1 One forward pass and its cost

1.1 The computation that produces one token

A decoder-only transformer maps a token sequence $x_1, \ldots, x_t$ to a probability distribution over x_{t+1}:

$$h_t^{(0)} = W_E[x_t] + (\text{position info}), \tag{1}$$

$$\tilde{h}_t^{(\ell)} = h_t^{(\ell-1)} + \mathrm{Attn}_\ell\big(\mathrm{LN}(h_t^{(\ell-1)});\ \{\mathrm{LN}(h_s^{(\ell-1)})\}_{s\le t}\big), \tag{2}$$

$$h_t^{(\ell)} = \tilde{h}_t^{(\ell)} + \mathrm{FFN}_\ell\big(\mathrm{LN}(\tilde{h}_t^{(\ell)})\big), \qquad \ell = 1, \ldots, L, \tag{3}$$

$$p(x_{t+1} \mid x_{\le t}) = \mathrm{softmax}\big(W_U\ \mathrm{LN}(h_t^{(L)})\big). \tag{4}$$

Two features matter for everything that follows.

(1) The residual stream. Each layer *adds* to the running vector h_t rather than replacing it. It is useful to think of h_t as a shared *bus* or *whiteboard*: attention heads and FFNs read from it and write back to it.

(2) Attention is the only place positions talk to each other. Position t reads from earlier positions $s < t$ through keys and values computed from *their* residual streams at the *previous layer*. In inference these keys and values are cached (the *KV cache*).

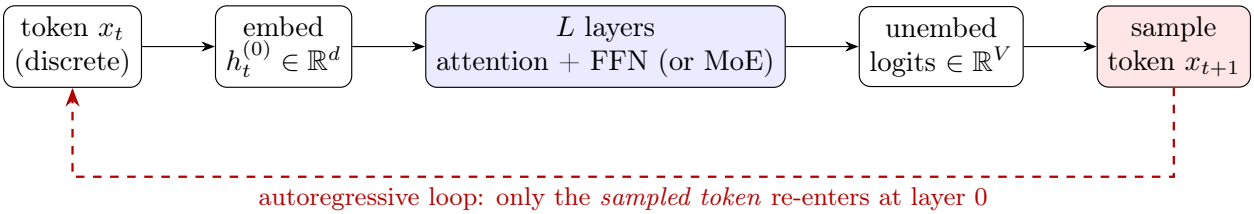


Figure 1: One decoding step. Everything between the embedding and the sampler is a continuous vector computation; the loop back to the input passes through a discrete symbol.

1.2 Compute accounting

Let N_{act} be the number of parameters that participate in processing a token ("active" parameters). A forward pass over one new token costs approximately

$$\text{FLOPs per token} \ \approx\ 2\,N_{\text{act}} \ +\ (\text{attention term} \propto T\,d\,L). \tag{5}$$

For a dense 70B-parameter model, $2N_{\text{act}} \approx 1.4 \times 10^{11}$ FLOPs per generated token. Three consequences:

1. The depth of computation available *in one step* is fixed: L sequential layers, regardless of how hard the question is.
2. Total compute for an answer is roughly (number of tokens processed) × (cost per token). A model that emits 1000 reasoning tokens before answering spends roughly 1000× the compute of one that answers in one token.
3. To change the compute budget, you can change *cost per token* (MoE) or *number of tokens* (CoT). These are independent knobs.

1.3 The token bottleneck

At the top of the stack the model holds a rich vector $h_t^{(L)} \in \mathbb{R}^d$ with, say, $d = 8192$ numbers of 16 bits: nominally $\sim 1.3 \times 10^5$ bits. The sampler turns it into a single symbol out of $V \approx 1.3 \times 10^5$, i.e. at most $\log_2 V \approx 17$ bits. That is a reduction of roughly four orders of magnitude (nominally; the *effective* information in h is of course smaller than its raw bit count, but still far larger than 17 bits).

