Exercise: What Could the Model Have Said?

This exercise uses the LLM Token Prediction Explorer, a custom interface connected to a live language model (TinyLlama-1.1B). You'll work through two modes — the Next-Token Predictor and the Full Sentence Predictor — to see what's actually happening when a language model "writes."

Part 1: The Model Doesn't Pick a Word — It Produces a Distribution

Open the **Next-Token Predictor** from the hamburger menu. Make sure the left and right columns both have their controls set to the defaults: Temperature 1.00, Top-K 50.

In the prompt field, type:

> The most important thing about education is

Hit **Predict →** and look at the results in the left column.

You're seeing the model's top 10 candidates for the next word, ranked by probability. Notice that none of them is "the answer." The model hasn't decided what comes next — it's produced a spread of options with different weights. Some are more likely than others, but several of them would produce a reasonable sentence.

Write down: What are the top 3 candidates? How close are their probabilities? Could you argue that any of them is the "right" next word?

Now try a different prompt: The

> capital of France is

Compare this distribution to the first one. You should see a much more concentrated result — one candidate dominates. The model is more "certain" here, but it's the same mechanism: a distribution over candidates, not a decision.

Takeaway: The model's actual output at each step is not a word. It's a probability distribution — a ranked list of everything it could say, weighted by likelihood. The word that ends up in the text is selected *from* this distribution, not generated by some deeper process of understanding.

Part 2: The Distribution Is Shapeable

Stay in the **Next-Token Predictor**. Go back to the first prompt: The most

> important thing about education is

Now change the **left column** to Temperature **0.1** and leave the **right column** at Temperature **1.0**
Hit Predict →.

model, given the same input, produces a distribution that looks completely different depending on this one parameter.

Now try the reverse: set the left column to Temperature **1.8** and the right to **0.1**. Hit Predict → again. At high temperature, the distribution is nearly flat — the model is almost equally likely to say any of its top candidates. At low temperature, it's locked onto one.

The point is not that temperature is a useful parameter. The point is that the distribution you saw in Part 1 — the one that felt like it represented the model's "opinion" about what comes next
— is not fixed. It's malleable. A single number controlled by the system's designers reshapes what the model is likely to say. The model doesn't become smarter at low temperature or more creative at high temperature. The same underlying weights produce the same raw scores.
Temperature just decides how sharply the system discriminates between them.

Now reset both columns to Temperature 1.0 and try adjusting **Top-K**. Set the left column to Top-K **5** and the right to Top-K **50**. Hit Predict →.

Top-K works differently — it doesn't reshape the distribution, it truncates it. With Top-K 5, only the top 5 candidates exist. Everything else is eliminated, no matter how reasonable it might have been. This is a hard cutoff imposed by the designer.

Takeaway: The distribution isn't just a fact about the model — it's something that gets sculpted by parameters that someone chose. Temperature and Top-K are design decisions that determine which words survive into the final output. The user never sees these controls, but they shape every word the model produces.

Part 3: Chaining — One Distribution AfterAnother

Switch to the **Full Sentence Predictor** from the hamburger menu. Set Tokens to Generate to **15**
Temperature to **1.0**, and Top-K to **50**

In the prompt field, type: A

> university should

Hit **Generate →** and watch the sentence appear word by word.

Each word is color-coded by confidence: darker, warmer colors mean the model was more confident; lighter, cooler colors mean it was less sure. Look at the Token Log below the generated text — it shows the probability of each word that was selected.

Notice the uneven terrain. Some words appear with high confidence (common function words like "the," "to," "and"). Others appear with much lower confidence — those are positions where the model could easily have gone a different direction. The sentence reads as coherent prose, but the confidence coloring reveals that the model was more uncertain at some positions than others.

whole sentence.

Pick a word where the model showed low confidence (lighter color). Look at its alternatives. There may be several candidates that were nearly as likely as the one that was chosen. The sentence you're reading is the result of the model having picked *this* word rather than *that* one at every step, and at many of those steps, the choice was close.

Takeaway: A generated sentence is a chain of selections from distributions. It looks like a single coherent output, but it's built from a sequence of probabilistic choices, each of which could have gone differently. The coherent surface is an effect of chaining, not evidence of a plan.

Part 4: Branching — The Sentence Didn't Have to Be This

Stay in the **Full Sentence Predictor**. Generate a fresh sentence with the same prompt: A university

> should

This time, find a word early in the sentence (the 2nd, 3rd, or 4th generated word) where the model shows moderate-to-low confidence. Click on it to open the alternatives panel. Now **click one of the alternative words**

The tool will branch the sentence: it keeps everything up to the word you clicked, substitutes the alternative you chose, and then generates the rest of the sentence forward from that new starting point. You'll see the original sentence with the branch point marked, and a new sentence growing underneath it from the word you selected.

Read both sentences. They share a common beginning but diverge — sometimes subtly, sometimes dramatically — from the point where you made a different choice. The model didn't "want" to say either of these things. Both are equally valid paths through the tree of possible outputs. The one that would have appeared in a chat interface is whichever one happened to get sampled first.

Try this several times. Branch from different positions. Branch early and watch the sentences diverge widely. Branch late and notice they may stay more similar. Try branching from a high-confidence word where one candidate dominated — the branch might not change much because the model reconverges on similar territory. Then branch from a low-confidence position and watch the output shift dramatically.

If you want to push this further, lower the temperature to **0.2** and generate again. The sentence will be more predictable and the color-coding will be uniformly high-confidence. Branching here produces less divergence because the distributions are so concentrated. Now raise it to **1.5** and generate again. More color variation, more viable alternatives at each step, and more dramatic branching. The "tree" of possible sentences is wider at high temperature and narrower at low temperature — but it's always a tree, never a single inevitable line.

sentence unfold. Nothing about the model's architecture produces a single intended output. It produces a field of possibilities, and the text you see is the result of a series of selections that could have gone otherwise.

Connecting the Two Modes

The Next-Token Predictor and the Full Sentence Predictor show the same mechanism at two different scales:

The Next-Token Predictor isolates a single moment — one position in the text, one distribution of candidates. It lets you see that the model doesn't have "an answer," it has a spread of possibilities. And it lets you see that this spread is shaped by parameters that someone chose.

The Full Sentence Predictor shows what happens when you chain those moments together: you get text that reads as coherent prose, but is built from a sequence of contingent selections. The branching lets you go back and re-make those selections, revealing that the sentence didn't have to be this sentence.

The gap between these two views — between the messy, probabilistic, shapeable process and the clean prose that comes out the other end — is exactly the gap that commercial AI interfaces hide. When you interact with ChatGPT or Claude, you see the clean prose. You don't see the distributions, the alternatives, the parameters, or the contingency. The interface presents the output as if the model composed it with intention. What you've seen in this exercise is the machinery underneath: weighted uncertainty, selection, and design decisions at every stage.