Accuracy Gains Without Training Your Own Model: Parallel Reasoning for LLM Apps

HTN 2025 Tech Talk

quick intro 👋

Eugene
Cho

Member of Technical Staff

- I work on the core post-training team
- Prev worked a lot on synthetic data + reasoning

some helpful pre-requisite knowledge

- a large language model (LLM) auto-regressively predicts the next token given some past text
    - for simplicity's sake, you can think of a token as a word
    - modern day LLMs are specifically trained to act like agentic assistants to user queries (done through post-training)

- LLMs are NON deterministic
    - that is, if you ask an LLM the exact same question multiple times, chances are the output will be different
    - the degree of non-determinism/randomness is steerable through a parameter called **temperature**. Higher temperature == more random

Agenda

01 what is "test-time compute"?

02 scaling test-time compute

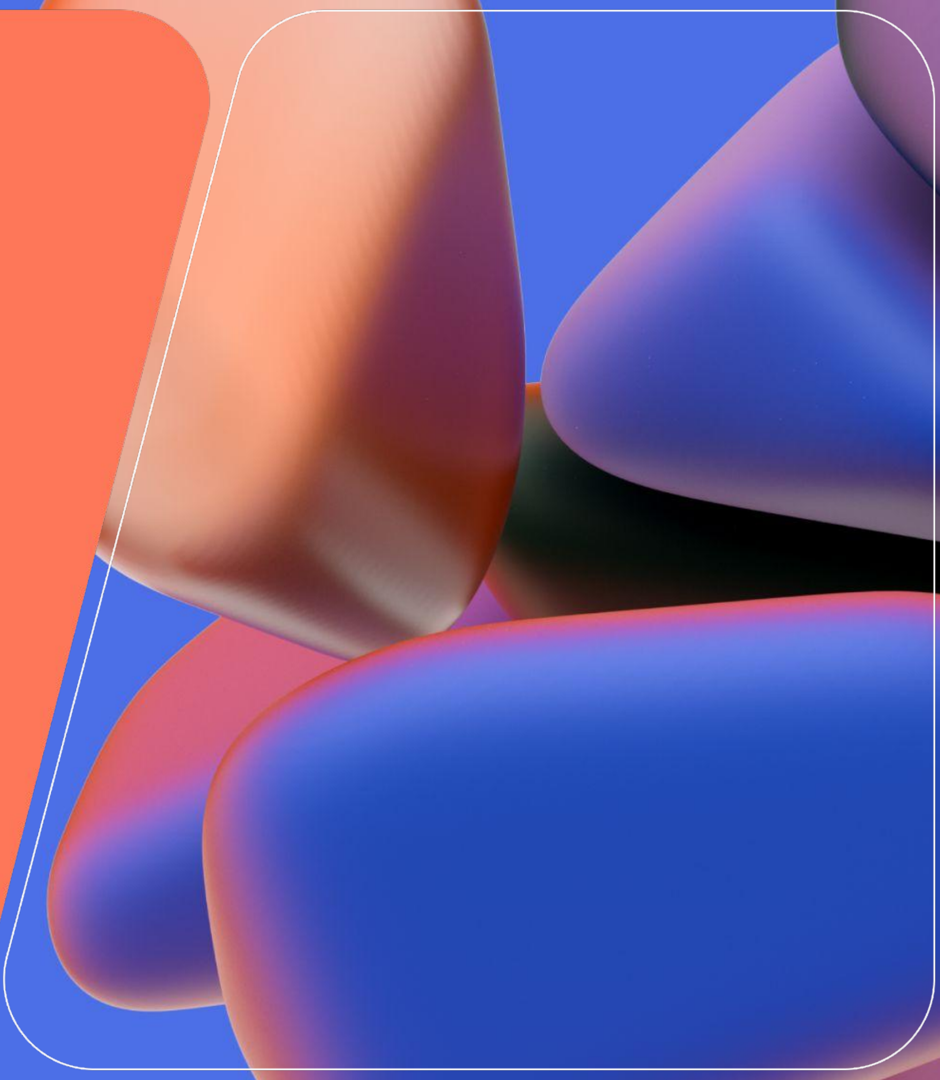
03 verifiers & how to implement them for your usecase

04 some demos (including the tradeoffs)

cohere

01

what is “test-time compute”?



