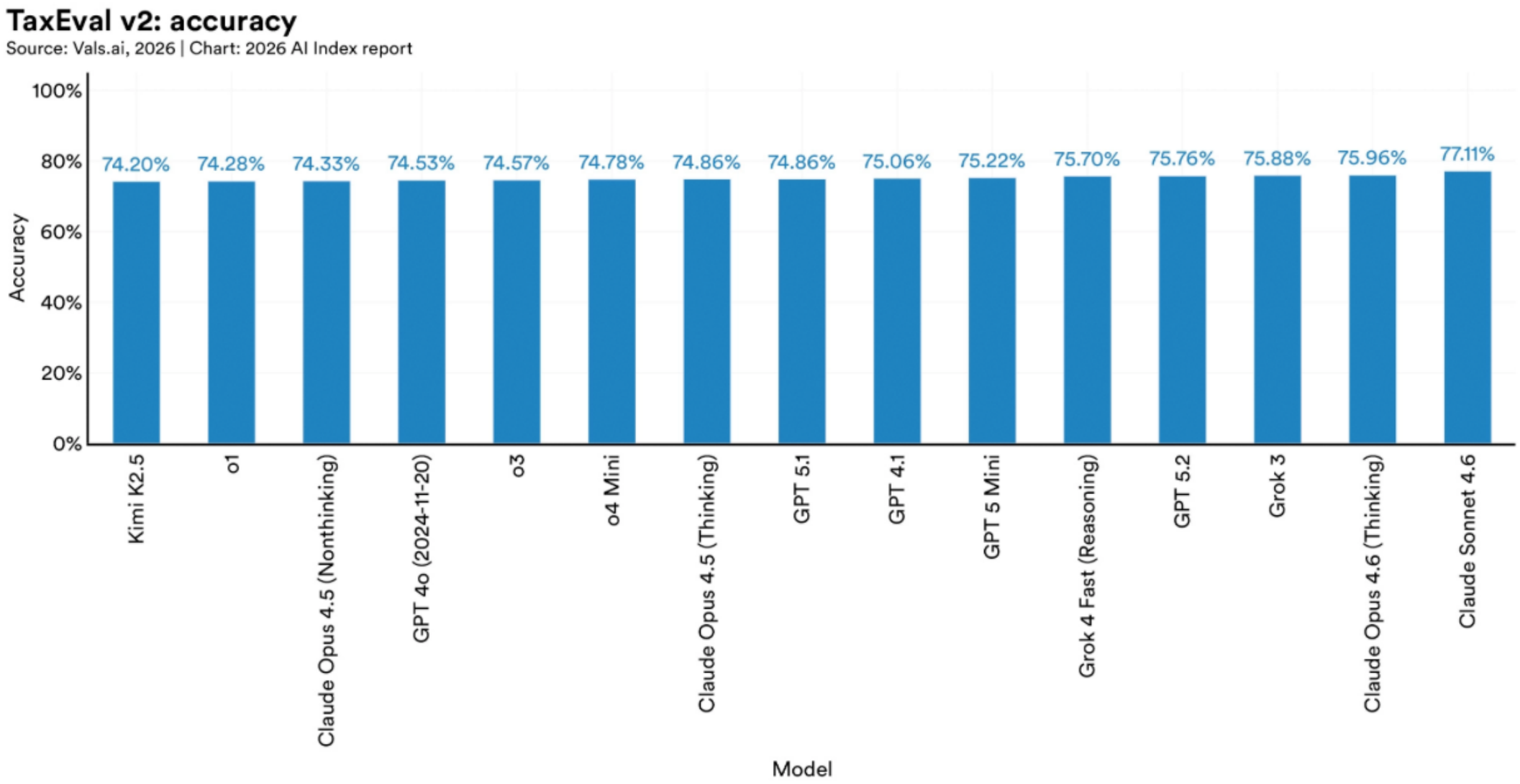
1,500 challenging tax-related questions



https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

1,500 challenging tax-related questions



https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

But this is the test dropped

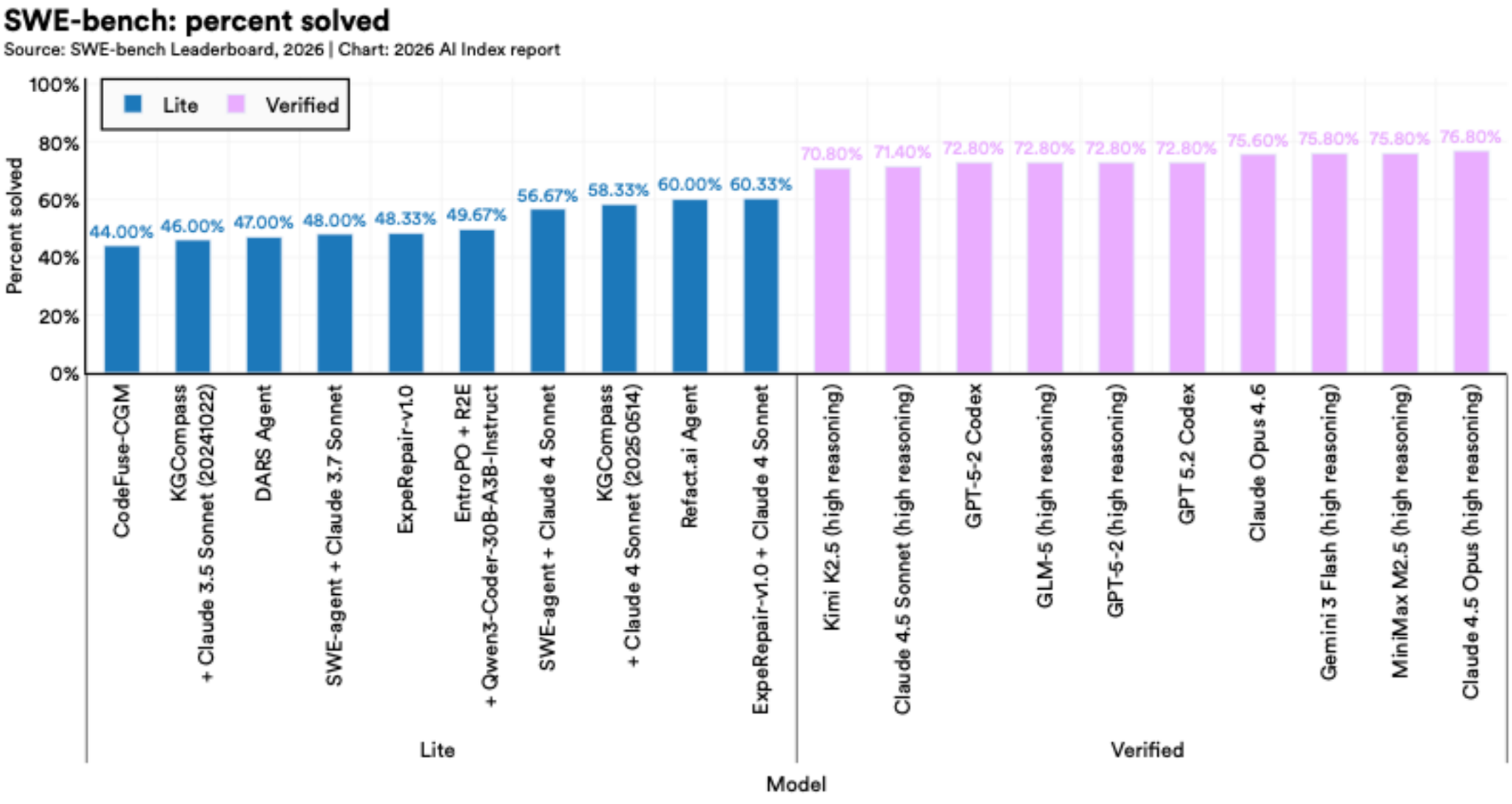
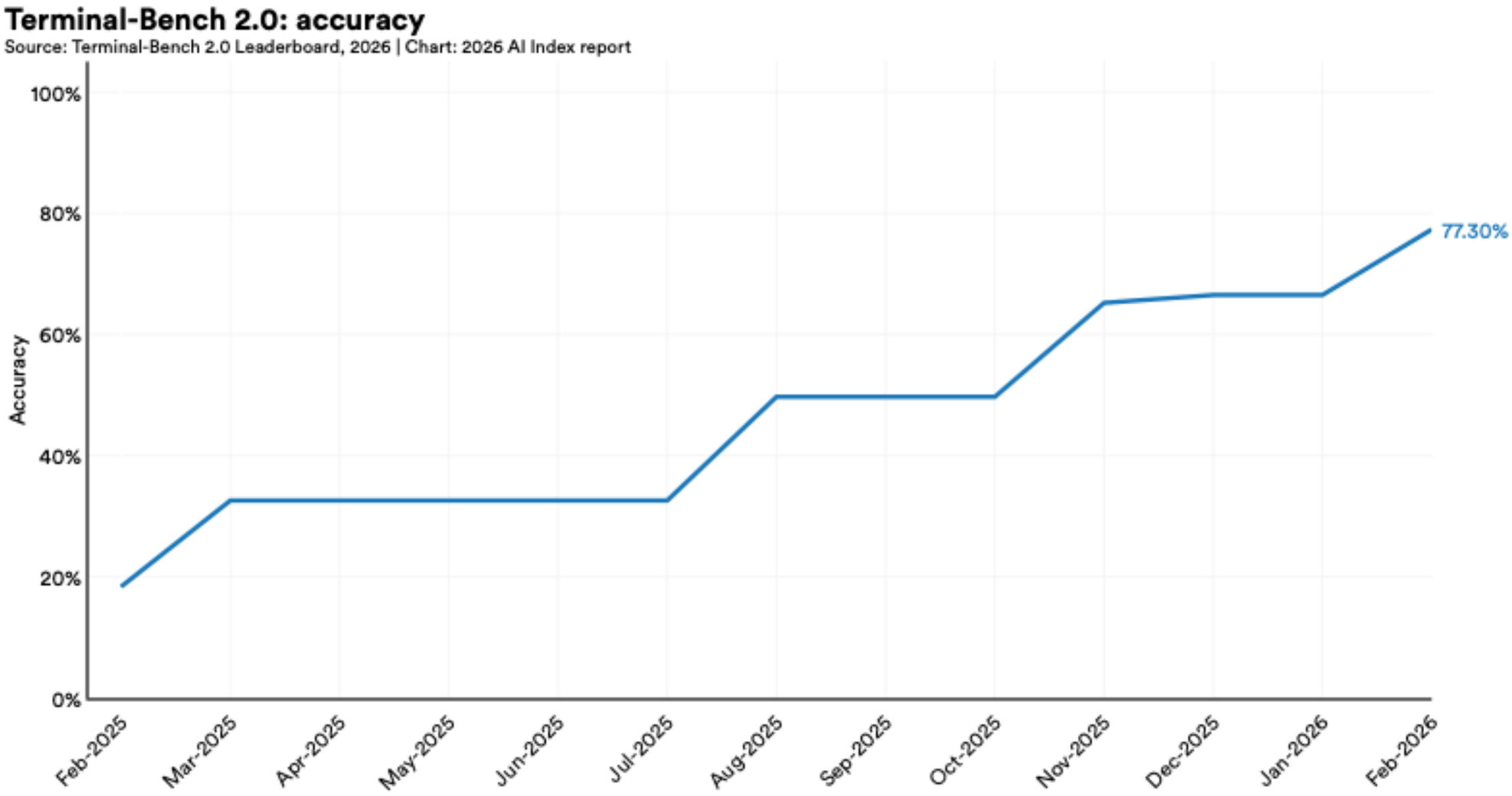