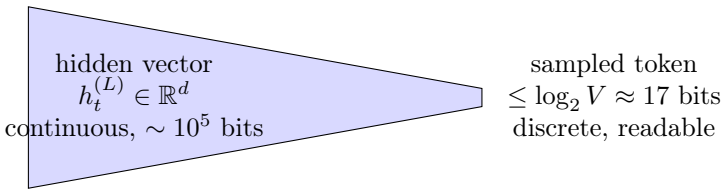


Figure 2: The bottleneck between internal state and the next input. Anything the model "knows" at the top of the stack that is not expressible in the chosen token is, for the purposes of the next step, discarded (but see Section 1.4 for the subtle role of the KV cache).

Why have a discrete bottleneck at all? Four reasons:

- *Training parallelism.* With tokens as the interface, the whole sequence can be processed in parallel during training (teacher forcing). A continuous feedback loop would force sequential training.
- *Error correction.* Snapping to a symbol is like digitizing a signal: small analog noise is erased at every step, so errors do not accumulate smoothly.
- *Shared language with humans and data.* The token interface lets us pretrain on text and read the output.
- *Sampling.* Discretization lets the model commit to one of several possibilities and explore (temperature, best-of-n, self-consistency).

1.4 What is carried from one step to the next?

This is the point at the heart of the "observable vs. non-observable" distinction, and it is worth being precise. There are exactly two channels from step t to step $t+1$:

1. **The sampled token**, which re-enters at layer 0 as an embedding.
2. **The KV cache**: the keys and values at *every layer* for position t, computed from the hidden vectors $h_t^{(0)}, \ldots, h_t^{(L-1)}$.

Notice what is *not* carried: the *top-of-stack* computation $h_t^{(L)}$ has no path to later positions except via the token. Information at layer ℓ of position t can reach later positions only at layers $\geq \ell + 1$ (through attention). So information flows *upward in layers* and *forward in positions*, but never downward.

Takeaways

(a) The "current state" is the vector $h_t^{(\ell)}$, not the token. The token is a lossy, discrete projection of the top-layer vector. (b) Per-token compute is fixed; depth per step is L. (c) Cost $\approx$ (tokens) $\times$ (active parameters). (d) The only way to feed top-layer results back to the bottom is through the sampled token.

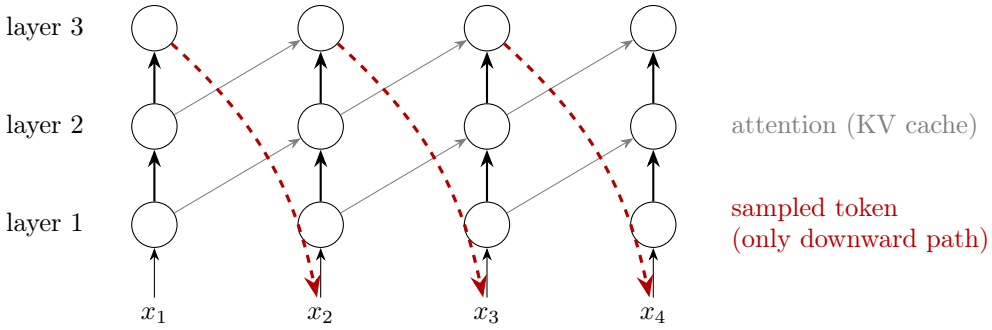


Figure 3: Information flow across positions. Gray arrows (attention through the KV cache) always go *up* one layer as they go forward in position. The red dashed arrows, the sampled token, are the only way for a top-layer result to re-enter at the bottom. This is why, in a standard transformer, serial depth beyond L *requires* emitting tokens.

2 What lives in the vector

We do not fully understand what is encoded in $h_t^{(\ell)}$. What follows is a mix of established empirical findings, working hypotheses, and analogies. We flag which is which.

2.1 Geometry: directions as features

Linear representation hypothesis (a working hypothesis, with substantial support): many human-meaningful properties—"this token is a French word", "the subject is plural", "the statement is true", "the previous move was by white"—correspond to *directions* in activation space, and the intensity of the property is the magnitude along that direction. The classic example is $\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$.

