Applied Machine Learning for Business Analytics

Lecture 6: LLM’s Inference & Reasoning

Lecturer: Zhao Rui

Agenda

1. LLM Inference
   a. Latency vs Throughput
   b. Batching
   c. Quantization
   d. KV Cache
   e. Speculative Decoding
2. Prompting
3. Understanding Reasoning Models

1. LLM Inference

Key Metrics in LLM Inference

- Latency Metrics
    - **Time to First Token (TTFT)**: Time from request to first output token
        - Critical for perceived responsiveness
        - Target: <500ms for “instant” feel
    - **Time Per Output Token (TPOT)**: Average generation time per subsequent token
        - Determines how smoothly text flows
    - Total Latency: TTFT + (TPOT x output_length)

Key Metrics in LLM Inference

- Throughput Metrics:
  - **Tokens per Second**: Output tokens generated across all concurrent requests
  - **Requests per Second**: Total requests processed per unit time

Key Metrics in LLM Inference

- Memory Metrics:
  - **Model Weights**: fixed cost, must fit in GPU memory
  - **Peak Memory**: Weights + KV Cache + Activations

Tradeoff Relationships

Memory ⟷ Batch Size ⟷ Throughput ⟷ Latency

Those three dimensions are interconnected - optimizing one often impacts the others

Two Phases of LLM Inferences

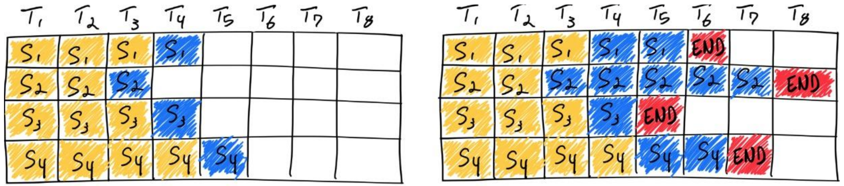
- Prefill vs Decode
  - Prefill Phase:
    - Process entire input prompt in parallel
    - High arithmetic intensity already
  - Decode Phase:
    - Generate tokens one at a time
    - Low arithmetic intensity: same weights loaded, minimal compute performed
    - However, batching can increase the arithmetic intensity in decode

Compute-to-memory ratio

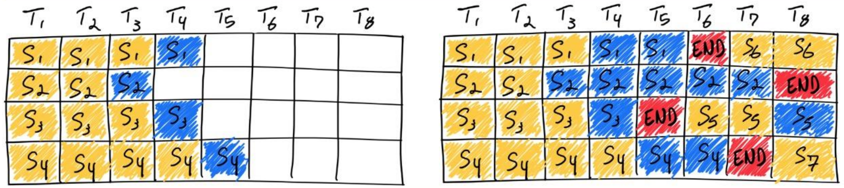
Phase	What Happens	Bound By	GPU Utilization
Prefill	Process input prompt	Compute	High
Decode	Generate tokens	Memory bandwidth	Low

Batching Strategies

- Static Batching
  - Wait for batch to fill, than process all together
  - Fast requests wait for slow ones
  - Poor Latency in low traffic scenarios



- Continuous Batching (State-of-the-art)
  - Insert new requests as old ones complete
  - No waitting for entire batch to finish



source: https://www.anyscale.com/blog/continuous-batching-llm-inference

Quantization

- Full Precision Models (FP32 or FPP16) are:
  - Large
  - Memory-bandwidth limited
  - Expensive to serve
- Quantization reduces precision to save memory

Almost lossless for most models

Format	Bits	Size (7B model)	Typical Usage
FP32	32	28 GB	Training
FP16	16	14 GB	Training, inference
INT8	8	7 GB	Inference
INT4	4	3.5 GB	Inference
IN2	2	1.7 GB	Experimental

Quantization Approaches

- Post-Training Quantization
  - Quantize after training
  - Faster, no retraining required
  - Some performance drop
  - AWQ, GPTQ, etc
- Quantization-Aware Training
  - Train with quantization in mind
  - Better accuracy
  - More expensive
  - QLoRA, etc

Use the Space below to help you pick a quantization method depending on your hardware and number of bits to quantize to.

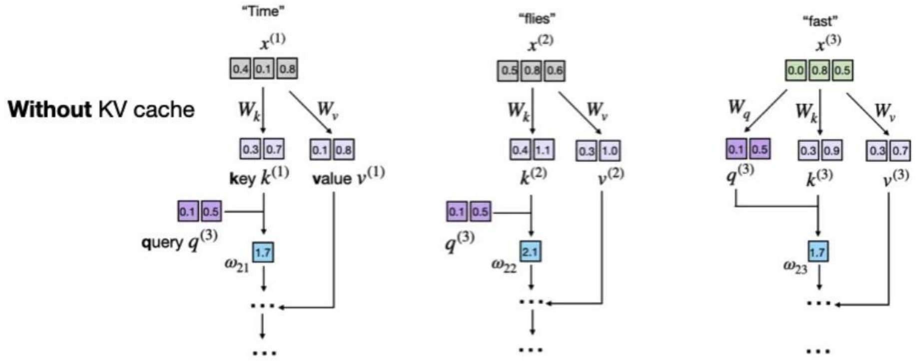
Quantization Method	On the fly quantization	CPU	CUDA GPU	ROCm GPU	Metal (Apple Silicon)	Intel GPU	Torch compile()	Bits	PEFT Fine Tuning	Serializable with 🤗 Transformers	🤗 Transform Suppor
AQLM	🔴	🟢	🟢	🔴	🔴	🟢	🟢	1/2	🟢	🟢	🟢
AutoRound	🔴	🟢	🟢	🔴	🔴	🟢	🔴	2/3/4/8	🔴	🟢	🟢
AWQ	🔴	🟢	🟢	🟢	🔴	🟢	?	4	🟢	🟢	🟢
bitsandbytes	🟢	🟢	🟢	🟡	🟡	🟢	🟢	4/8	🟢	🟢	🟢
compressed-tensors	🔴	🟢	🟢	🟢	🔴	🔴	🔴	1/8	🟢	🟢	🟢
EETQ	🟢	🔴	🟢	🔴	🔴	🔴	?	8	🟢	🟢	🟢
FP-Quant	🟢	🔴	🟢	🔴	🔴	🔴	🟢	4	🔴	🟢	🟢
GGUF / GGML (llama.cpp)	🟢	🟢	🟢	🔴	🟢	🟢	🔴	1/8	🔴	See Notes	See Notes
GPT-QModel	🔴	🟢	🟢	🟢	🟢	🟢	🔴	2/3/4/8	🟢	🟢	🟢
HIGGS	🟢	🔴	🟢	🔴	🔴	🔴	🟢	2/4	🔴	🟢	🟢
HQQ	🟢	🟢	🟢	🔴	🔴	🟢	🟢	1/8	🟢	🔴	🟢
optimum-quanto	🟢	🟢	🟢	🔴	🟢	🟢	🟢	2/4/8	🔴	🔴	🟢
FBGEMM_FP8	🟢	🔴	🟢	🔴	🔴	🔴	🔴	8	🔴	🟢	🟢
torchao	🟢	🟢	🟢	🔴	🟡	🟢		4/8		🟢🔴	🟢
VPTQ	🔴	🔴	🟢	🟡	🔴	🔴	🟢	1/8	🔴	🟢	🟢
FINEGRAINED_FP8	🟢	🔴	🟢	🔴	🔴	🟢	🔴	8	🔴	🟢	🟢
SpQR	🔴	🔴	🟢	🔴	🔴	🔴	🟢	3	🔴	🟢	🟢
Quark	🔴	🟢	🟢	🟢	🟢	🟢	?	2/4/6/8/9/16	🔴	🔴	🟢

source: https://huggingface.co/docs/transformers/en/quantization/overview

Quantization affects different models differently.