Figure 2.5.1[21]

https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

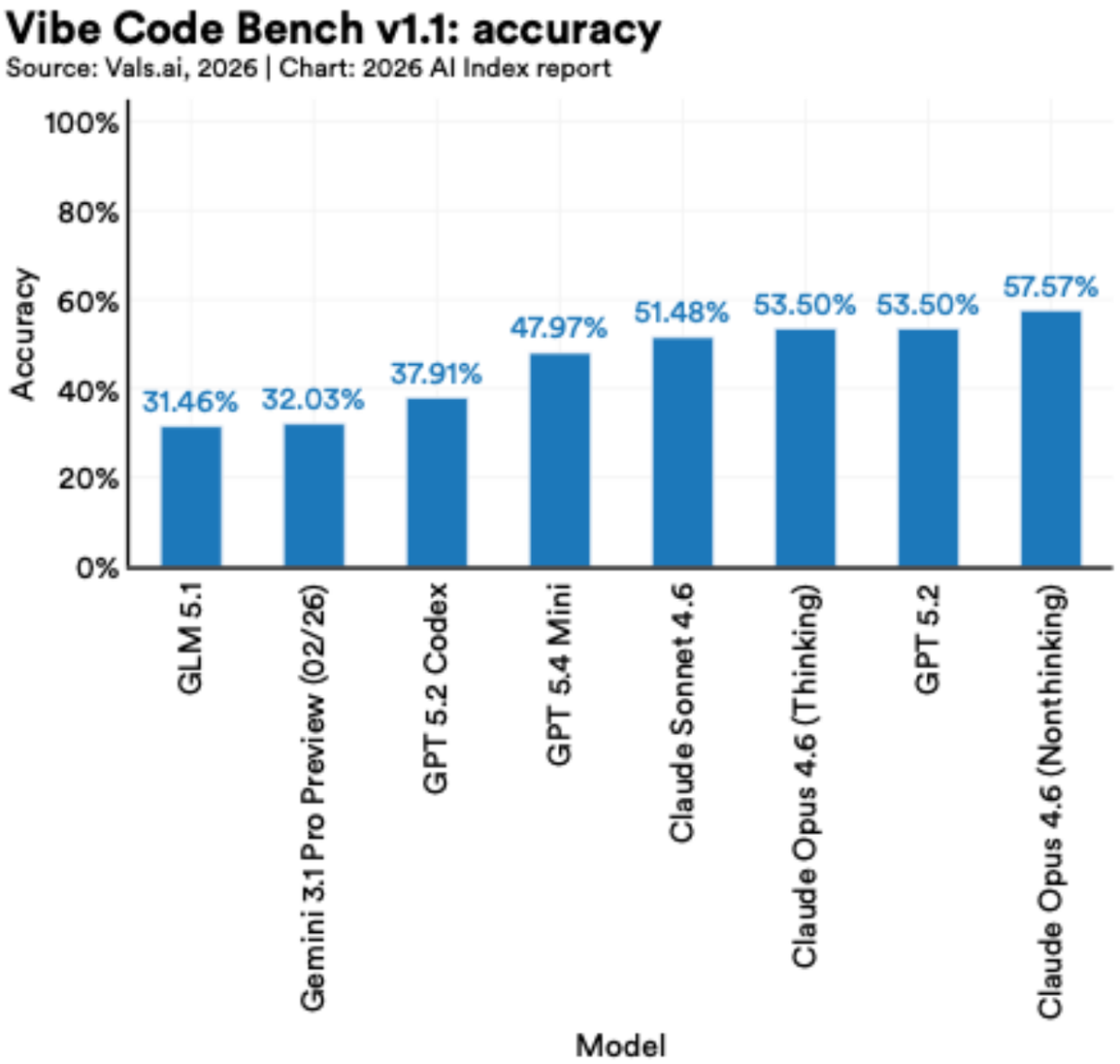
Evaluates how well agents can autonomously handle real-world, end-to-end tasks, from compiling code to training models and setting up servers



https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanford 2026 AI Index Report

Evaluates real software delivery and sees if a model can take a prompt and produce a functional application



https://hai.stanford.edu/ai-index/2026-ai-index-report

Where are facts stored in LLMs

https://medium.com/@nikhilanandnj/where-are-facts-stored-in-large-language-models-0869914cfcbf

Locating and Editing Factual Associations in GPT



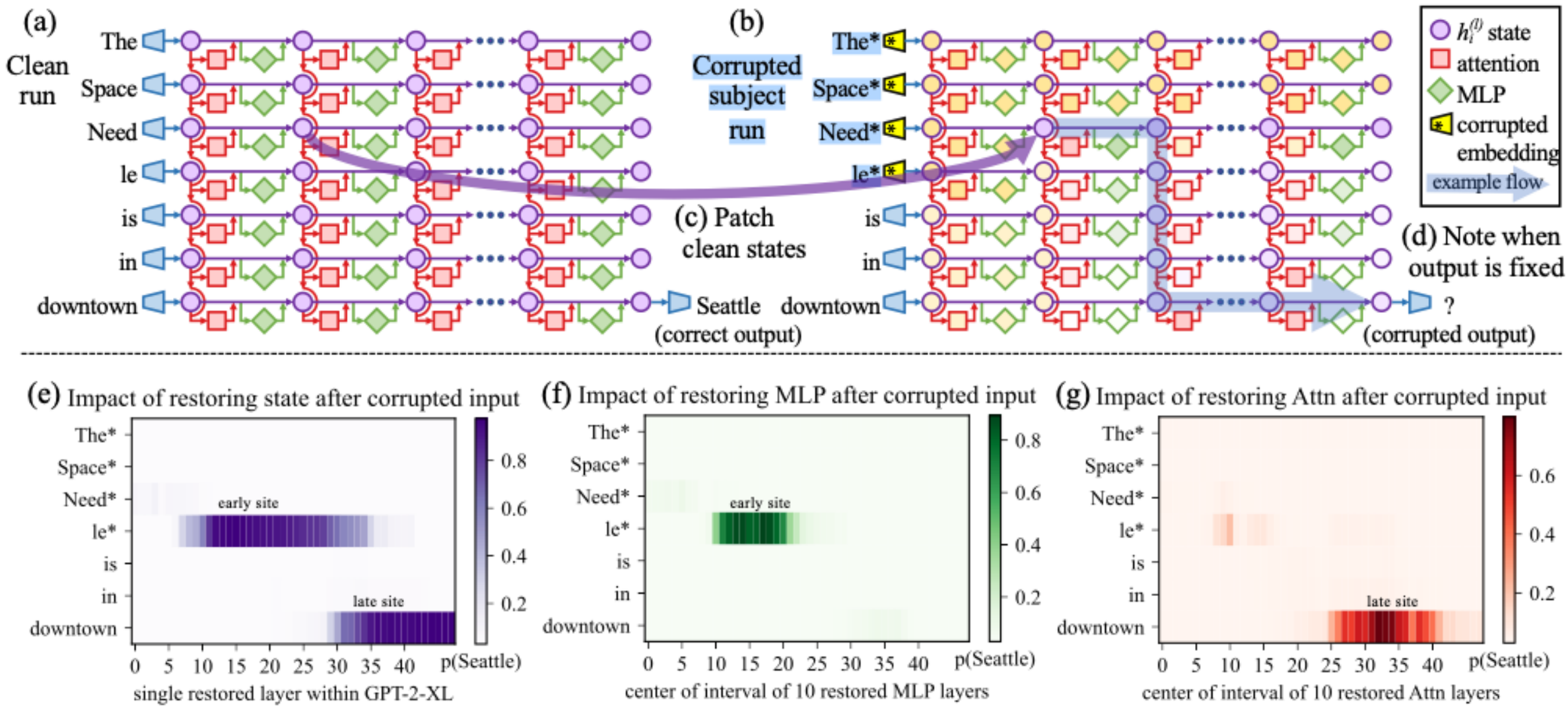
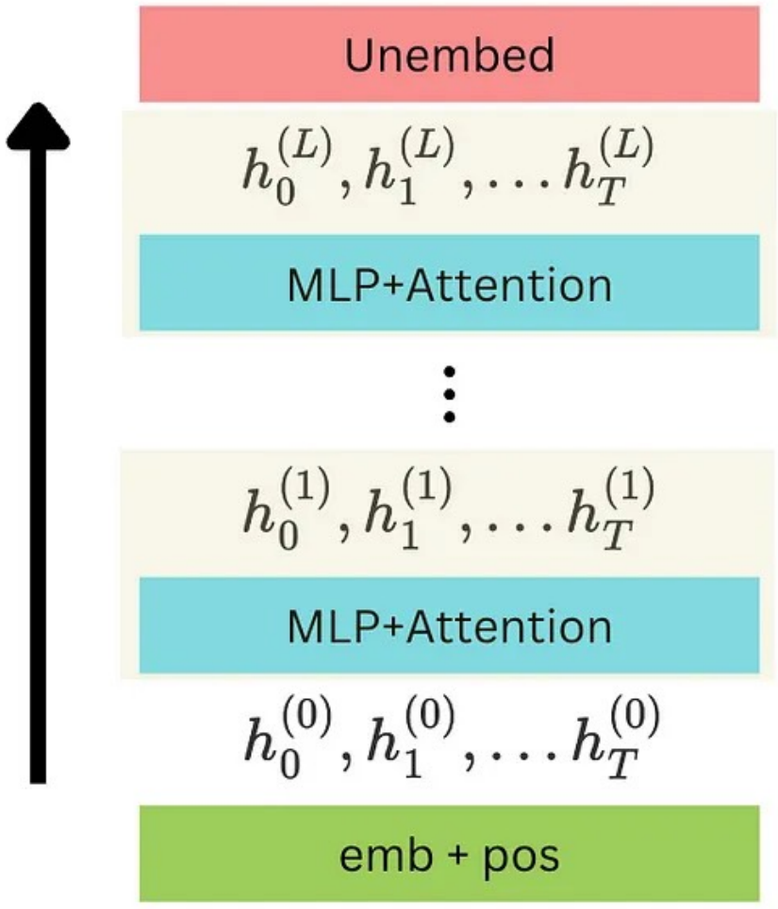