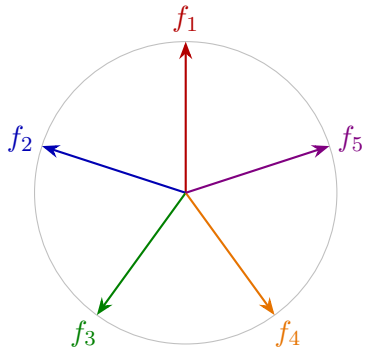
Evidence that states are richer than surface text:

- A transformer trained only to predict legal Othello moves develops a linear-decodable representation of the *board state*. The model never saw a board.
- Mid-layer representations of text in different languages appear to share concepts; logit-lens analyses of Llama models found English-like intermediate token predictions even for non-English input, and circuit-tracing studies report language-independent concept features that feed language-specific output features.
- When completing a rhyming couplet, the model appears to select the rhyme word *before* writing the line leading to it, i.e. it plans ahead in its hidden state.
- For a multi-hop question ("the capital of the state containing Dallas"), an intermediate feature for the bridge entity ("Texas") is active inside the network even though it never appears in the output.

2.2 Superposition: more features than dimensions

A vector in $\mathbb{R}^d$ has only d orthogonal directions, but a model needs to represent far more than d concepts. In high dimensions one can pack many more than d directions that are *almost* orthogonal. The model exploits this: features are stored *in superposition*, sharing dimensions, and relies on

sparsity (few features are active at once) to keep interference tolerable.



Five features stored in $d = 2$. Adjacent directions have $\cos 72^\circ \approx 0.31$: small but non-zero *interference*. Reading f_1 off the plane also picks up $0.31\,(f_2{+}f_5)$. Works well only if features are rarely active together.

Figure 4: Toy picture of superposition: more features than dimensions at the price of interference.

Consequence for interpretability: individual neurons are often *polysemantic* (respond to several unrelated things) because the natural basis of features is not the coordinate basis. Sparse autoencoders try to recover the feature basis.

2.3 Is there an "internal language"?

Informally people speak of an internal "language" (sometimes *neuralese*). A careful version of the claim:

- The vectors form a *code*: a mapping between states of the world/text and points in $\mathbb{R}^d$, learned only because it is useful for next-token prediction.
- It is *not* a language in the human sense: no grammar is known, features are superposed, and the code is not unique (any invertible linear transform of the residual stream, applied consistently to all readers and writers, gives an equivalent network).
- It is *partly* language-independent and concept-like in the middle layers, and becomes increasingly token-specific near the input and the output.

2.4 Analogies (and where each breaks)

Analogy	What it captures	Where it breaks
Thought vs. spoken word	The hidden vector is like your full mental state; the token is the one word you say. Much of what you "mean" is not in the word.	Humans have introspective access and a persistent state; the model's state is recomputed from scratch each step (apart from the KV cache).
Mixing console	Each of the d coordinates (or, better, each direction) is a slider; a "thought" is a particular combination of slider settings; layers nudge sliders.	Real mixing consoles have fixed, labeled sliders; here the labeled axes are not the coordinate axes.
Radio spectrum	Many stations (features) share one cable (the vector), separated by distinct frequencies (directions); sparse use limits cross-talk. This is superposition.	Frequencies are orthogonal by design; here interference is real and the model must tolerate it.
Shared whiteboard	Layers read and write notes on the residual stream; later layers rely on earlier notes ("Texas" $\rightarrow$ "Austin").	The whiteboard has no human-readable labels, and notes are overwritten implicitly by cancellation.
Chess intuition	A grandmaster "sees" the position holistically and announces a move; the justification (if requested) is reconstructed afterward. Hidden state $\neq$ the announced move.	Humans can introspect and reexamine; the model's explanation is just more generated text.