KV Cache

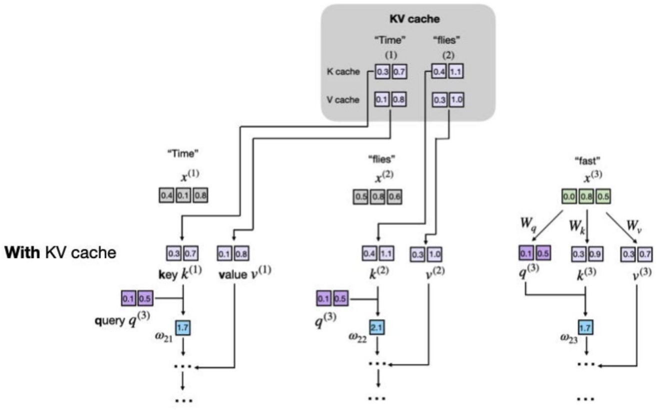
- Problem Statement: Redundant Computation
  - Without KV Cache: Recompute K,V projection
  - The waste:
    - Token 1: compute k1,v1 -> attention
    - Token 2: Compute k1,v1,k2,v2 -> attention
    - Token 3: compute k1,v1,k2,v2,k3,v3 -> attention



source:https://medium.com/@joaolages/kv-caching-explained-276520203249

KV Cache

- Solution: Store previously computed K and V vectors in memory
  - To generate token 3 (“fast):
    - Load k1, v1, k2,v2 from cache (fast memory read)
    - Compute only q3, k3, v3 for the new token
    - Attend q3 to all key vectors -> get attention weights
    - Store k3, v3 in cache for future tokens



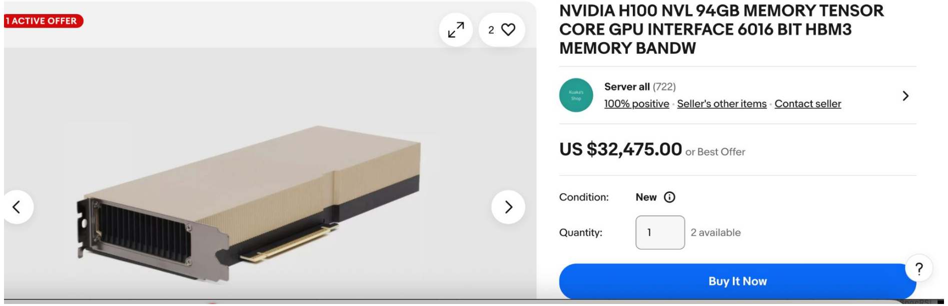
source:https://medium.com/@joaolages/kv-caching-explained-276520203249

KV Cache: Compute vs Memory

- KV Cache is trading Memory for Speed
  - Tale Llama 70B Model Size:
    - Model Parameters: ~140 GB
      - For FP16, 70 billion times 2 bytes per parmas = 140 GB
      - For Int4, 70 billion times 0.5 bytes per params=35 GB
    - KV cache (32K context): ~10 GB @ FP 16
      - KV cache = 2 (K+V) × layers × kv_heads × head_dim × bytes_per_element × num_tokens

KV Cache: Limit Throughput

- Assume H100 is used for hosting
  - We need to quantize the model firstly

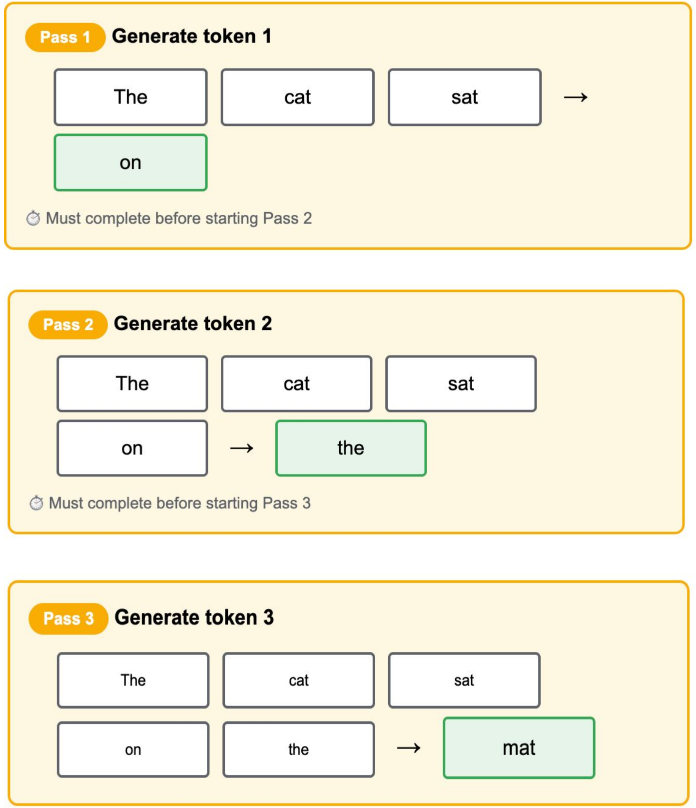


Configuration	Model Size	KV Cache	Available VRAM	Max Concurrent Requests
INT4 model + FP16 KV	35 GB	10 GB	59 GB	~5
INT4 model + INT8 KV	35 GB	5 GB	59 GB	~10

KV Cache Optimization

- Key Problems: for long context and a big batch size, KV cache might dominate memory usage
- Optimization Techniques:
  - Architecture-level:
    - Multi-Query Attention, Group-Query Attention, etc
  - KV Quantization
  - Memory Management:
    - PagedAttention (vLLM), RadixAttention (SGLang), etc
  - Hardware Optimization:
    - Tensor/Sequence Parallelism

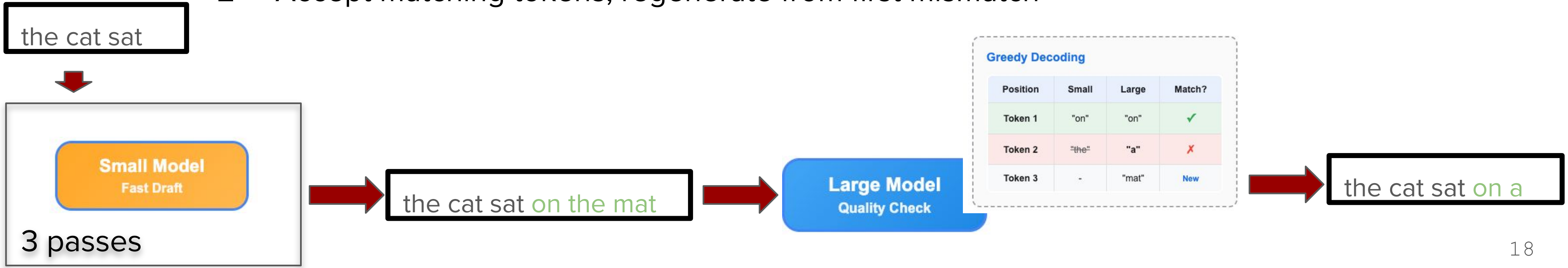
Bottleneck in Decoding



Three forward pass are required to generate three tokens.