“train-time compute”

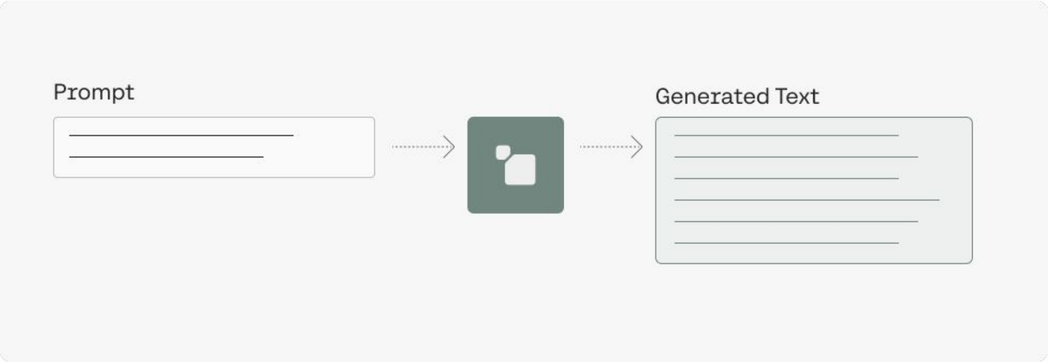
The amount of computational power used to train an AI model (i.e. compute gradients, update weights, etc.)

“test-time compute”

The amount of computational power used by an AI model at “inference” or “test”-time (i.e. using a model to provide predictions on new data)

in our case

test-time compute in LLM applications refers to the amount of energy spent generating response(s) to a user request/prompt, or more generally, some kind of agentic (tool calling) workflow



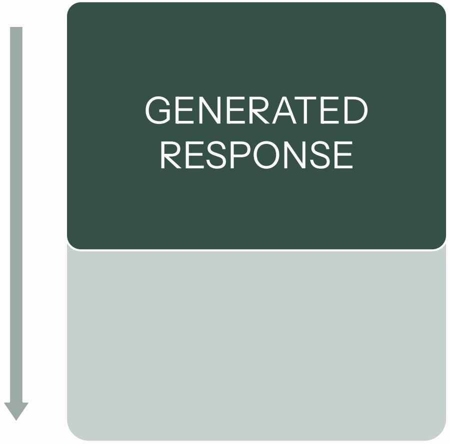
02

scaling test-time compute

two ways to scale

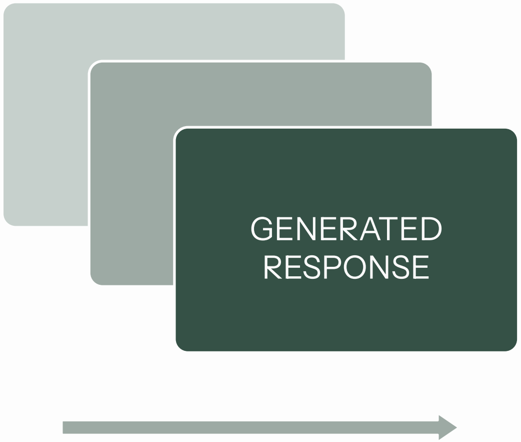
vertically

increasing response
trajectory length



horizontally

increasing the number of
response trajectories



vertical scaling of ttc (brief)

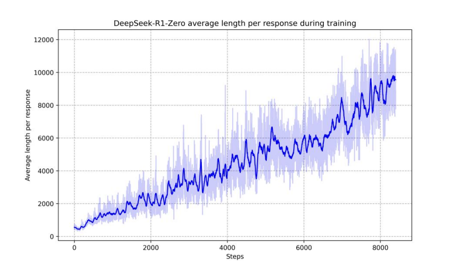
there are a few ways to do this:

1. prompt the model to think step by step (Chain of Thought Prompting)

which is larger 9.9 or 9.11? → 9.11

which is larger 9.9 or 9.11? think step by step → Looking at the tens digit of both numbers, 9 is larger than 1, thus 9.9 is larger.