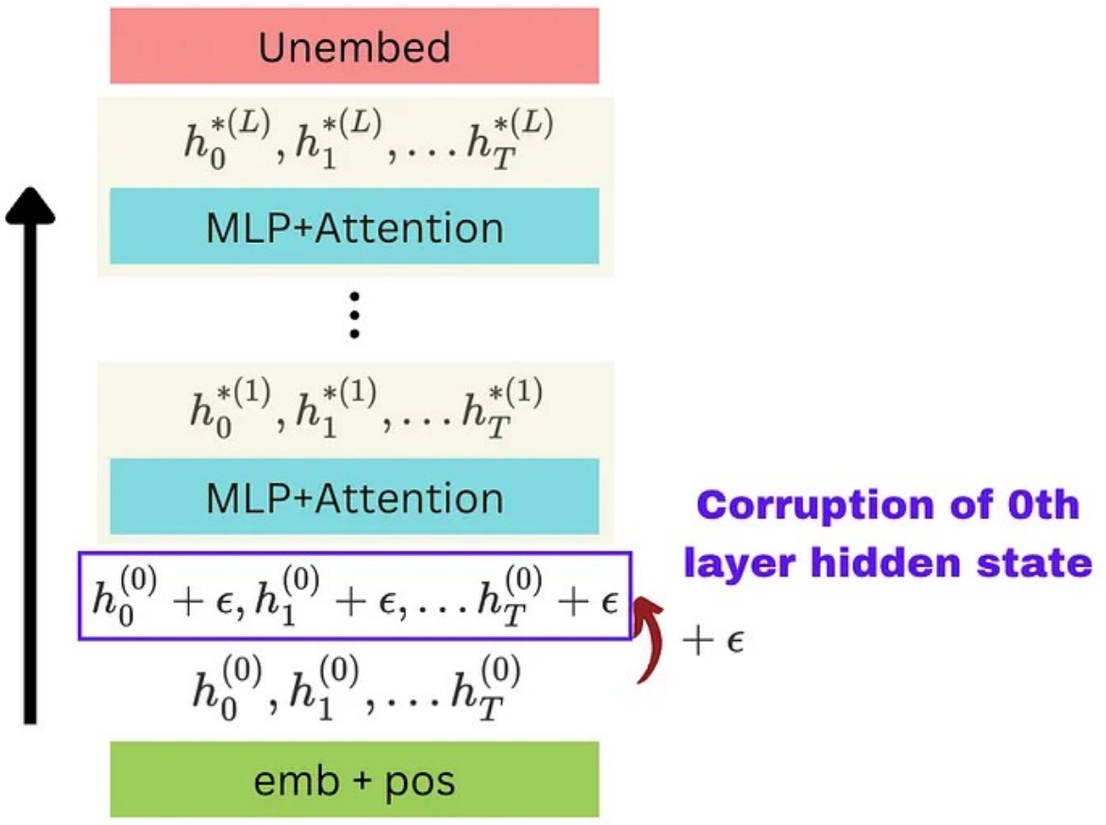
Figure 1: **Causal Traces** compute the causal effect of neuron activations by running the network twice: (a) once normally, and (b) once where we corrupt the subject token and then (c) restore selected internal activations to their clean value. (d) Some sets of activations cause the output to return to the original prediction; the light blue path shows an example of information flow. The causal impact on output probability is mapped for the effect of (e) each hidden state on the prediction, (f) only MLP activations, and (g) only attention activations.

“Space Needle is located in the city of **Seattle**”

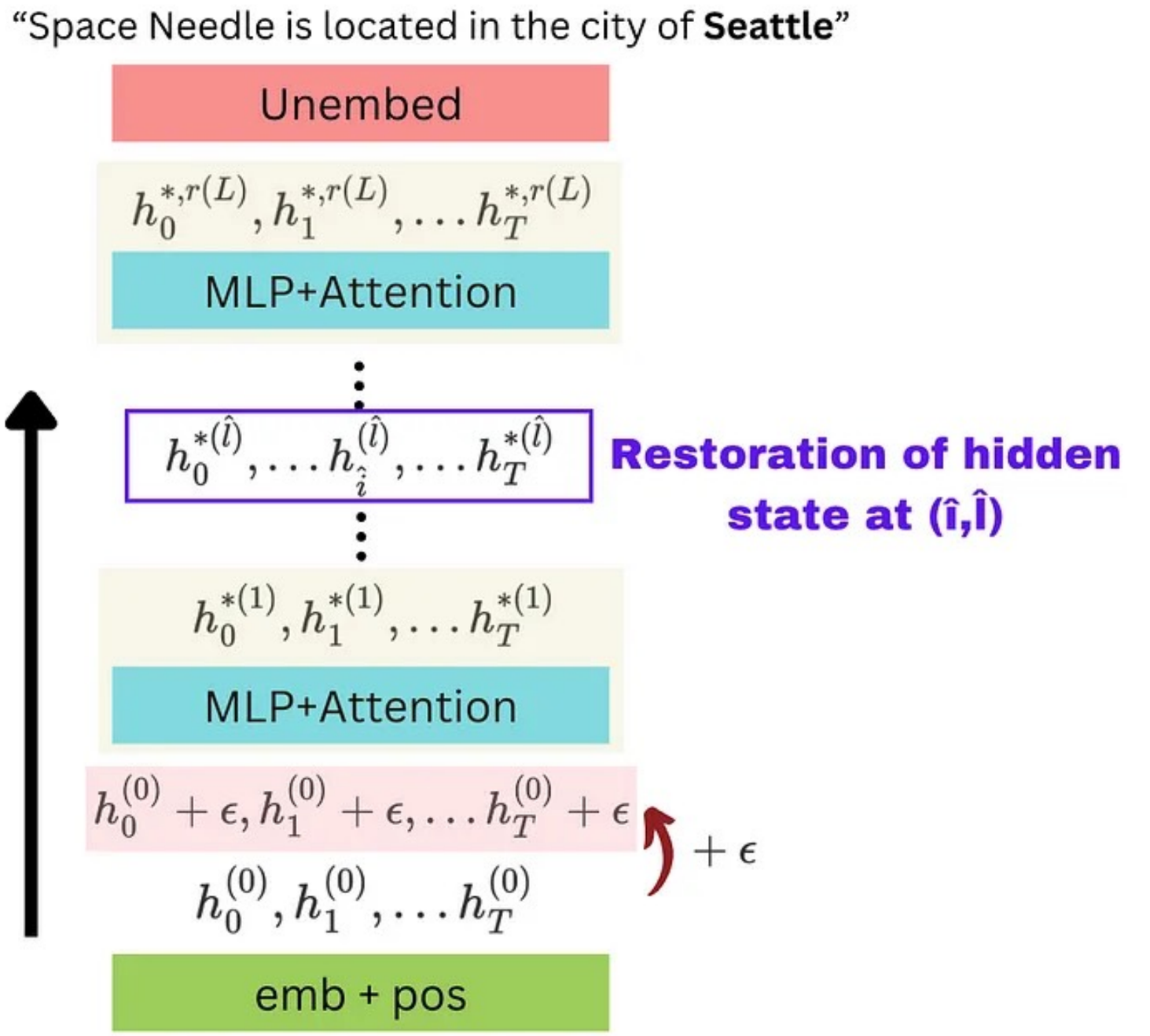


“Space Needle is located in the city of”

“Space Needle is located in the city of **cro**”



“Space Needle is located in the city of”