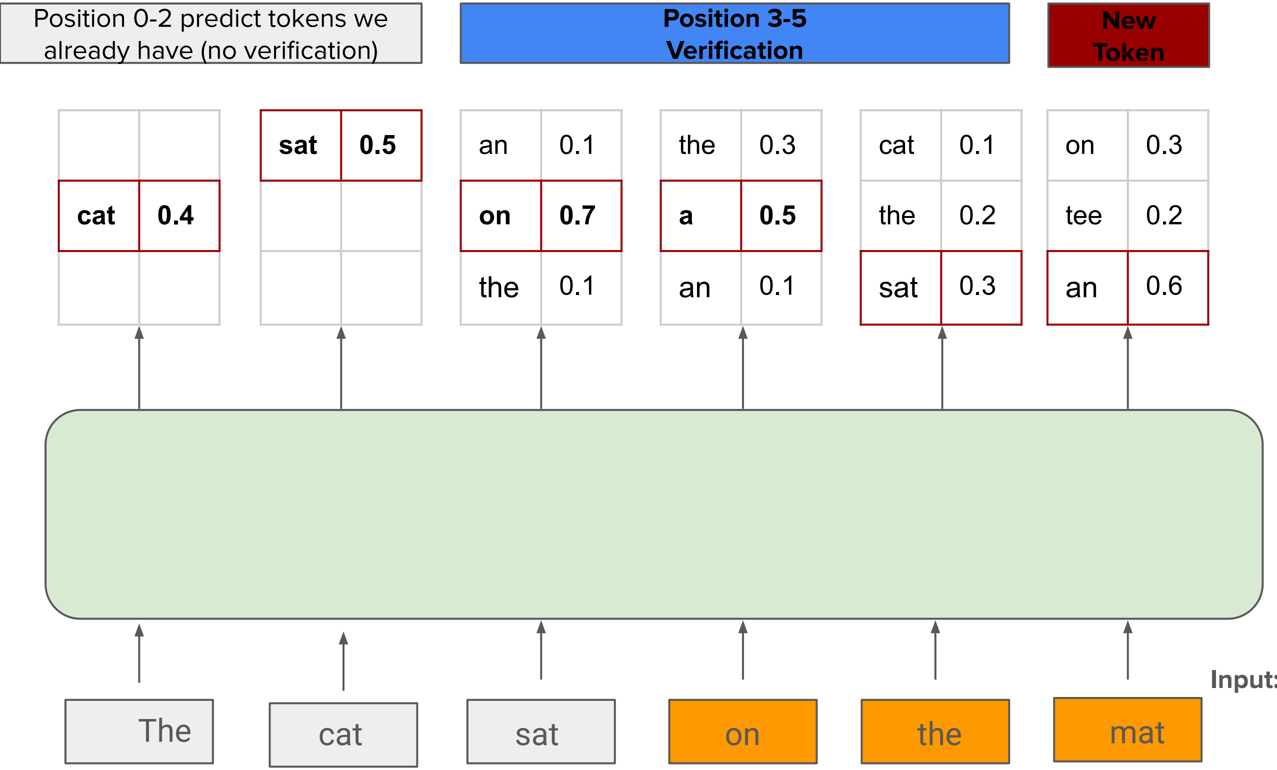
Speculative Decoding: Draft and Verify

- How it works
  - Draft Phase:
    - Small models generate K candidate tokens
  - Verify Phase:
    - Large model process those K tokens in one-forward pass
  - Accept/Reject:
    - Compare draft vs target model probabilities
  - Continue:
    - Accept matching tokens, regenerate from first mismatch



Next Iteration: Input becomes “the cat sat on a -> Small Model generates 3 new candidates

How Large Models Verify in One Pass



Input: Original + candidates

- Verification:
  - Position 3: Predicts on matches candidates
  - Position 4: Predicts a doesn't match "the"
  - Position 5: Predicts sat (not relevant)
  - Position 6: Predicts an (new token and also irrelevant)
- Result: Verify 3 candidates + genera
  1 new token in **one forward pass**

Speculative Decoding: Speed Up Formula

- The Speedup Formula:
  - Sequential (Old Way):
    - K forward pass on Large LM: K * T_large
  - Speculative Decoding:
    - K forward pass on small LM: K*T_small
    - One forward pass on Large LM: T_large
    - With a chance, it would might generate from 1 to K+1 tokens
- When It really speed-up:
  - Draft small model is much faster than target large model (>> 5x)
  - Draft small model has high acceptance rate (>50%)

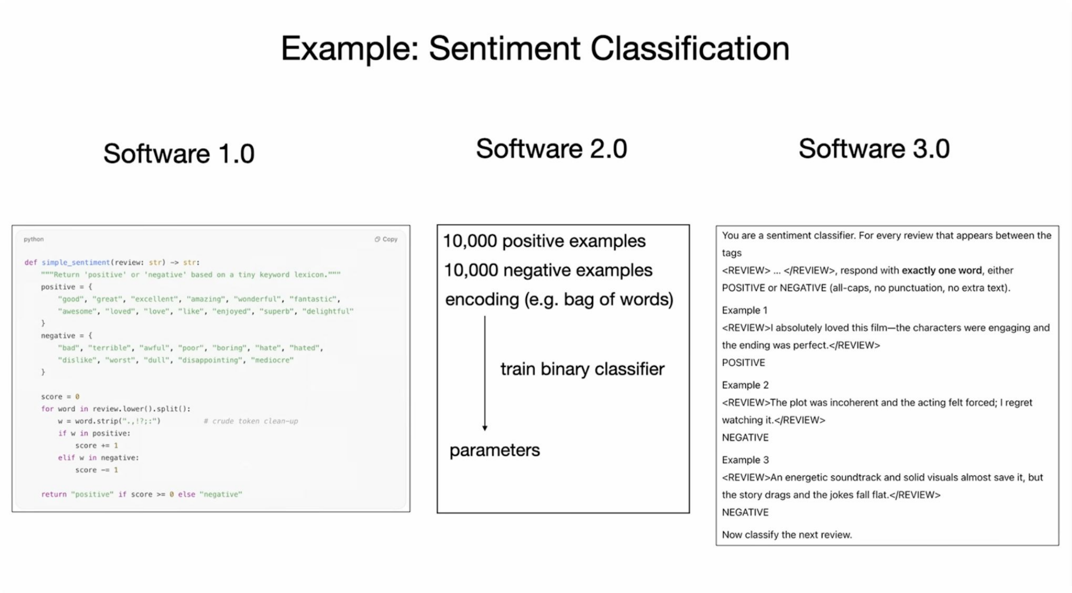
Inference Optimization

To Improve	Optimize
Max Batch Size	Memory(Quantization, PagedAttention, MQA,etc)
Throughput	Bigger Batch Size, Continuous Batching, etc
Latency	KV Caching, Smaller Batch, Speculative decoding, faster hardware, etc

2. Prompting Techniques

Prompting

- Prompts are the steering wheels to control LLMs to do what we want and also effectively boost their performance.

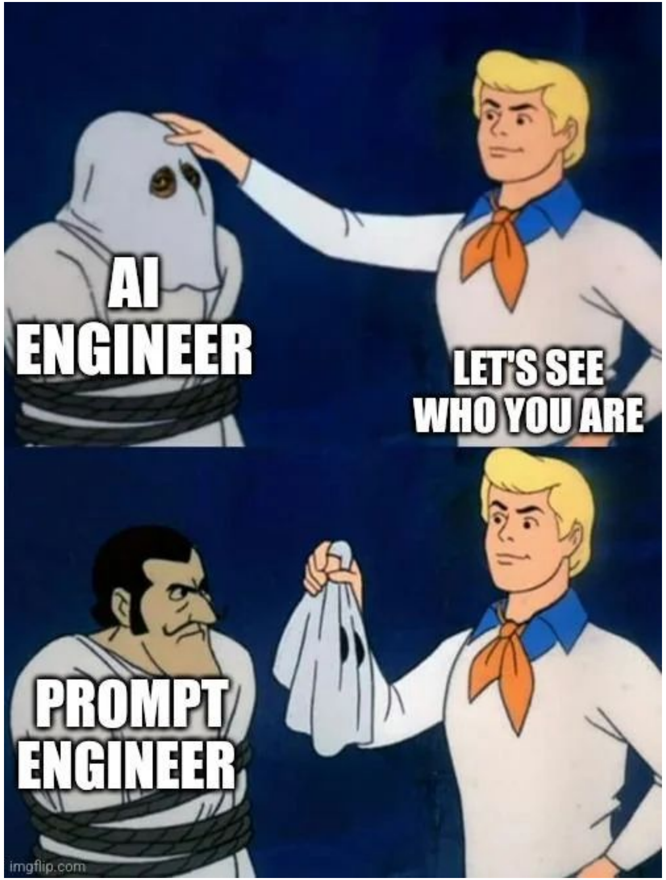


Andrej Karpathy: Software Is Changing (Again)

source:https://www.youtube.com/watch?v=LCEmiRjPEtQ

Prompting

Prompting is becoming a must-have skill for AI era



Prompt engineering (Do it first)