2. use a reasoning model (trained with RL which learns to natively reason for longer) (DeepSeek R1 Paper)
   a. most reasoning models allow you to control the amount of thinking it does via a thinking budget/effort API param.



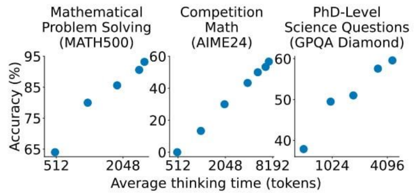


Figure 1. **Test-time scaling with s1-32B.** We benchmark **s1-32B** on reasoning-intensive tasks and vary test-time compute.

S1 paper

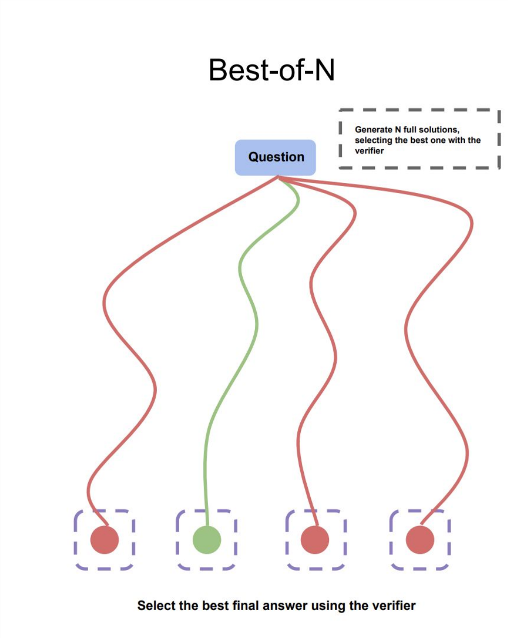
- Literature has established scaling laws that as we increase response generation length, **accuracy/performance** increases!

horizontal scaling of ttc (focus of today)

we can also just generate multiple responses for the same prompt. using these responses, we can then design a method of selecting or creating the best response based on some definition of correctness to help boost accuracy

There are a few names / variations of this:

- "best-of-N"
- parallel reasoning
- maj@k (self-consistency)



why does this work?

- recall LLMs are **non-deterministic**, so it could be the case that *sometimes* the model will answer a prompt “correctly” and *sometimes* it won’t
  - For many tasks, a well aligned LLM will not require many “trials” to get at least one correct trajectory

- for certain use cases, verifying correctness is often an easier task than answering the question correctly cohere

03

verifiers

what is a verifier?

for the purpose of this talk, a **verifier** is an entity that assigns a label of correctness to a single LLM trajectory.

- think of it as an arbitrary function ***v*** which takes in an LLM trajectory ***t*** as input, and spits out a boolean (True or False), representing the correctness of ***t***.That is:

$$v: t \rightarrow \{\text{True}, \text{False}\}$$

  - Note: in practical and more complex cases, the verifier function can output any scalar

implementing my verifier

it all boils down to defining “correctness” in your use case

- Can be as simple as:
  - A property of the LLM response meeting some criteria
    - Provides code that passes a pre-defined unit test
    - Response contains a specific desired keyword

  - A change in some external state (will come up if model has access to tools)
    - A email is sent successfully
    - A status is toggled off

- Can be as complex as:
  - A combination of simple conditions like the above:
    - I.e. **only if** all the conditions above are met, then the response is correct

  - Using an LLM to provide a label based on a more subjective criteria
    - ex: let’s say my usecase is a chatbot that outputs encouraging messages, I can use another LLM to verify whether or not the message is encouraging or not!

synthesis: how to select the “best” response?

- it’s use case dependent
    - you’ll likely need to get your hands dirty and try a few different applicable approaches

- some popular synthesization approaches (easiest to hardest):
    - **Ground-truth programmatic verification**
        - Use this when it is easy to programmatically verify the correctness of an **arbitrary response** (ex. Basic textual properties, quantitative measurements)
        - Here your verifier directly selects your best response !