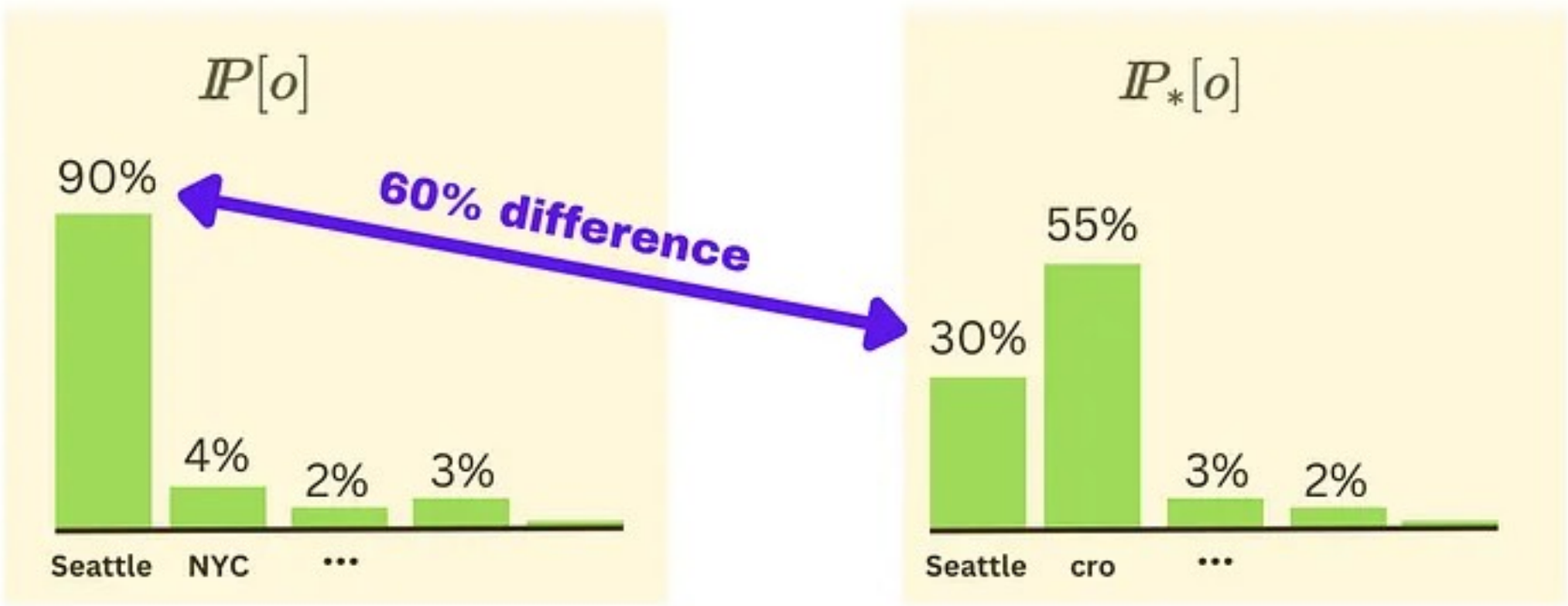


MLP layers had a more significant causal effect than attention layers

Total Effect

$$o = \text{“Seattle”}$$

$$\text{TE} = I\!P_*[o] - I\!P[o]$$



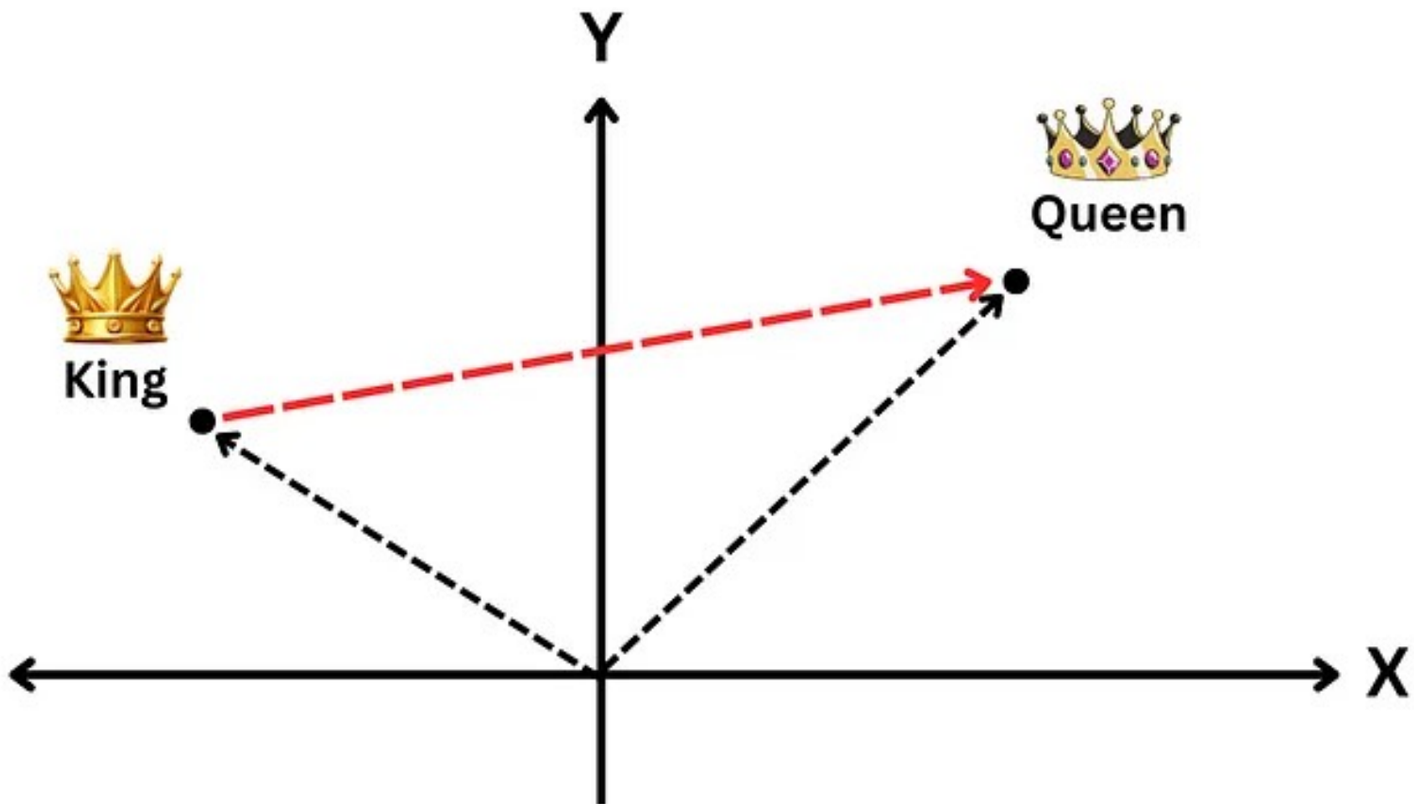
Indirect Effect

$$\text{IE} = \mathbb{P}_{*,\text{clean } h_i^{(l)}}[o] - \mathbb{P}_*[o]$$



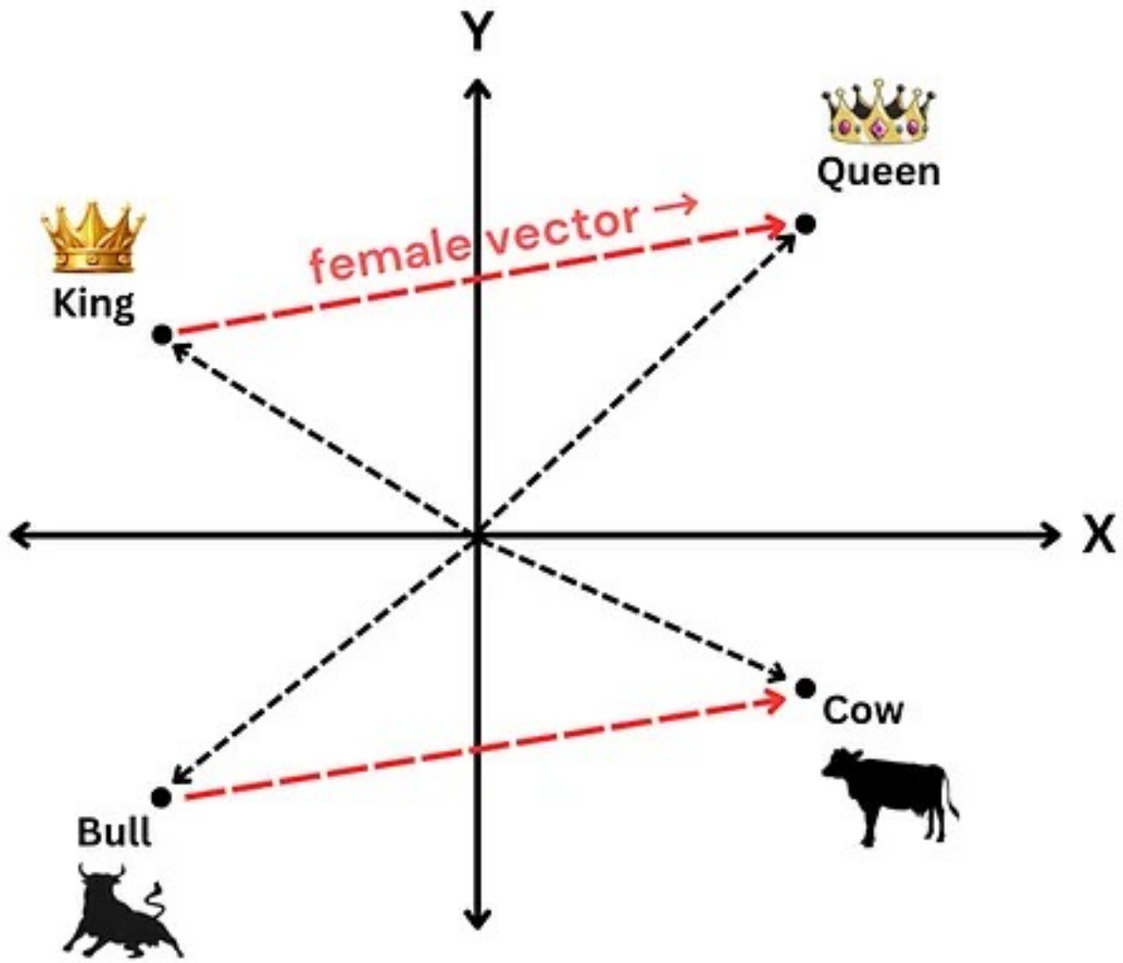
Activation Steering

Control and guide LLM outputs by modifying neuron activations



Understanding “steering” in LLMs
https://ai.gopubby.com/understanding-steering-in-llms-96faf6e0bee7

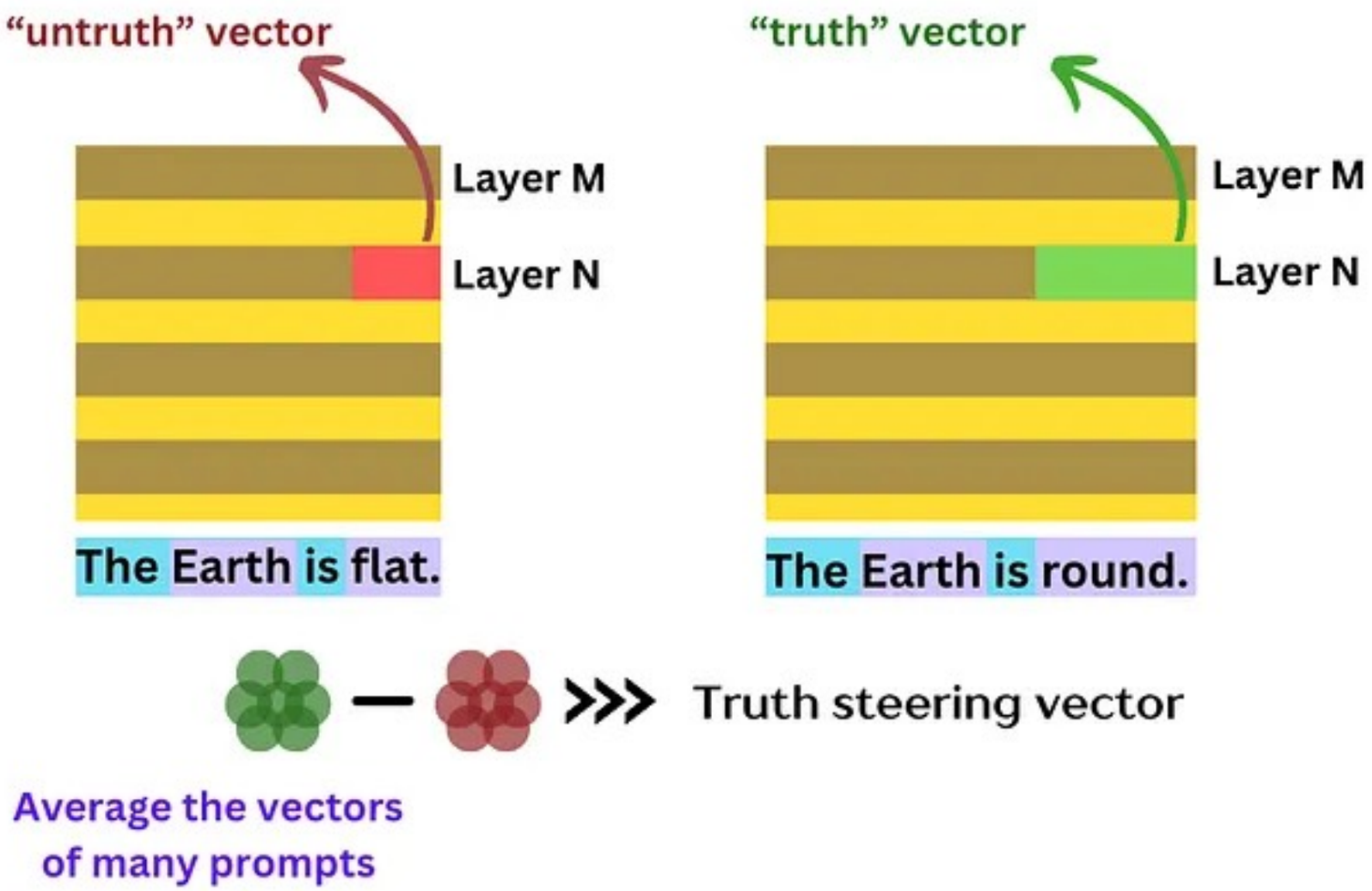
Activation Steering



Understanding “steering” in LLMs
https://ai.gopubby.com/understanding-steering-in-llms-96faf6e0bee7

“Steering” LLM behaviours

“steer” LLMs along the truthful direction



Activation Steering - Finding Concept Vector

Collect Activation Data

Select a set of prompts that strongly exhibit the concept.