Takeaways

(a) Hidden vectors encode far more than the next token: world state, plans, intermediate entities. (b) Features are directions, stored in superposition, so neurons are not the right unit. (c) The "internal language" is a learned, non-unique code; useful analogies exist but each one fails somewhere. (d) Much of this is empirical and partial; we read these vectors through noisy instruments.

3 Mixture of Experts

3.1 Motivation: parameters vs. compute

In a dense transformer, every parameter participates in every token. Parameters buy *capacity* (knowledge, skills) and FLOPs-per-token buy *thinking*; in a dense model they are the same quantity, $N_{\text{act}} = N$. **Conditional computation** breaks this link: have many parameters but use only a small subset for each token.

Why this is plausible. Two empirical observations suggest sparse activation is natural for FFNs:

- FFN layers behave like key–value memories: each hidden unit's input weights act as a "key" matching some input pattern and its output weights as the "value" written when matched.
- In trained dense transformers, the FFN hidden activations are already extremely sparse: for any given token most hidden units are essentially inactive.

So dense FFNs spend FLOPs multiplying by weights whose contribution is near zero. MoE makes the sparsity *structured* (whole blocks of weights are skipped), so that hardware can actually exploit it.

3.2 The MoE layer

In the standard design the attention sublayers, embeddings and unembedding stay dense. Only the FFN sublayer is replaced by N expert FFNs $E_1, \ldots, E_N$ and a router:

$$s(x) = W_r\, x \in \mathbb{R}^N, \qquad \mathcal{S}(x) = \mathrm{TopK}\big(s(x),\, k\big), \tag{6}$$

$$g_i(x) = \frac{\exp(s_i(x))}{\sum_{j\in\mathcal{S}(x)} \exp(s_j(x))} \quad (i \in \mathcal{S}(x)), \tag{7}$$

$$\mathrm{MoE}(x) = \sum_{i\in\mathcal{S}(x)} g_i(x)\, E_i(x). \tag{8}$$

The routing happens *per token and per layer*.

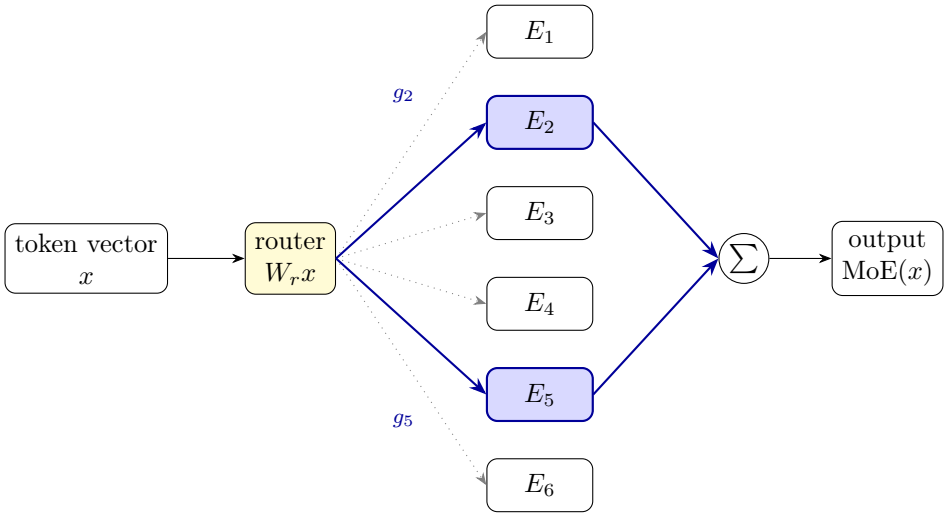


Figure 5: An MoE layer with $N = 6$ experts and top-$k = 2$ routing. Only E_2 and E_5 run for this token; their outputs are mixed with weights g_2, g_5.