- It refers to methods for how to communicate with LLM to steer its behavior for desired outcomes <u>without updating the model</u>
  - More empirical, less scientific
- Advantages
  - Testing and learning early
  - When paired with evaluation it provides your baseline and sets up further optimization
- Disadvantages
  - Introducing new information
  - Reliably replicating a complex style or method
  - Minimizing token usage

Intuition behind Prompt Engineering

- LLMs understand better when you use familiar language and constructs
- LLMs can not know everything. If information is neither in training or in the prompt, they do not know it
- If you look at Prompt and you do not understand it, the prompt can not work

Prompting techniques

- Zero-shot prompting
  - No examples are given in prompt
- Few-shot prompting
  - A few shot examples of tasks are provided
- Chain of Thoughts prompting
  - Examples with the reasoning processes are improved.
  - It could be zero-shot (Think step by step) or few-shots
- Self-Consistency Prompting
  - Sample multiple times from output (typically with CoT) and aggregate most common results
- More prompting techniques could be found in this good survey and this openAI blog.

Source: https://arxiv.org/abs/2201.11903

Prompt Structure

- System prompt:
  - First message provided to LLM (usually hidden towards end user)
  - Provides persona, rules about LLM output, style

Resources

System Prompts

Copy page

See updates to the core system prompts on Claude.ai and the Claude iOS and Android apps.

Claude's web interface (Claude.ai) and mobile apps use a system prompt to provide up-to-date information, such as the current date, to Claude at the start of every conversation. We also use the system prompt to encourage certain behaviors, such as always providing code snippets in Markdown. We periodically update this prompt as we continue to improve Claude's responses. These system prompt updates do not apply to the Anthropic API. Updates between versions are bolded.

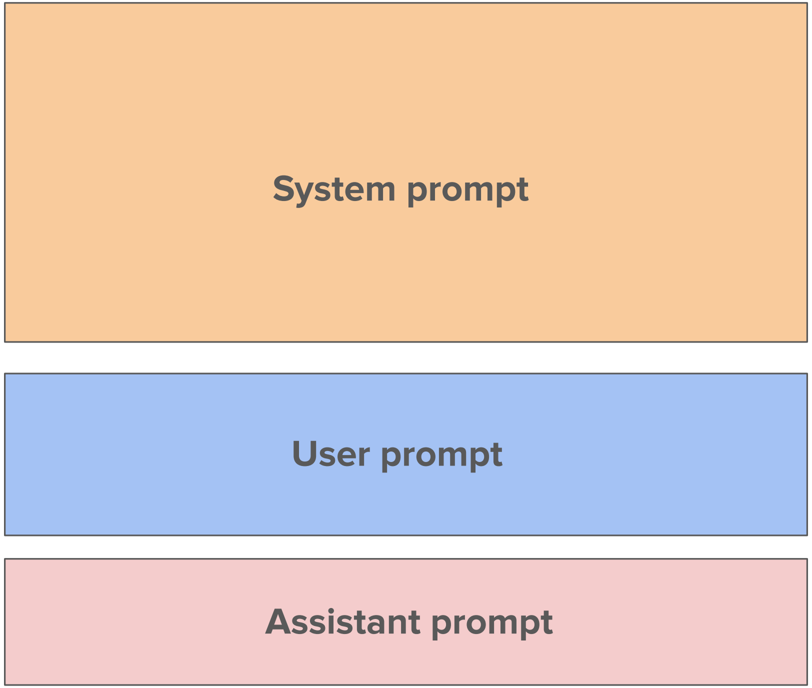
Claude uses a warm tone. Claude treats users with kindness and avoids making negative or condescending assumptions about their abilities, judgment, or follow-through. Claude is still willing to push back on users and be honest, but does so constructively - with kindness, empathy, and the user's best interests in mind. </tone_and_formatting> <user_wellbeing> Claude uses accurate medical or psychological information or terminology where relevant.

Claude cares about people's wellbeing and avoids encouraging or facilitating self-destructive behaviors such as addiction, self-harm, disordered or unhealthy approaches to eating or exercise, or highly negative self-talk or self-criticism, and avoids creating content that would support or reinforce self-destructive behavior even if the person requests this. Claude should not suggest techniques that use physical discomfort, pain, or sensory shock as coping strategies for self-harm (e.g. holding ice cubes, snapping rubber bands, cold water exposure), as these reinforce self-destructive behaviors. In ambiguous cases, Claude tries to ensure the person is happy and is approaching things in a healthy way.

source: https://platform.claude.com/docs/en/release-notes/system-prompts

Prompt Structure

- User Prompt:
  - The actual ask or instruction from users
- Assistant Prompt:
  - What the LLM actually generates



How to prepare a good prompt

- Start with:
  - Write clear instructions
  - Split complex tasks into simpler subtasks
  - Give GPTs time to “think”
  - Test changes systematically
- Extend to:
  - Provide reference text
    - Few-shot prompt -> RAG
  - Use external tools
    - Tool-use Prompt
- Prompting Practices from Antropic
  - https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
  - Ask LLM to improve your prompts

Best Practices

- Use role prompting to make system prompts more powerful

You are a helpful assistant that loves programming at the level of a senior software developer and is very detailed and pedantic in your answers.


I WON'T BECOME LIKE YOU!!
HOW MANY TIMES DO I HAVE TO SAY IT!?
I WANT TO BECOME HOKAGE!!

Best Practices

- Prompts should be formatted with structure
  - Multiple components like context, instructions, and examples, XML tags should be used

Example: Generating financial reports

Without XML tags, Claude misunderstands the task and generates a report that doesn't match the required structure or tone. After substitution, there is also a chance that Claude misunderstands where one section (like the the Q1 report example) stops and another begins.

Role	No XML Tags	With XML Tags
User	You're a financial analyst at AcmeCorp. Generate a Q2 financial report for our investors. Include sections on Revenue Growth, Profit Margins, and Cash Flow, like with this example from last year: {{Q1_REPORT}}. Use data points from this spreadsheet: {{SPREADSHEET_DATA}}. The report should be extremely concise, to the point, professional, and in list format. It should and highlight both strengths and areas for improvement.	You're a financial analyst at AcmeCorp. Generate a Q2 financial report for our investors. AcmeCorp is a B2B SaaS company. Our investors value transparency and actionable insights. Use this data for your report: <data>{{SPREADSHEET_DATA}} </data> <instructions> 1. Include sections: Revenue Growth, Profit Margins, Cash Flow. 2. Highlight strengths and areas for improvement. </instructions> Make your tone concise and professional. Follow this structure: <formatting_example> {{Q1_REPORT}} </formatting_example>