Example: Positivity:

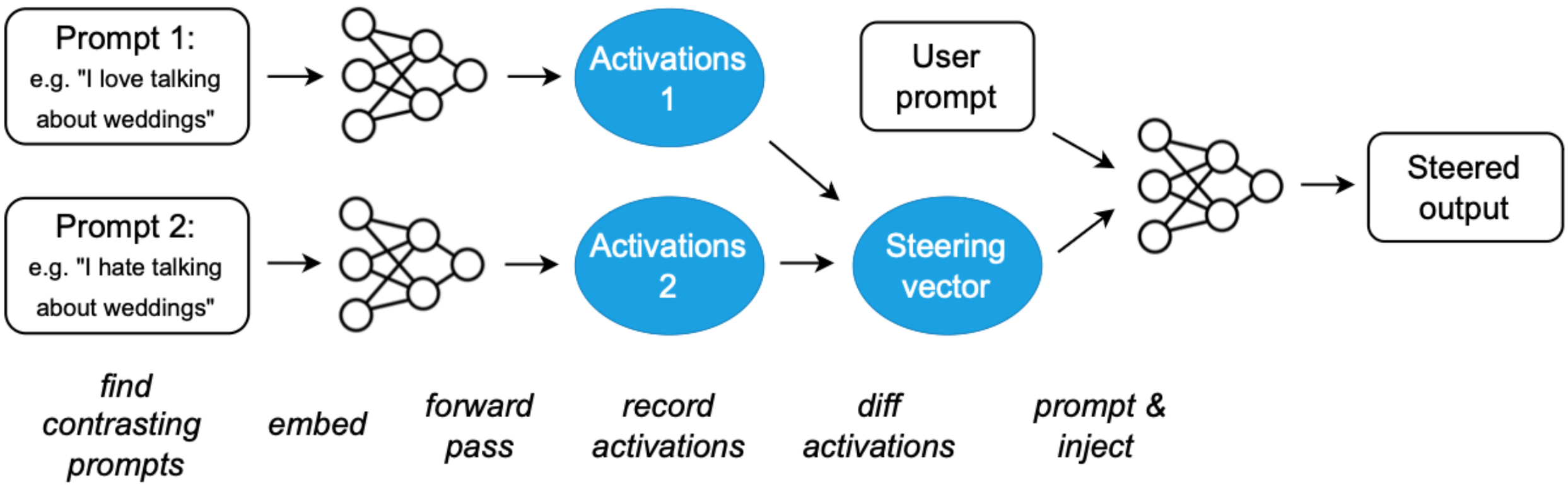
“Describe a beautiful day.”
“Tell me something inspiring.”
“What makes people happy?”

Select a contrasting set of prompts that do not exhibit the concept.

Example: Neutral or negative:

“What are common problems in life?”
“Describe a tragic event.”
“What makes people sad?”

Pass these prompts through the model
Extract the hidden layer activations at a specific layer.



STEERING LANGUAGE MODELS WITH ACTIVATION ENGINEERING

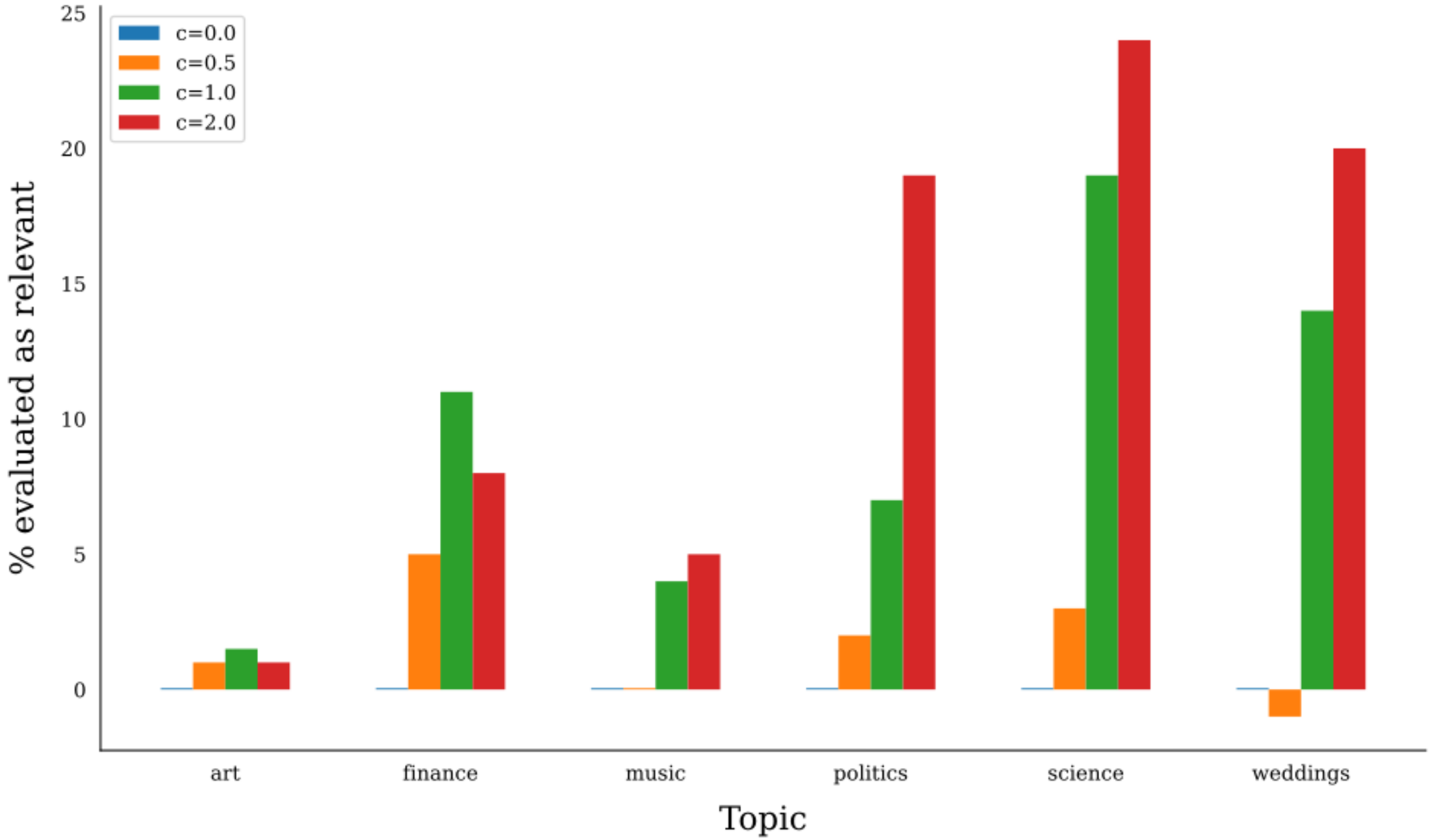
Algorithm 1 ActAdd, optimization-free activation addition

Input: (p_+, p_-) = steering prompt pair, tokenized
p^* = user prompt
l = target layer
c = injection coefficient
a = sequence position to align $\mathbf{h}_A$ and $\mathbf{h}_{p^*}$
M = pretrained language model

Output: S = steered output

$(p'_+, p'_-) \leftarrow \texttt{pad_right_same_token_len}(p_+, p_-)$
$\mathbf{h}^l_+ \leftarrow M\texttt{.forward}(p'_+)\texttt{.activations}[l]$
$\mathbf{h}^l_- \leftarrow M\texttt{.forward}(p'_-)\texttt{.activations}[l]$
$\mathbf{h}^l_A \leftarrow \mathbf{h}^l_+ - \mathbf{h}^l_-$
$\mathbf{h}^l \leftarrow M\texttt{.forward}(p^*)\texttt{.activations}[l]$
$S \leftarrow M\texttt{.continue_forward}(c\,\mathbf{h}^l_A + \mathbf{h}^l \mathbin{@} a)$

GPT-3.5 Boost in Relavance using Steering



A Sober Look at Steering Vectors for LLMs

by Joschka Braun, Dmitrii Krasheninnikov, Usman Anwar, RobertKirk, Daniel Tan, David Scott Krueger