    - **take the majority answer (maj@k)**
        - Use this when the correctness of outcomes/answers are not known ahead of time, but the space of possible outputs is objective and atomic such that comparing amongst them is simple (ex. Integer answers)
        - Also is only practical if you suspect the model gets the answer right *most* of the time

    - **ask another LLM (or a *panel* of LLMs) to choose the best one (most complex and hand-wavy)**
        - Use this when outcomes are more difficult to objectively or programmatically compare (ex. writing quality)

best ML practices

BEFORE DOING ANYTHING:

- Define your verifier
- Define an evaluation set
  - this consists of prompts + some definition of a ground truth / correct outcome
  - the larger the evaluation set the better, but in the interest of time you may want to keep it small (< 100 samples)

- evaluate your synthesization methods (including your baseline: which is where you just give the model one chance to solve the problem!)
  - **Evaluate your baseline FIRST before implementing anything else to save yourself time**)
  - If your baseline performance is mid, only then should you consider using test-time scaling!

how do i know if scaling test-time compute will be fruitful?

a checklist

- if there is a clear way to define “correctness” in your usecase ✅
- if the baseline performance of the model i’m using is trash ✅
- if there is some <u>non-zero probability</u> that the LLM can perform the task correctly ✅
  - The higher the probability, the better!

If the checklist is not met, then horizontal scaling will likely **not** be fruitful 💔

- what do i do then?
  - try again with a larger / stronger model (more expensive! 😒)
  - might need to do task specific fine-tuning (doomed for a hackathon 😒)

04

demo (including tradeoffs)

toy problem

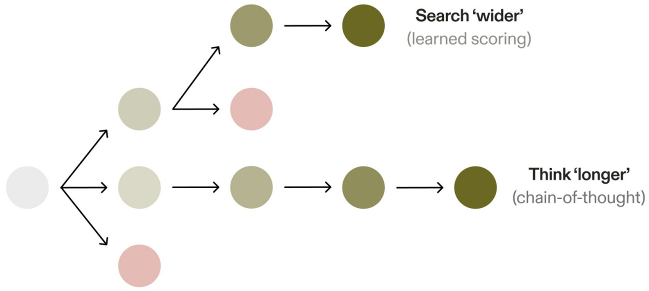
Spec:

- I have an application that fills in job applications for people.
- The problem:
    - many text boxes in job applications have word count limits.
- The LLM I'm using out of the box is not always good at adhering to the word count limit, despite me asking it to :(!
- For simplicity:
    - We will assume that all the responses the LLM gives are **high quality!** Realistically, this will not be the case, and extra work should be done into prompt engineering, etc.
    - We will also assume that the longer response, the higher quality it is! This is also not really the case!

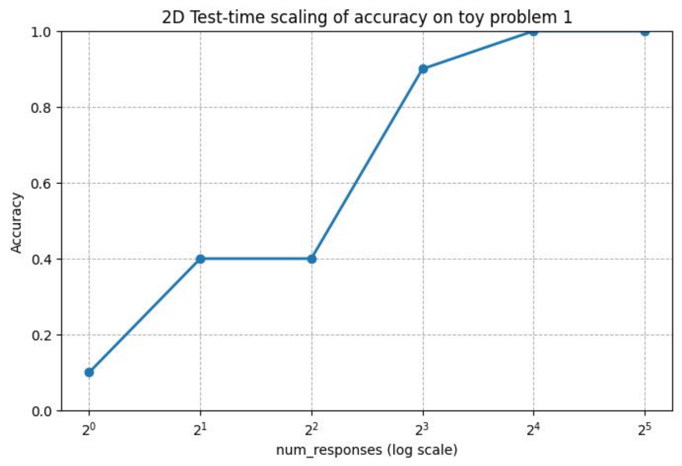
Follow along here:
https://github.com/cohere-ai/htn-2025-techtalk/blob/master/toy_problem.ipynb

final food for thought 🤔

What happens if we scale test-time compute both vertically and horizontally at the same time? (2 dimensionally)



even more gains!!! but you are going to go bankrupt 😒.
but if it is worth it, then go ahead!



QUESTIONS?

THANKS FOR LISTENING