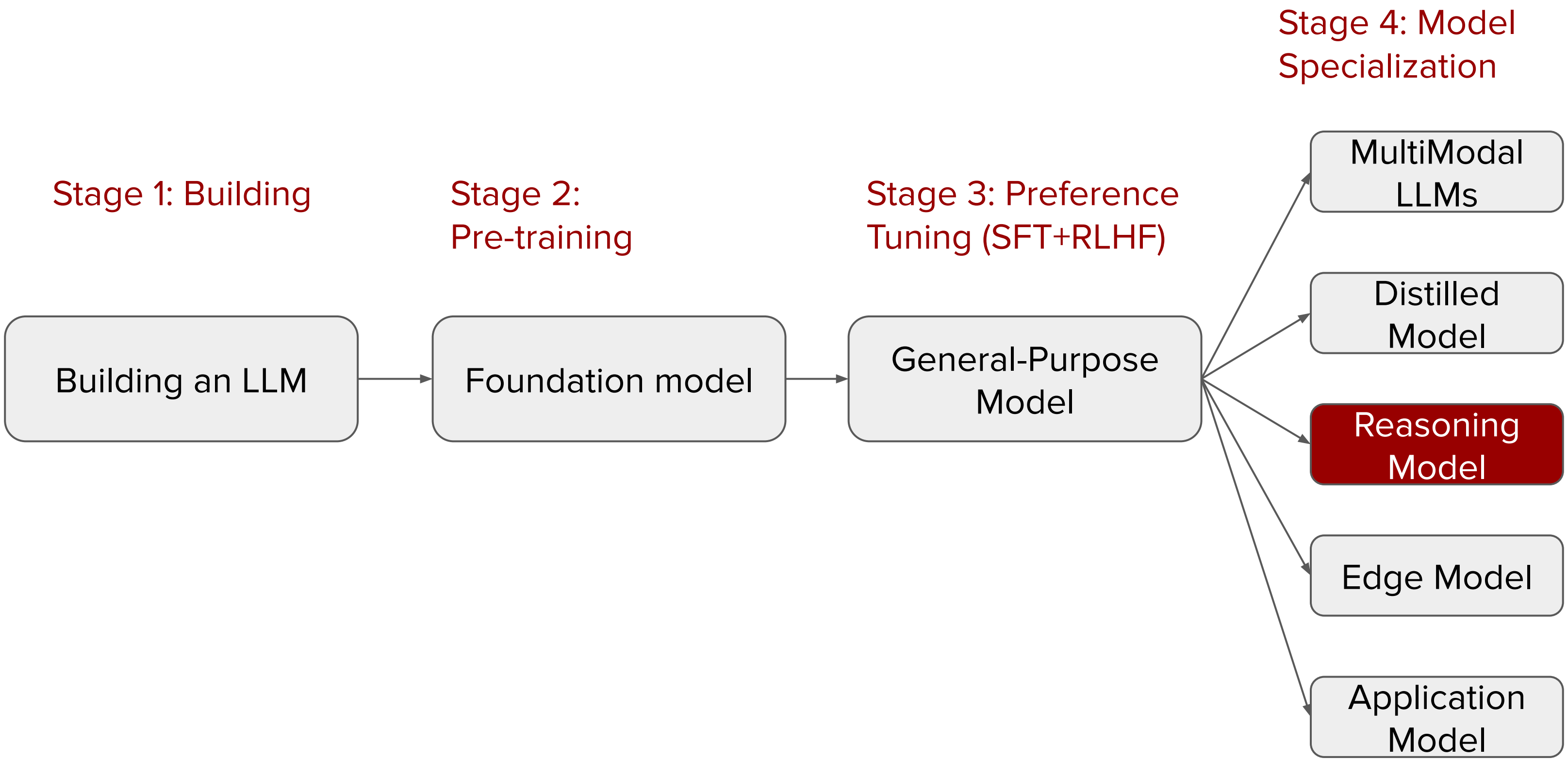
Source:
https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/use-xml-tags

3. Reasoning Models

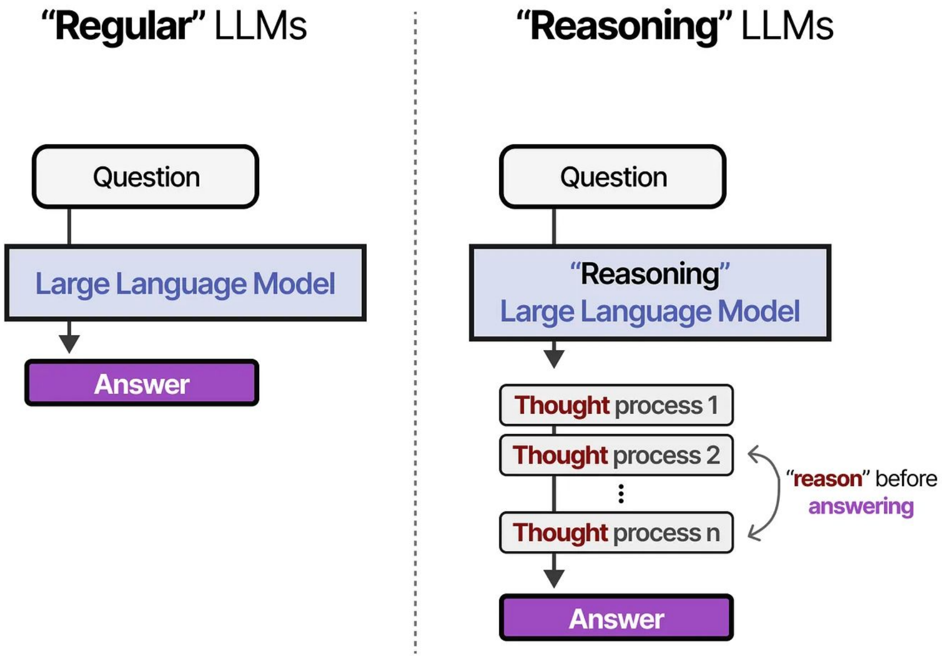
Source & Acknowledgements

The following pages especially lots of visual guide take reference from the following blog:
A Visual Guide to Reasoning LLMs

Developing LLM



What is a Reasoning Model?



Popular reasoning models in 2025

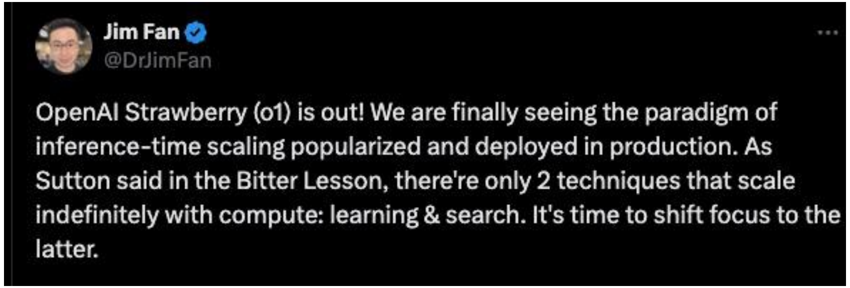
Instead of learning "What" to answer, reasoning models learn "How" to answer through structured intermediate steps called Chain-of-Thought (COT)

A Trivial Example

- Q: What is 3+2
  - General Purpose LLM: 5
  - Reasoning LLM:
    - <think> Adding 3 and 2 gives </think> 5
    - <think> If 3+1 = 4, 4+1=5, the total is </think> 5

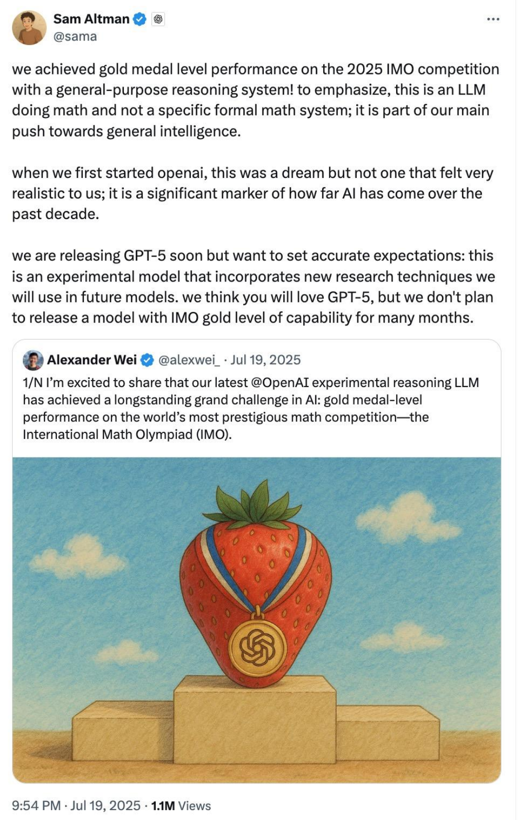
OpenAI o1

- First Large-scale Reasoning Model
- The [one line](#) from OpenAI:
  - AI models designed to spend more time thinking before they respond



Dominating Math Competition

- OpenAI o1 was the first AI to win IMO Gold Medal
- All reasoning models beat non-reasoning models significantly in AIME 2024
    - Leaderboard is [here](#)



Sam Altman
@sama

we achieved gold medal level performance on the 2025 IMO competition with a general-purpose reasoning system! to emphasize, this is an LLM doing math and not a specific formal math system; it is part of our main push towards general intelligence.

when we first started openai, this was a dream but not one that felt very realistic to us; it is a significant marker of how far AI has come over the past decade.