3.3 Why it helps computationally

FLOPs scale with k, not N. If each expert is the size of the dense FFN it replaces, the FFN parameter count grows by $N\times$ but FFN FLOPs grow only by $k\times$. Real examples:

Model	Experts (routed / chosen)	Total params	Active params / token
Switch Transformer (largest)	2048 / top-1	∼1.6T	(small fraction)
Mixtral 8×7B	8 / top-2	∼46.7B	∼12.9B
DeepSeek-V3	256 + 1 shared / top-8	∼671B	∼37B
Dense baseline	—	N	N

DeepSeek-V3 uses only $\approx 37/671 \approx 5.5\%$ of its parameters per token; Mixtral about 28%.

Combinatorial capacity. A token chooses its experts independently in each of the L layers, so the number of distinct computational *paths* is $\binom{N}{k}^L$. For Mixtral-like $N = 8, k = 2, L = 32$ this is $28^{32} \approx 10^{46}$. Fine-grained designs (*DeepSeekMoE*: many small experts plus always-on "shared" experts) push this further: choosing 8 of 64 small experts gives $\binom{64}{8} \approx 4.4 \times 10^9$ combinations per layer at the *same* FLOPs as a dense FFN, versus 28 for 2-of-8.

Scaling. Empirically, routed models follow scaling laws in which, at fixed active compute, more experts keep improving loss, though with diminishing returns. The practical interpretation: MoE lets you spend training and inference FLOPs on "thinking" and spend memory on "knowing".

3.4 What MoE does *not* save

- **Memory footprint.** All N experts must be resident (usually sharded across devices). MoE trades *memory* for *FLOPs*.
- **Memory bandwidth at realistic batch sizes.** At batch size 1 only k experts per layer are read from memory, a genuine bandwidth saving. But with a batch of B tokens, the expected fraction of experts touched in one layer is

$$1 - \left(1 - \tfrac{k}{N}\right)^B. \tag{9}$$

  For Mixtral-like $k/N = 1/4$ and $B = 8$: $\approx 90\%$. For DeepSeek-like $k/N = 1/32$ and $B = 64$: $\approx 87\%$. At even moderate batch sizes, almost every expert's weights are read anyway, so the saving is mostly in FLOPs, and serving needs careful sharding (*expert parallelism*) with all-to-all communication.
- **Attention cost and KV cache**, which stay dense.

3.5 Training issues: routing collapse and load balancing

The top-k selection is not differentiable, but the gate values g_i are, so the router learns from gradients flowing through the selected experts' outputs. A well-known failure mode is *routing collapse*: early in training, a few experts do slightly better, receive more tokens, improve further, and the rest starve.

Standard remedies:

- **Auxiliary load-balancing loss.** With f_i the fraction of tokens dispatched to expert i and P_i the mean router probability for expert i: $\mathcal{L}_{\text{bal}} = \alpha\, N \sum_{i=1}^{N} f_i P_i$, minimized by uniform routing.
- **Capacity factor and token dropping**: cap the number of tokens per expert; overflow tokens skip the layer via the residual connection.
- **Noise in the router**, **router z-loss** for numerical stability, and **expert-choice routing** (experts pick tokens rather than tokens picking experts).
- **Auxiliary-loss-free balancing**: a per-expert bias on the routing score adjusted online, avoiding interference of the balance loss with the main objective.

3.6 What do experts actually specialize in?

The marketing picture is "a math expert, a law expert, a French expert." The evidence is more modest. Analyses of Mixtral found routing correlates strongly with *syntactic and positional* properties and with local token identity, and little with the *domain* of the document. A better mental model:

> **Analogy: hospital triage, with a twist**
> A hospital has many specialists, and a triage nurse (the router) sends each patient to two of them. Capacity grows with the number of specialists; the time per patient does not. The twist: here the "patients" are individual *tokens at individual layers*, and the specialists are not organized by medical specialty but by whatever partition of the work was easiest to learn. A single sentence may visit many different experts at different layers. Expect experts that are "good at punctuation-in-code" or "good at numeric tokens" more often than "the biology expert".