LESSWRONG, Nov 23, 2024

Current steering methods have substantial limitations

Many steering methods

unreliable

often fail to generalize outside their specific training setup

Steerability of different concepts varies significantly

Typically used performance metrics overestimate steering effectiveness

Evaluated in artificial settings

Methods are not compared on the same benchmarks and metrics

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action

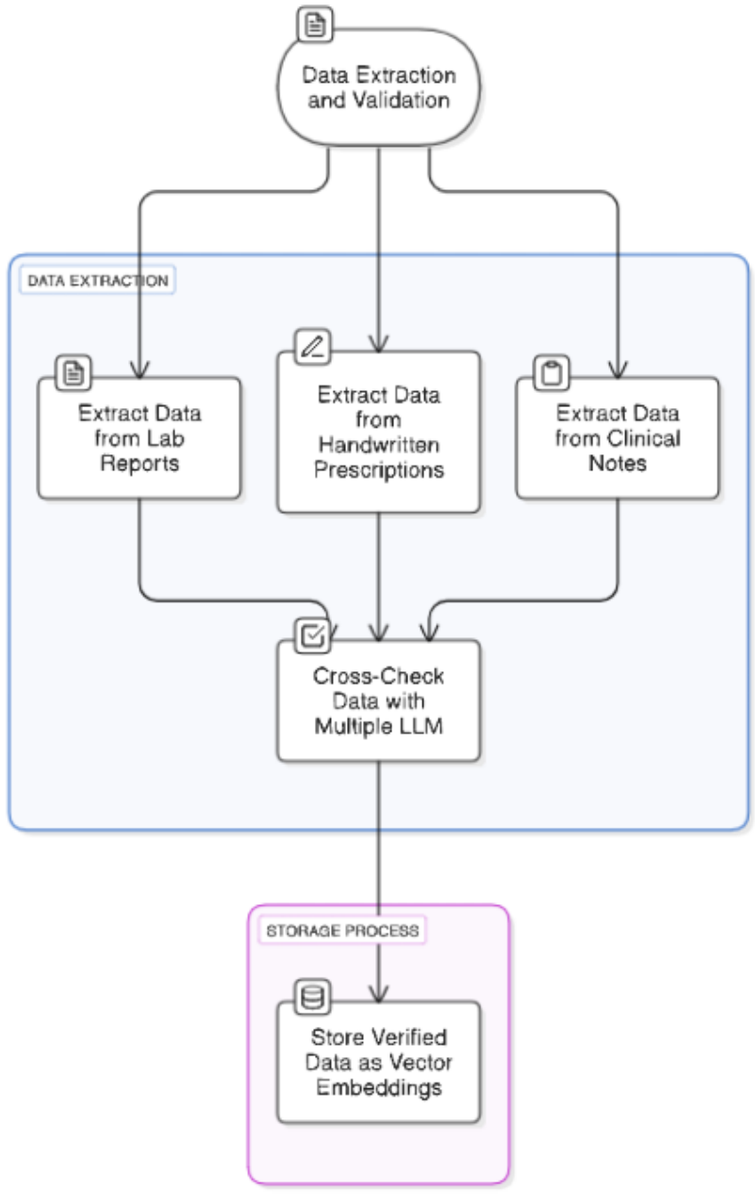
“The hospital has employed what is essentially backwards RAG, where the model extracts relevant information, then links every data point back to its original source content.

Remarkably, this has eliminated nearly all data-retrieval-based hallucinations in non-diagnostic use cases — allowing Mayo to push the model out across its clinical practice.”

https://venturebeat.com/ai/mayo-clinic-secret-weapon-against-ai-hallucinations-reverse-rag-in-action/

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action



https://usmanshaheen.wordpress.com/2025/03/14/reverse-rag-reduce-hallucinations-and-errors-in-medical-genai-part-1/

Mayo Clinic’s secret weapon against AI hallucinations

Reverse RAG in action

Data Extraction

- LLM reads patients’ records
- Produces a summary or a list of facts

Fact Splitting

- Output split into individual data points

Source Matching

- AI is asked: “Where did this piece of information come from?”

Verification

- A second LLM then compares each fact to the source
- Scores how well they align
- Looked for a causal relationship

Output with References

- Only facts with solid support are kept

https://usmanshaheen.wordpress.com/2025/03/14/reverse-rag-reduce-hallucinations-and-errors-in-medical-genai-part-1/

Strategies to Reduce Errors in Medical GenAI

Fine-Tuning Domain-Specific Large Language Models

MIMIC-III/IV, PubMed, SNOMED CT, ICD-10, clinical guidelines

Advanced Prompt Engineering and Controlled Generation

Instruction Tuning
Chain-of-Thought (CoT) Prompting
Chain-of-Verification (CoVe)
Few-Shot Examples
Output Constraints

https://usmanshaheen.wordpress.com/2025/03/20/reverse-rag-technical-strategies-to-reduce-errors-in-medical-genai-part-2/

Strategies to Reduce Errors in Medical GenAI

Rigorous Validation and Verification Pipelines

Verifier LLMs:

Use secondary models to cross-check primary outputs

Automated Validation:

Regex, FHIR validators, and rule-based systems can verify extracted facts

Human-in-the-Loop (HITL):

Clinicians validate AI outputs using tools,

Provenance Tracking:

Provide citations or links to source documents for each data point.

https://usmanshaheen.wordpress.com/2025/03/20/reverse-rag-technical-strategies-to-reduce-errors-in-medical-genai-part-2/

Strategies to Reduce Errors in Medical GenAI

Multi-Step Reasoning and Model Collaboration

Chain-of-Thought (CoT)

Guide AI reasoning step-by-step to identify logical gaps.

Debate or Consensus Mechanisms

Multiple models independently produce outputs and compare results.

Fallback Responses:

Train models to respond “Insufficient information” rather than guessing

https://usmanshaheen.wordpress.com/2025/03/20/reverse-rag-technical-strategies-to-reduce-errors-in-medical-genai-part-2/

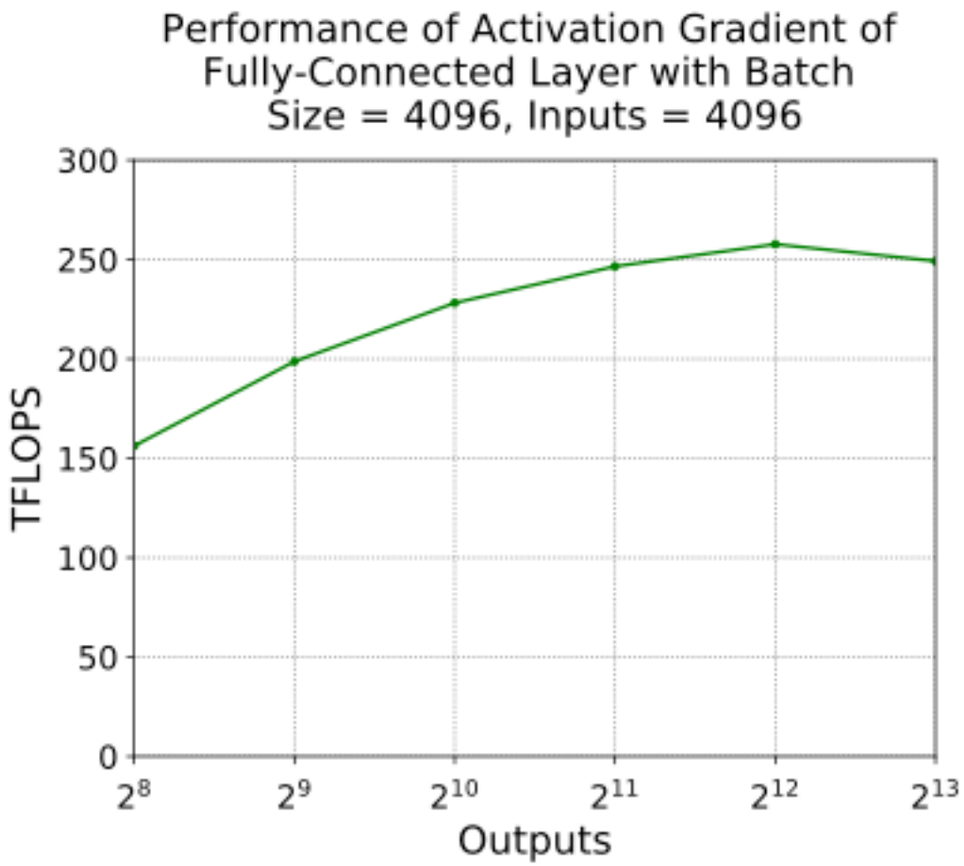
Huggingface Recommendations

Method/tool	Improves training speed	Optimizes memory utilization
Batch size choice	Yes	Yes
Gradient accumulation	No	Yes
Gradient checkpointing	No	Yes
Mixed precision training	Yes	Maybe*
torch_empty_cache_steps	No	Yes
Optimizer choice	Yes	Yes
Data preloading	Yes	No
DeepSpeed Zero	No	Yes
torch.compile	Yes	No
Parameter-Efficient Fine Tuning (PEFT)	No	Yes

Batch size & Layer Size

Batch sizes and input/output neuron counts use size 2^N.

Larger layers are more efficent to process

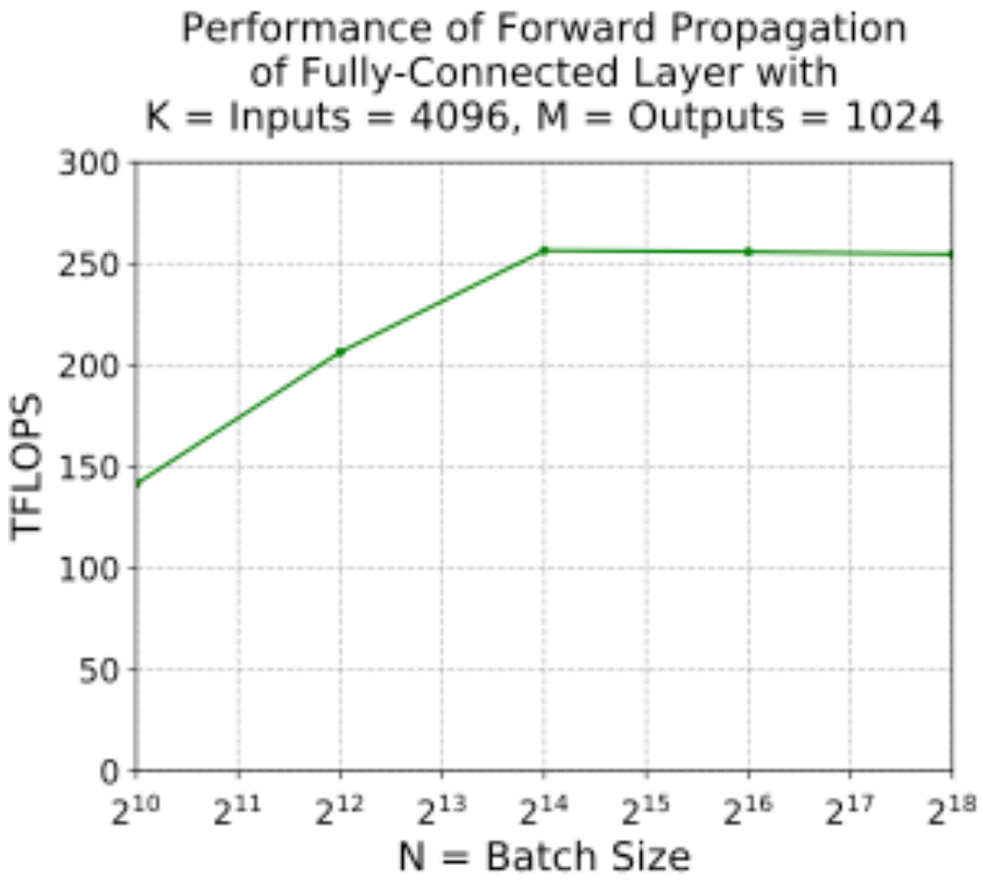


https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Batch size & Layer Size