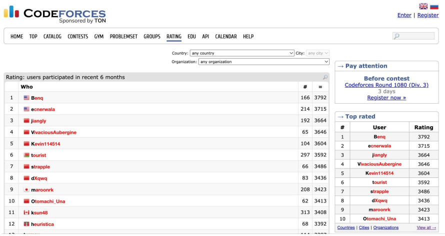
we are releasing GPT-5 soon but want to set accurate expectations: this is an experimental model that incorporates new research techniques we will use in future models. we think you will love GPT-5, but we don't plan to release a model with IMO gold level of capability for many months.

Alexander Wei @alexwei_ · Jul 19, 2025

1/N I'm excited to share that our latest @OpenAI experimental reasoning LLM has achieved a longstanding grand challenge in AI: gold medal-level performance on the world's most prestigious math competition—the International Math Olympiad (IMO).

9:54 PM · Jul 19, 2025 · 1.1M Views

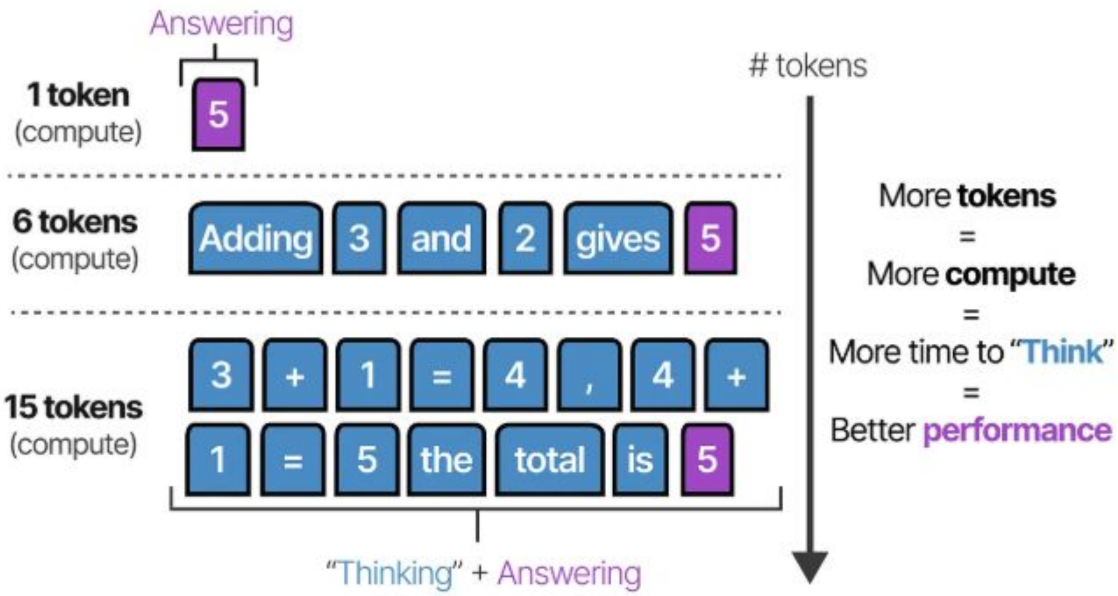
Crushing Coding Competition



- Codeforces-based Elo is used to rank competitive programmers
- LLM Models’ Performances

LLM Models	Elo rating	Beat x% of Coders
DeepSeek-R1	2029	96.3
OpenAI o1	1807	93
o1-mini	1605	86
GPT-4o	668	22

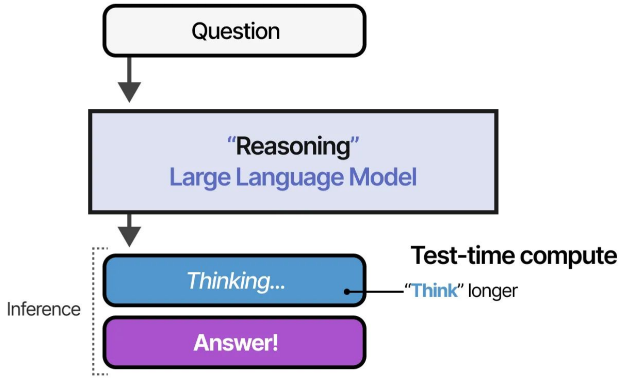
Test-time Compute



Test-time Compute: More compute (e.g., tokens) is spend generating the answer

The Paradigm Shift

- From Train-Time Compute to Test-Time compute



Train-Time Compute

- Model Size: # Parameters
- Dataset Size: # Tokens
- Compute: # FLOPs

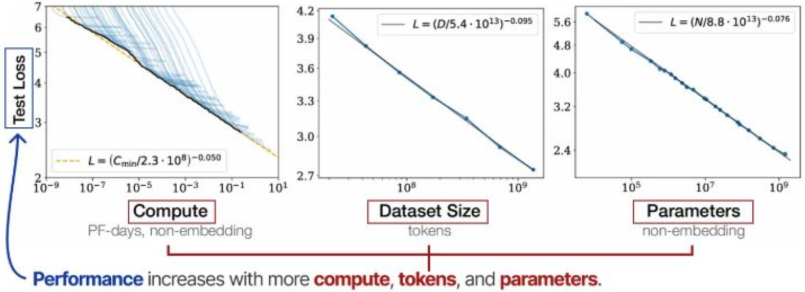
Hitting the wall in 2024

Test-Time Compute

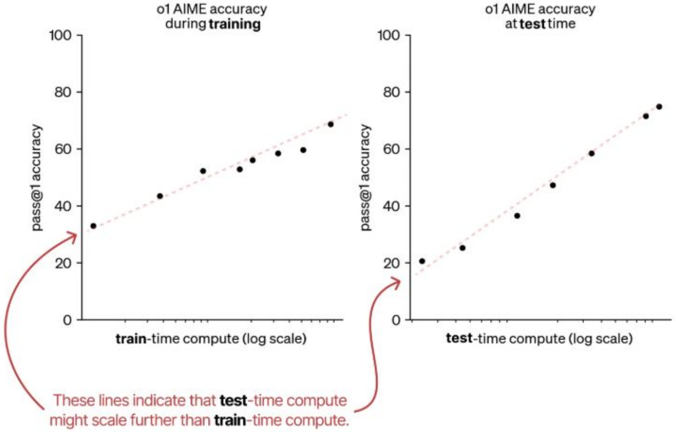
- Reasoning tokens: More thinking steps
- Search/Sampling: Multiple attempts
- Verification: self-checking

Scales with inference budget

The Paradigm Shift



Annotated figure of the "Scaling laws for neural language models" paper. It shows how performance may increase with different measures of compute (longer training, dataset size, and parameter size).



Annotated figure from "Learning to reason with LLMs". The red dotted line was added to demonstrate how OpenAI suggests the new paradigm might be test-time compute.