3.7 MoE and the hidden state

The router decision $\mathcal{S}(x)$ is itself a *discrete internal variable*, computed from the hidden vector and invisible in the output. It is another channel of non-observable computation: two inputs with nearly identical hidden states can take different paths, and the path taken is evidence about what the network "decided" at that layer. Some interpretability work uses router statistics as a window; it is also a source of non-smoothness (small changes in x can flip the chosen expert).

3.8 Summary

(a) MoE decouples *total* parameters from *active* parameters; compute per token depends on k, capacity on N. (b) Only the FFN is sparsified; attention is dense; routing is per token per layer. (c) The savings are FLOPs and (at small batch) bandwidth, not memory. (d) Load balancing is the central training difficulty. (e) Experts specialize in token/syntax patterns, not human-labeled topics. (f) Cheaper tokens mean more tokens per dollar.

4 Chain-of-thought

4.1 The phenomenon

Prompting a model to write intermediate steps before answering dramatically improves accuracy on arithmetic, symbolic and multi-step problems. Even the instruction "let's think step by step" helps, and sampling several chains and voting improves further. More recently, models are *trained* with reinforcement learning to produce long reasoning traces before answering, and accuracy increases with the number of reasoning tokens allowed ("test-time compute scaling").

4.2 Why it works: five complementary explanations

H1. Extra serial computation (the depth argument). From section1: in one step, the network can compose at most L sequential operations, and the *only* way to run more is to pass through

the token. A chain of m generated tokens gives up to $\approx m \cdot L$ sequential layers of computation. Formalizing this:

- Constant-depth transformers with (log-)precision arithmetic fall within the circuit class TC^0. Problems believed outside TC^0 (for example composing a long sequence of permutations, or evaluating Boolean formulas) cannot be solved by a fixed-depth forward pass, under standard complexity conjectures ($\mathsf{TC}^0 \neq \mathsf{NC}^1$), unless width or depth grows with input length.
- With CoT, expressive power grows with the number of steps: T steps of CoT let a constant-depth transformer simulate circuits of size $\approx T$; polynomially many steps raise it to all of P, while a linear number of steps is a strictly more limited regime. Arithmetic and equation-solving tasks that need super-polynomial size without CoT become tractable with CoT of modest length.

Caveat. These are expressivity results: they say CoT *can* help, not that SGD finds such solutions.

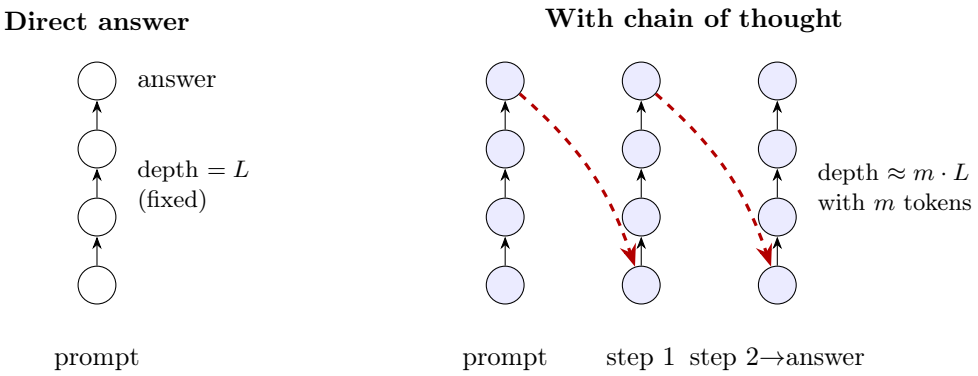


Figure 6: Unrolling serial computation through the token loop. Each emitted token resets the stack at layer 0 but carries a result forward (red).

H2. External working memory. A fixed-width vector holds only so much. Writing intermediate results as tokens stores them in the context, where later steps can attend to them. Think of long multiplication: you can multiply 4-digit numbers in your head with great effort, but paper is a dramatic improvement, not because paper computes but because it stores partial results and frees working memory. Related: tracking state across many operations (swaps, variable assignments) is a state-tracking problem, which also lies outside constant-depth parallel computation.