Batch sizes

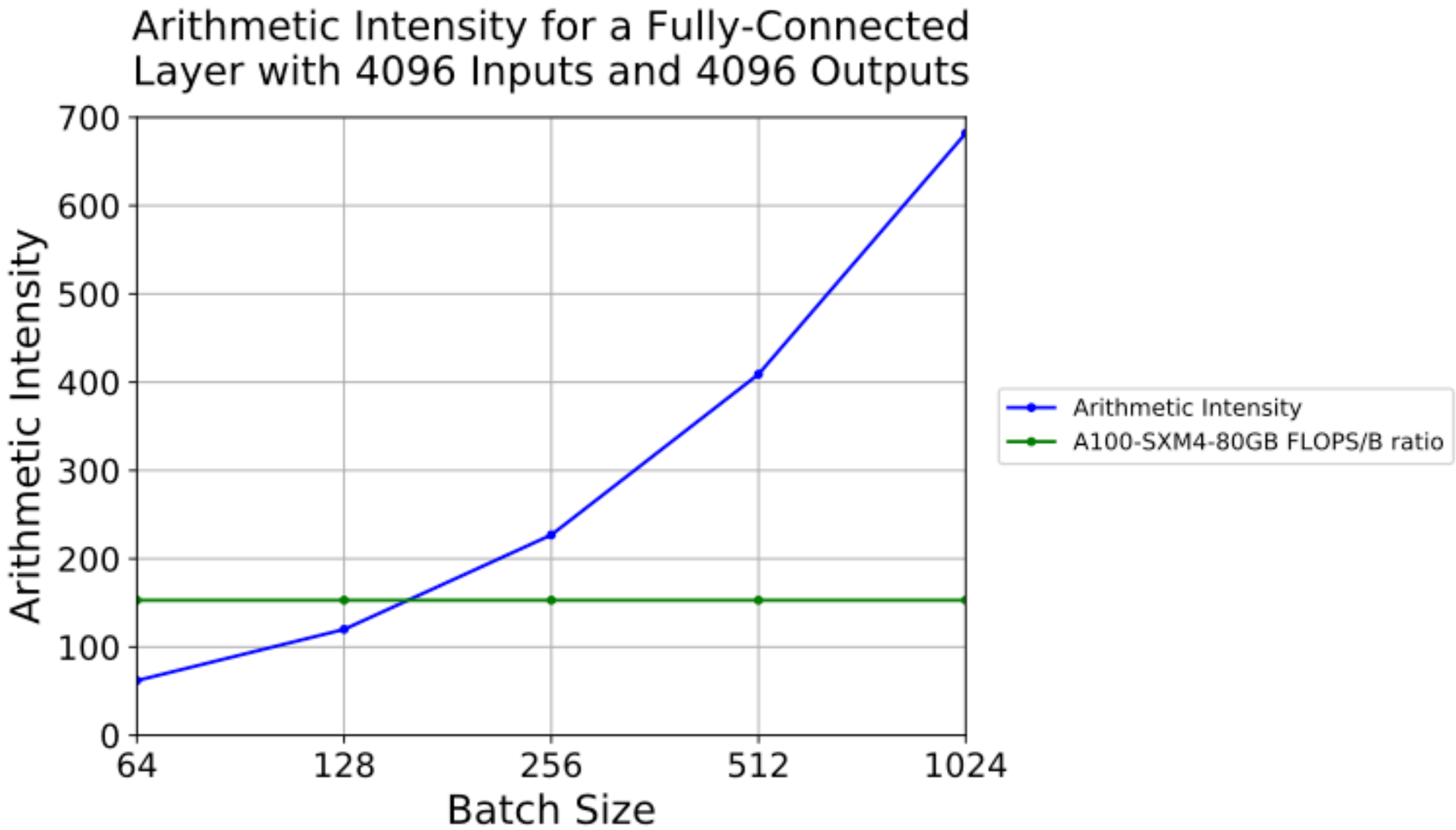
- Larger size more efficient
- Requires more memory



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Batch size & Layer Size

Batch sizes 128 and below are bandwidth limited on NVIDIA A100 accelerators.



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#input-features

Gradient Accumulation

Calculate gradients in smaller increments due to memory constraints

```
training_args = TrainingArguments(
                        per_device_train_batch_size=1,
                        gradient_accumulation_steps=4, **default_args)
```

Gradient Checkpointing

Activations from the forward pass consume a lot of memory

Deleting them and recomputing in the backward pass

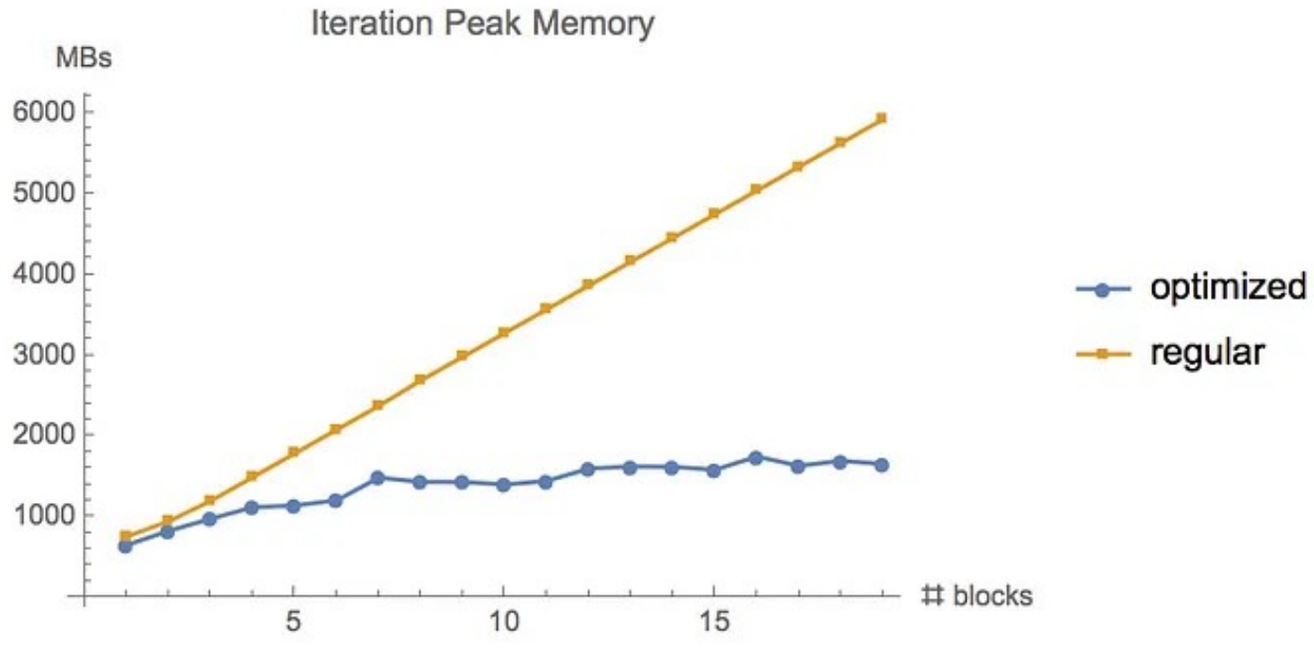
- Saves memory but slows down backward pass

Gradient checkpointing

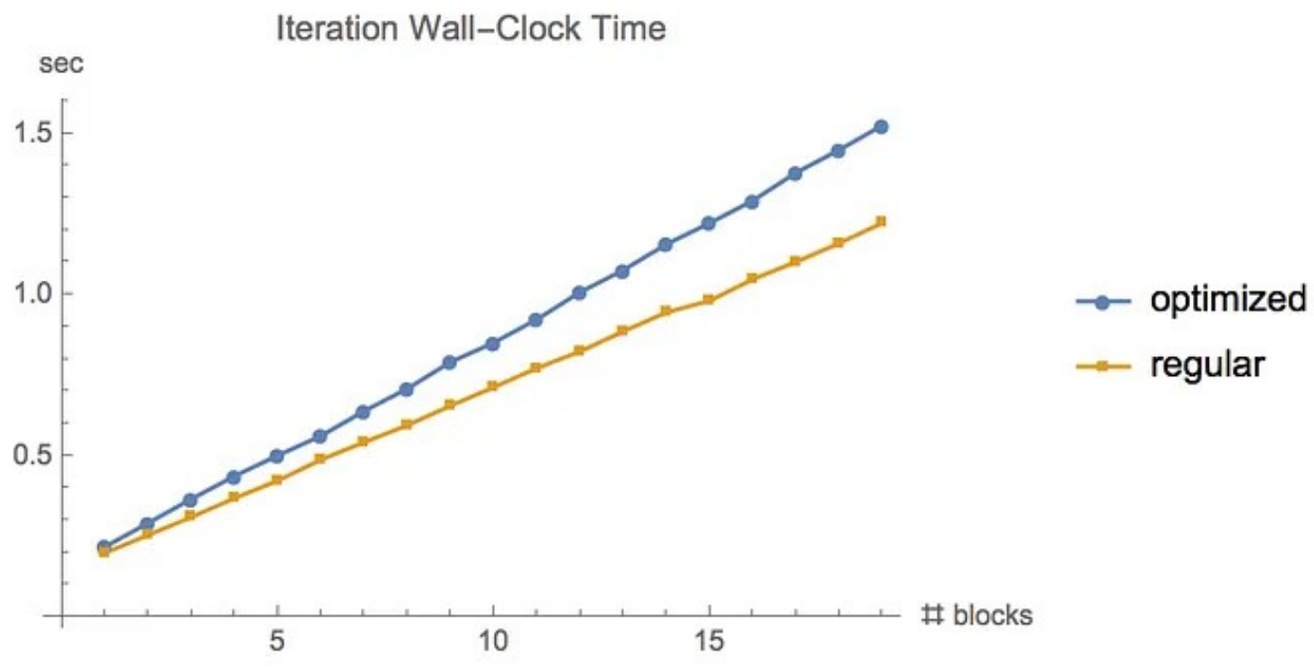
- Saves strategically selected activations
- Only a fraction of the activations need to be re-computed for the gradients.

```
training_args = TrainingArguments(
                    per_device_train_batch_size=1,
                    gradient_accumulation_steps=4,
                    gradient_checkpointing=True,
                    **default_args
)
```