What is “Reasoning” Really

- It is not human-like thinking - it is intermediate tokens that unlock capabilities
- Why more tokens work?
  - **Hidden working memory:**
    - intermediate tokens provide “working memory” - each step conditions the next prediction, breaking complex problems into simpler sub-problem
  - **Search and path exploration:**
    - instead of just taking the first most likely token, reasoning allows the model to check many more candidates
  - **Self-Correction and “Verifiers”:**
    - Those tokens can act as its own critic. The model can pivot in the middle of process if a mistake is spotted.

Last Letter Concatenating Problems

What is the output when concatenating the last letter of each word in “artificial intelligence”?

No reasoning

The answer is “le”.

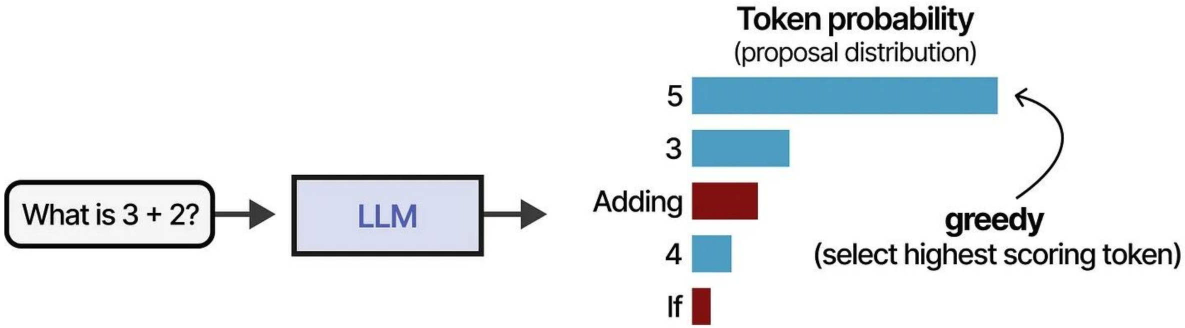
Reasoning

The last letter of “artificial” is “l”. The last letter of “intelligence” is “e”. Concatenating “l” and “e” leads to “le”. So the answer is “le”.

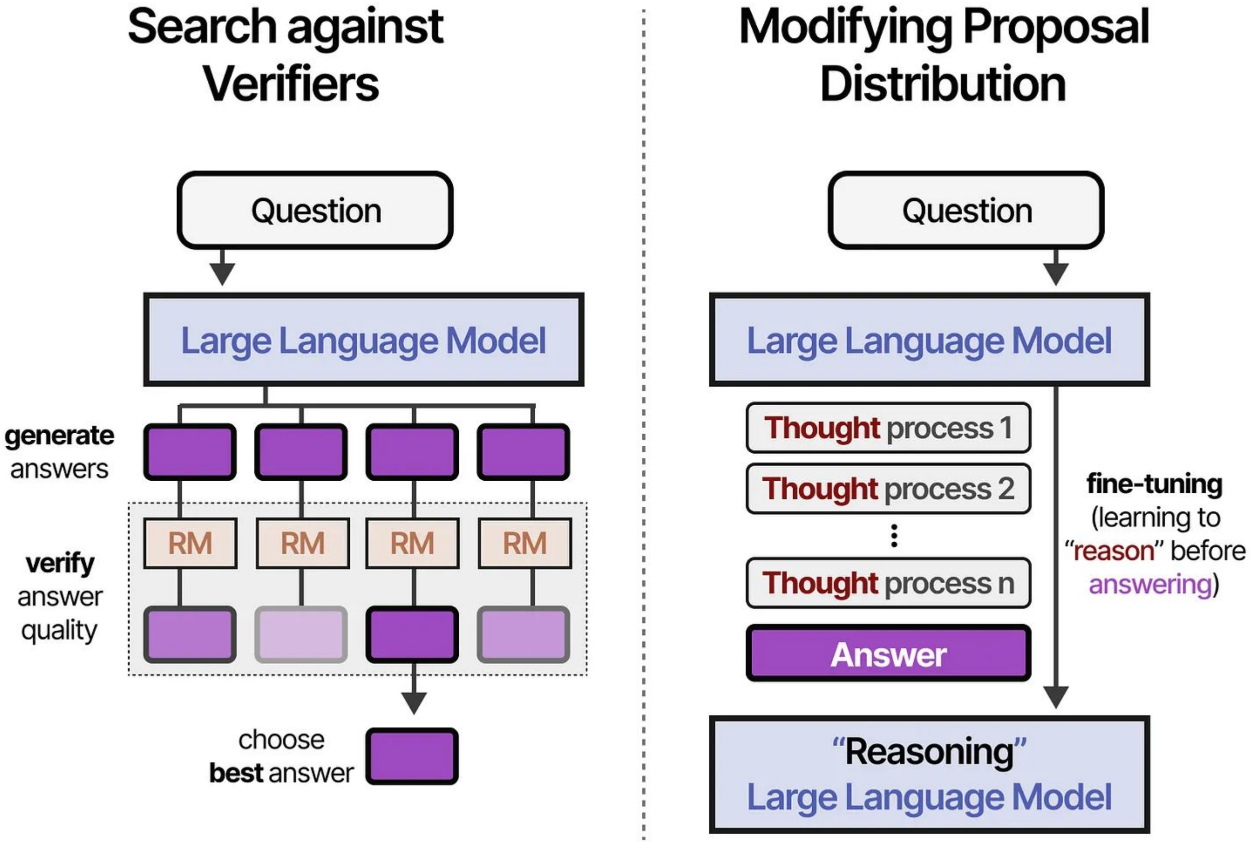
source: https://dennyzhou.github.io/LLM-Reasoning-Stanford-CS-25.pdf

The Model Already Knows

- LLM already has reasoning capabilities embedded in its weights from pre-training
- Challenge: Greedy decoding picks the highest-probability:**5**, skipping reasoning tokens
- Solution: Modify the decoding process to favor reasoning tokens



Two Types of Test-Time Compute



Search Against Verifiers

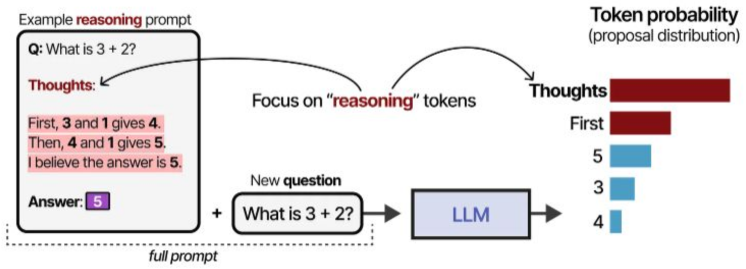
- Output-focused:
  - Generate multiple answers, then score and select the best one
- Two-steps Process:
  - Multiple samples of reasoning processes and answers are created
  - A verifier (reward model) scores the generated output
    - Two types of RM: outcome and process
- Techniques:
  - Majority Voting (self-consistency)
  - Best-of-N Sampling
  - Beam Search with Process RM
  - Monte Carlo Tree Search

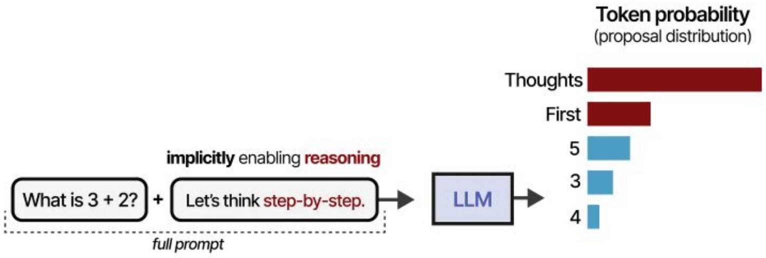
Modifying Proposal Distribution

- Input-focused:
  - Tune/train the model to naturally produce better reasoning tokens
- Techniques:
  - Prompting: CoT Prompting
  - Training: SFT on CoT, Self-Taught Reasoner, Reinforcement Learning (RLVR) and etc