H3. Decomposition into easy, in-distribution steps (a learnability argument). Even if a task is expressible without CoT, a direct mapping from question to answer may be unlearnable from finite data, while each *local* step of a derivation is simple and well-covered in training text. Experiments with synthetic data support the claim that reasoning emerges when training data has local structure linking variables through intermediates.

H4. Priming and retrieval. Generating relevant facts and definitions puts them in context, effectively performing retrieval from the model's own parameters before the answer is committed. Writing "Dallas is in Texas; the capital of Texas is Austin" gives the final step what it needs as explicit tokens, where otherwise the bridge entity would have to live only in the hidden state.

H5. Search, verification and self-correction (the trained-reasoner story). With reinforcement learning on verifiable rewards, long traces include backtracking, checking and alternatives. The trace functions as an explicit search procedure that the model learned to run, and "more thinking tokens" means more search budget.

Evidence that content is not the whole story: filler tokens

On certain synthetic tasks, models trained to emit meaningless filler tokens ("`.....`") before answering outperform those that answer immediately. That supports H1: extra positions provide extra parallel hidden computation even without semantic content. (It does not show frontier models use filler this way in practice.)

4.3 What CoT costs, and why tokens are the interface

- **Compute**: m reasoning tokens cost $\approx m$ forward passes of active parameters (plus growing attention and KV cache). This is why MoE and CoT are complementary: MoE lowers cost per token, so a fixed budget funds longer chains.
- **Latency**: decoding is sequential; m tokens means m dependent steps.
- **Bandwidth bottleneck**: each reasoning step squeezes $\lesssim 17$ bits out of a $\sim 10^5$-bit state (Fig. 2). The model must choose which small part of its thinking to "say". Verbose chains are partly a response to this narrow channel.

4.4 Beyond tokens: latent reasoning

Since the bottleneck discards the top-layer vector, an obvious idea is to feed it back directly. **Coconut** (continuous thought) appends the final hidden state as the next input embedding for some steps, instead of sampling a token. Reported benefits: the continuous thought can encode several candidate continuations at once (superposition; behavior like breadth-first search in some graph-reasoning tasks). **Recurrent-depth** models iterate a shared block a variable number of times before decoding.

Trade-offs to discuss:

	Token CoT	Latent (continuous) reasoning
Bandwidth per step	$\leq \log_2 V$ bits	$\sim d$ floats
Training	Parallel over positions (teacher forcing)	Sequential through the loop (backprop through time), costly
Error correction	Discretization snaps noise away	Noise and drift can accumulate
Readability	Human-readable (with caveats)	Opaque
Monitoring / safety	Possible to inspect traces	Hard to inspect

Takeaways

(a) In a standard transformer, serial depth beyond L *requires* emitting tokens; CoT unrolls computation through the token loop. (b) Theory: constant-depth is limited (TC^0); CoT with m steps expands what is computable. (c) CoT also acts as working memory, decomposition, retrieval and search. (d) Cost is linear in tokens; MoE makes tokens cheaper. (e) The discrete bottleneck buys training parallelism and robustness but throws away most of the hidden state; latent reasoning removes the bottleneck at a price.

5 Observable vs. non-observable state

5.1 An inventory of what an observer can see

Quantity	API user	Open-weights user	Comment
Prompt and sampled tokens	yes	yes	The only things *both* sides always see.
Token log-probabilities	sometimes	yes	A view of the output distribution, not the state.
Visible reasoning text	depends on provider	yes	Readable, but is it faithful? (below)
Residual-stream vectors $h_t^{(\ell)}$	no	yes	Available but uninterpreted.
Attention patterns, KV cache	no	yes	Shows *where* information is read, not *what*.
MoE router choices	no	yes	Discrete internal decisions.