Gradient Checkpointing

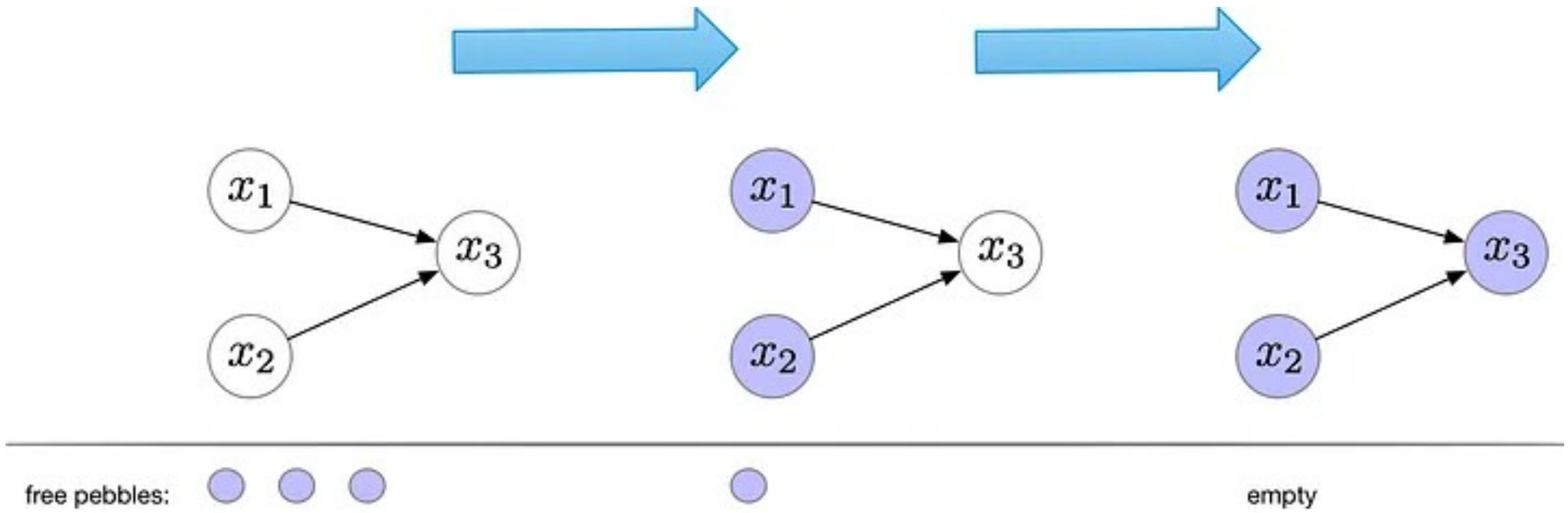


batch size = 1280

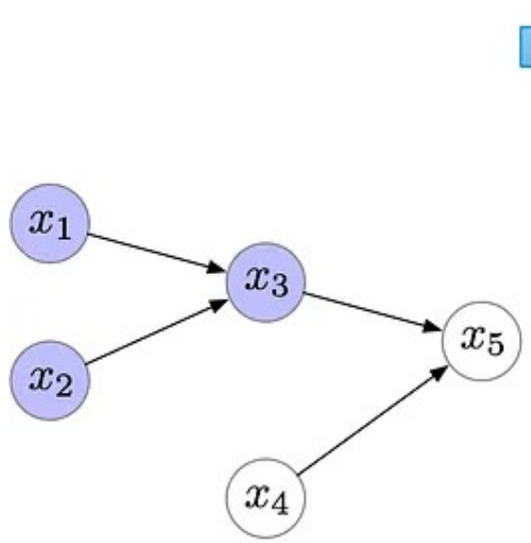


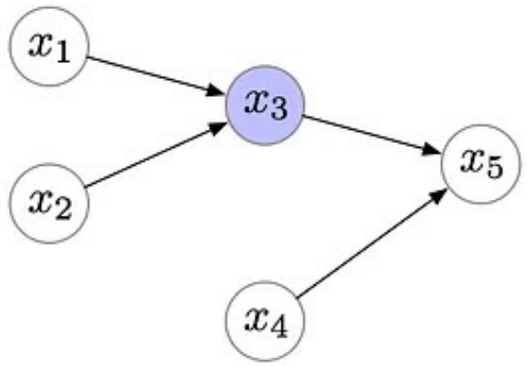
https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

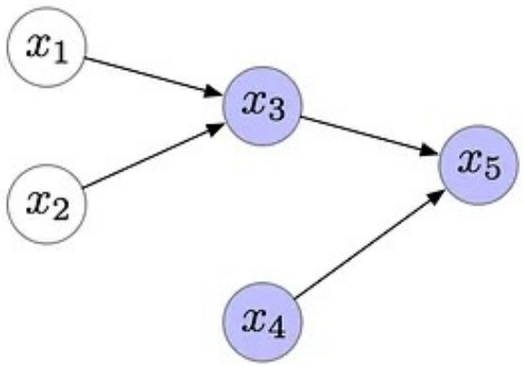
Pebble Analogy

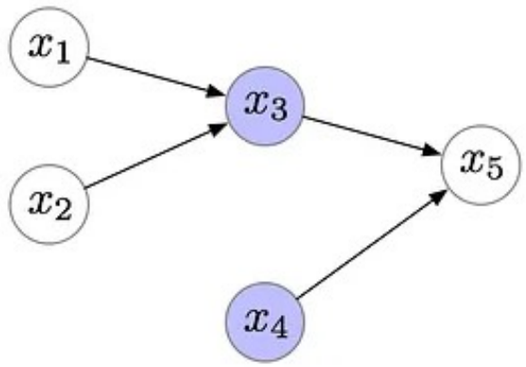


Pebble Analogy

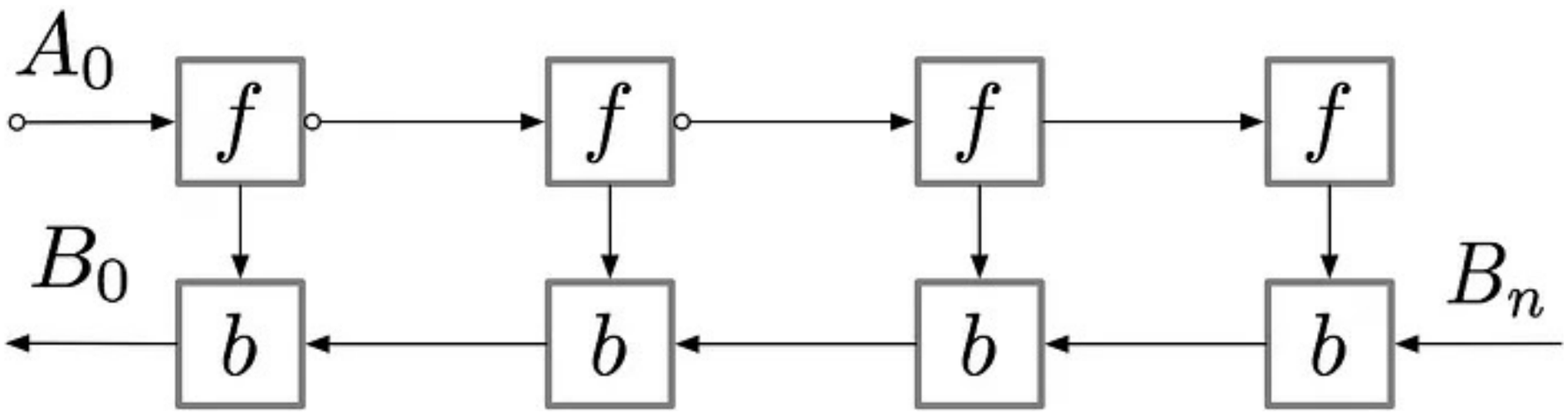








Gradient Computation



Checkpoints every sqrt(n) steps

Memory Requirement	$O(\sqrt{n})$
Compute Requirement	$O(n)$
Forward calcs per node	1 to 2

https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

FlashAttention-2

Additionally parallelizing the attention computation over sequence length

Partitioning the work between GPU threads to reduce communication and shared memory reads/writes

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, LlamaForCausalLM

model_id = "tiiuae/falcon-7b"
tokenizer = AutoTokenizer.from_pretrained(model_id)

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.bfloat16,
    attn_implementation="flash_attention_2",
)
```

model's dtype must be fp16 or bf16

DeepSpeed Good Practices in Training

Version Control and Reproducibility

Use some form of version control

Save the `githash`

Save all the hyperparameters associated with the experiment

Seed your random generators

Specify all the packages and their versions

`requirements.txt` file,
conda `env.yaml` file or
`pyproject.toml` file

DeepSpeed Good Practices in Training

Seed your random generators

```
import torch
torch.manual_seed(0)
```

```
import random
random.seed(0)
```

```
import numpy as np
np.random.seed(0)
```

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Seed your random generators

CUDA convolution

Based on parameters, cuDNN will select the fastest algorithm
Different runs end up with different results

torch.backends.cudnn.benchmark = False
Causes cuDNN to deterministically select an algorithm

Avoiding nondeterministic algorithms

torch.use_deterministic_algorithms()

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Seed your random generators

DataLoader

Default - each worker gets different random seed

```
def seed_worker(worker_id):
    worker_seed = torch.initial_seed() % 2**32
    numpy.random.seed(worker_seed)
    random.seed(worker_seed)

g = torch.Generator()
g.manual_seed(0)

DataLoader(
    train_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g,
)
```

https://pytorch.org/docs/stable/notes/randomness.html

DeepSpeed Good Practices in Training

Writing Unit Tests

```
def test_model_checkpointing(checkpoint_dir: str):
    train_params = {
        "checkpoint_dir": checkpoint_dir,
        "checkpoint_every": 2,
        "num_layers": 2,
        "num_heads": 4,
        "ff_dim": 64,
        "h_dim": 64,
        "num_iterations": 5,
    }
    exp_dir = train(**train_params)
   # now check that we have 3 checkpoints
    assert len(list(exp_dir.glob("*.pt"))) == 3
    model = create_model(
        num_layers=train_params["num_layers"],
        num_heads=train_params["num_heads"],
        ff_dim=train_params["ff_dim"],
        h_dim=train_params["h_dim"],
        dropout=0.1,
    )
    optimizer = torch.optim.Adam(model.parameters())
    step, model, optimizer = load_model_checkpoint(exp_dir, model, optimizer)
    assert step == 5
    model_state_dict = model.state_dict()
    correct_state_dict = torch.load(exp_dir / "checkpoint.iter_5.pt")
    correct_model_state_dict = correct_state_dict["model"]
    assert set(model_state_dict.keys()) == set(correct_model_state_dict.keys())
    assert all(
        torch.allclose(model_state_dict[key], correct_model_state_dict[key])
        for key in model_state_dict.keys()
    )
    # Finally, try training with the checkpoint
    train_params.pop("checkpoint_dir")
    train_params["load_checkpoint_dir"] = str(exp_dir)
    train_params["num_iterations"] = 10
    train(**train_params)
```