CoT Prompting

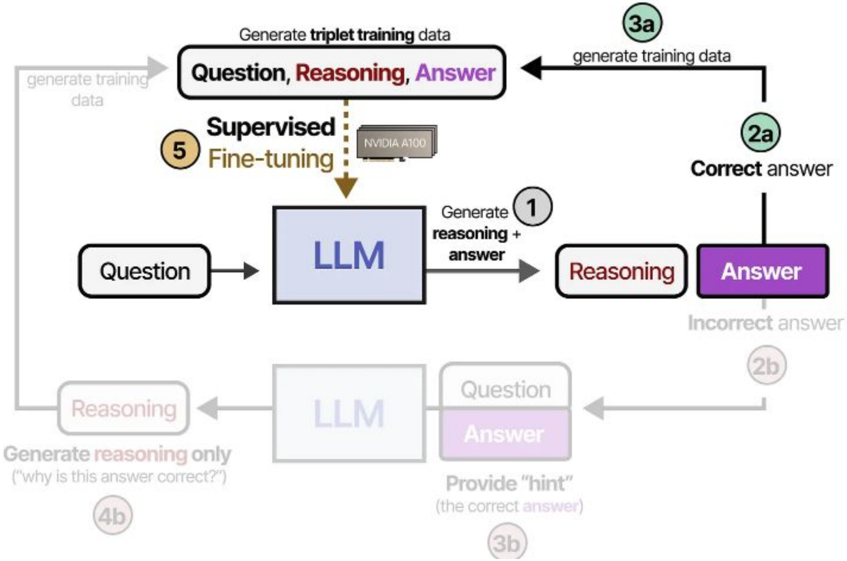
- It can be zero-shot or few-shot





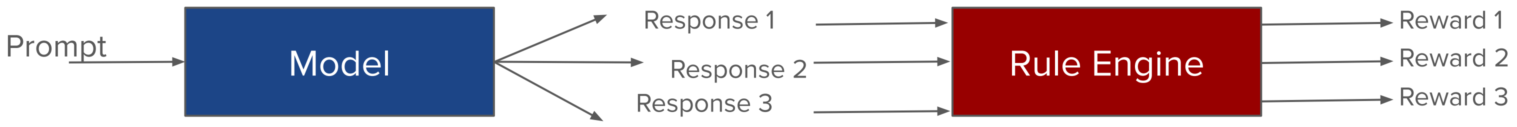
SFT

- The SFT data-triple training data: question, reasoning, answers
  - Human annotations
  - Synthetic data



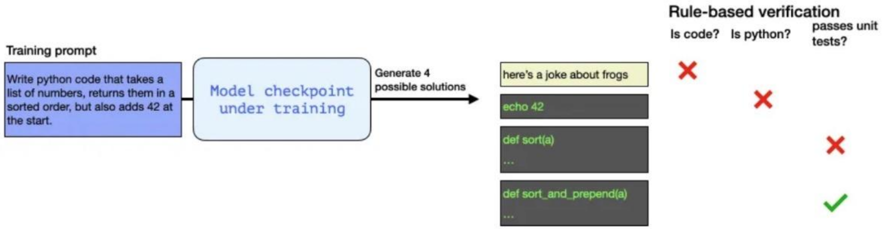
Reinforcement Learning with Verifiable Rewards

- It is used by **DeepSeek-R1**
- The reward is automatically calculated by rule-based systems



Verifiable Rewards

- Accuracy:
  - Math: validate answer
  - Coding: auto-validation
- Format: validate output format
  - Using <thinking> and <answer> tags



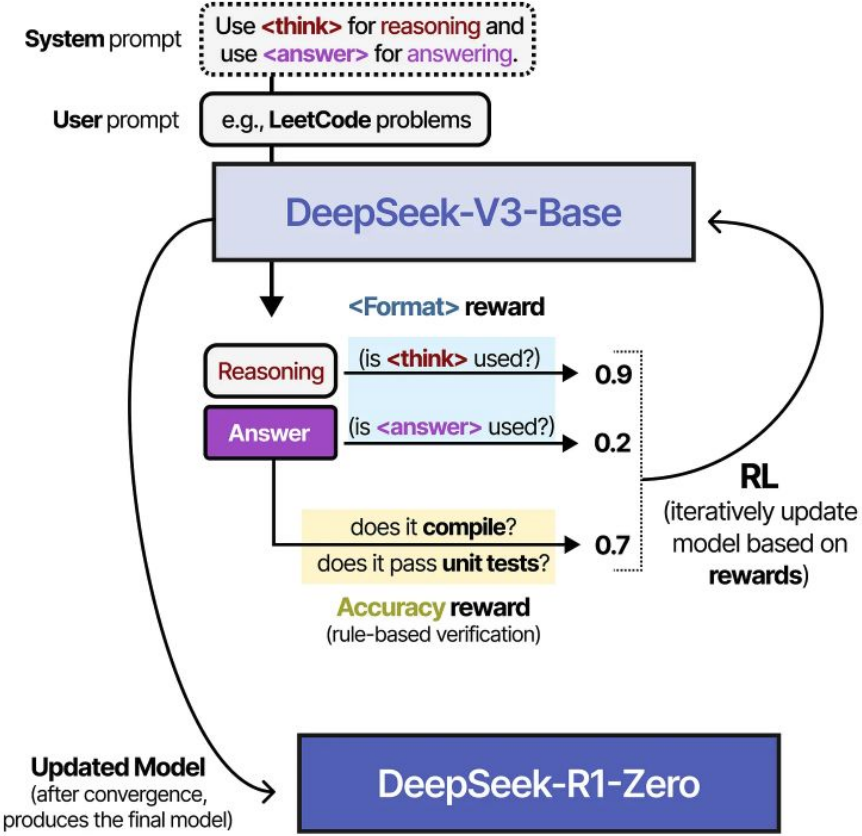
Source: https://newsletter.languagemodels.co/p/the-illustrated-deepseek-r1

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: prompt. Assistant:

Table 1 | Template for DeepSeek-R1-Zero. prompt will be replaced with the specific reasoning question during training.

Source: https://arxiv.org/pdf/2501.12948

RLVR for DeepSeek-R1



4 Ways to Build Reasoning Models

Test-time Scaling

No training required. Use prompting (CoT), voting, and search strategies at runtime

Test-time Scaling

Train with RL only (no SFT). DeepSeek-R1-Zero showed reasoning can emerge from pure RL

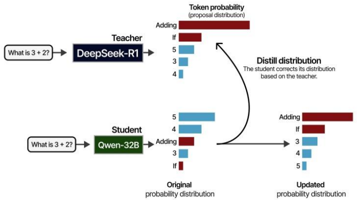
SFT+RL

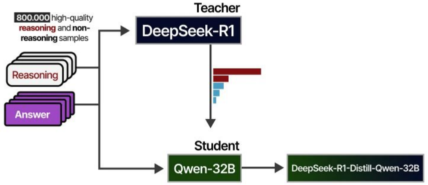
Best approach. Used by DeepSeek-R1 like CoT SFT + RLHF + RLVR.

[The illustrated DeepSeek-R1]()

Distillation (Pure SFT)

Fine-tune smaller models on CoT data generated by larger models. Cheap but effective. Like Qwen on R1 outputs





When to Use Reasoning Models

USE FOR

- Complex multi-step math problems
- Advanced coding challenges
- Tasks requiring verification
- Problems with verifiable answers

DON'T USE FOR

- Simple factual questions
- Summarization, translation, creative writing
- Knowledge-based Q&A
- Tasks where speed matters

Next Class: RAG & Context Management