Key distinction: *observable* (can be read out) is not *understood* (we know what it means). Even with full access to $h_t^{(\ell)}$, we need tools to interpret it.

5.2 Tools for reading hidden vectors

Logit lens and tuned lens. Apply the unembedding to an intermediate vector: $\text{softmax}\left(W_U\ \text{LN}(h_t^{(\ell)})\right)$. This shows what the model would predict if it stopped at layer ℓ, and often reveals a gradual refinement across layers. Caveat: intermediate layers need not use the output basis.

Linear probes. Train a small classifier $h \mapsto$ property (board square, truth value, sentiment). If a linear probe succeeds, the property is linearly decodable. Caveat: decodable $\neq$ used by the model; probes can also learn the property themselves.

Causal interventions. Activation patching and steering: replace or add a direction in h and observe the change in output. This tests whether the model *uses* the feature. (Steering a feature in a large model's residual stream produces correspondingly targeted behavior changes.)

Sparse autoencoders (SAEs). Learn an overcomplete dictionary so that $h \approx Dz$ with $z \geq 0$ sparse:

$$\min_{D,E}\ \left\|h - D\,z\right\|_2^2 + \lambda\|z\|_1, \qquad z = \text{ReLU}(Eh + b). \tag{10}$$

Each column of D is a candidate *feature direction*, often much more interpretable than individual neurons. This directly targets the superposition problem of section 2.

Circuit tracing / attribution graphs. Replace parts of the model with interpretable approximations and trace how features influence each other across layers for a given prompt; applied to planning, multi-step inference, and multilingual behavior.

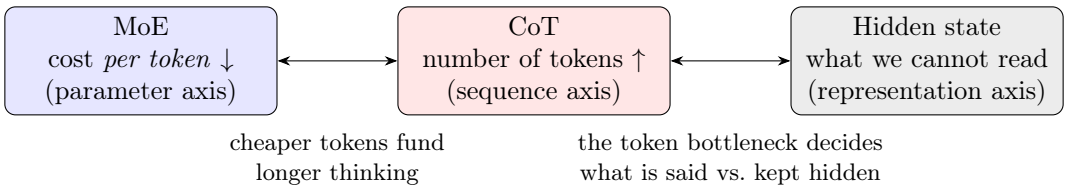
5.3 Is the visible chain of thought a faithful report?

The CoT text is observable, and *it also causally affects later computation* since later tokens attend to it. But that does not mean it is a complete or honest account of the computation that produced the answer.

- Models given a biasing hint (answer in the prompt, a previous answer choice) often change their answer but write a CoT that never mentions the hint.
- Truncating or corrupting a CoT changes the final answer to varying degrees across tasks and model sizes, suggesting that in some settings the trace is used, in others it is largely post hoc.
- Reasoning models verbalize a hint they used only a minority of the time in some evaluations, and outcome-based RL did not make this much more faithful.
- Training *against* a CoT monitor can lead to models that hide their intent in the CoT while continuing to misbehave.
- Nevertheless, the CoT is currently a useful and relatively cheap window; the community argues for preserving its monitorability, noting it is a fragile property and may erode under training pressure or architectural change (latent reasoning).

Think of CoT as a *partial, causally-relevant, but non-guaranteed* readout. Two layers of hidden state sit beneath it: (1) the hidden vectors at each token (continuous, superposed, not human-readable), and (2) discrete internal choices such as MoE routing. A reasoning trace is the portion of the computation the model chose (or learned) to externalize, and nothing forces it to externalize everything relevant.

6 Summary



A transformer computes with continuous vectors that encode far more than the next word; it can only pass the result of a full-depth computation forward by committing to a discrete token. Mixture of Experts lets us make each such computation cheap by touching only a few experts, leaving capacity large. Chain-of-thought lets us chain many such computations, buying serial depth, working memory, decomposition and search at the price of one forward pass per token. The chain we read is the externalized, partial, sometimes unfaithful tip of a computation whose bulk lives in vectors and routing decisions we are still learning to read.