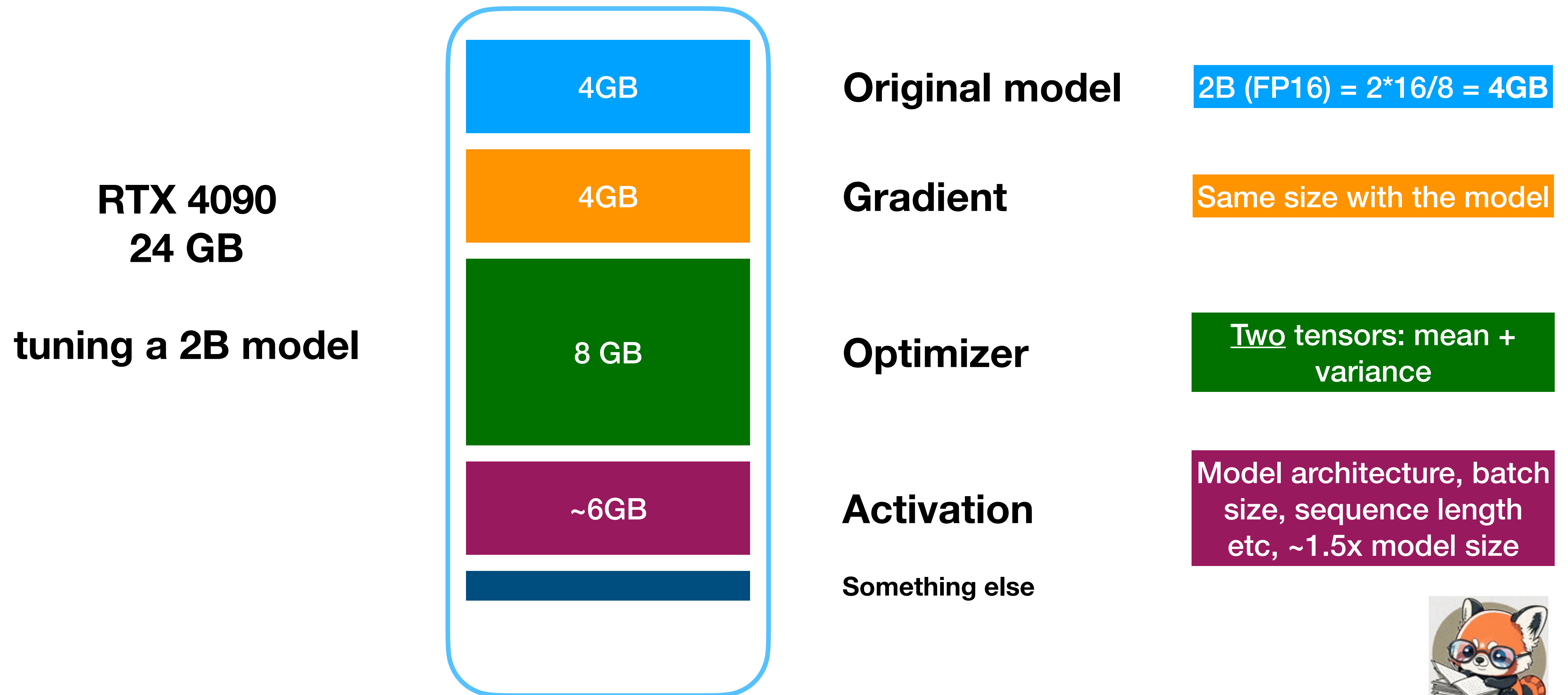


# Fine tune a LLM: how much memory do I need?



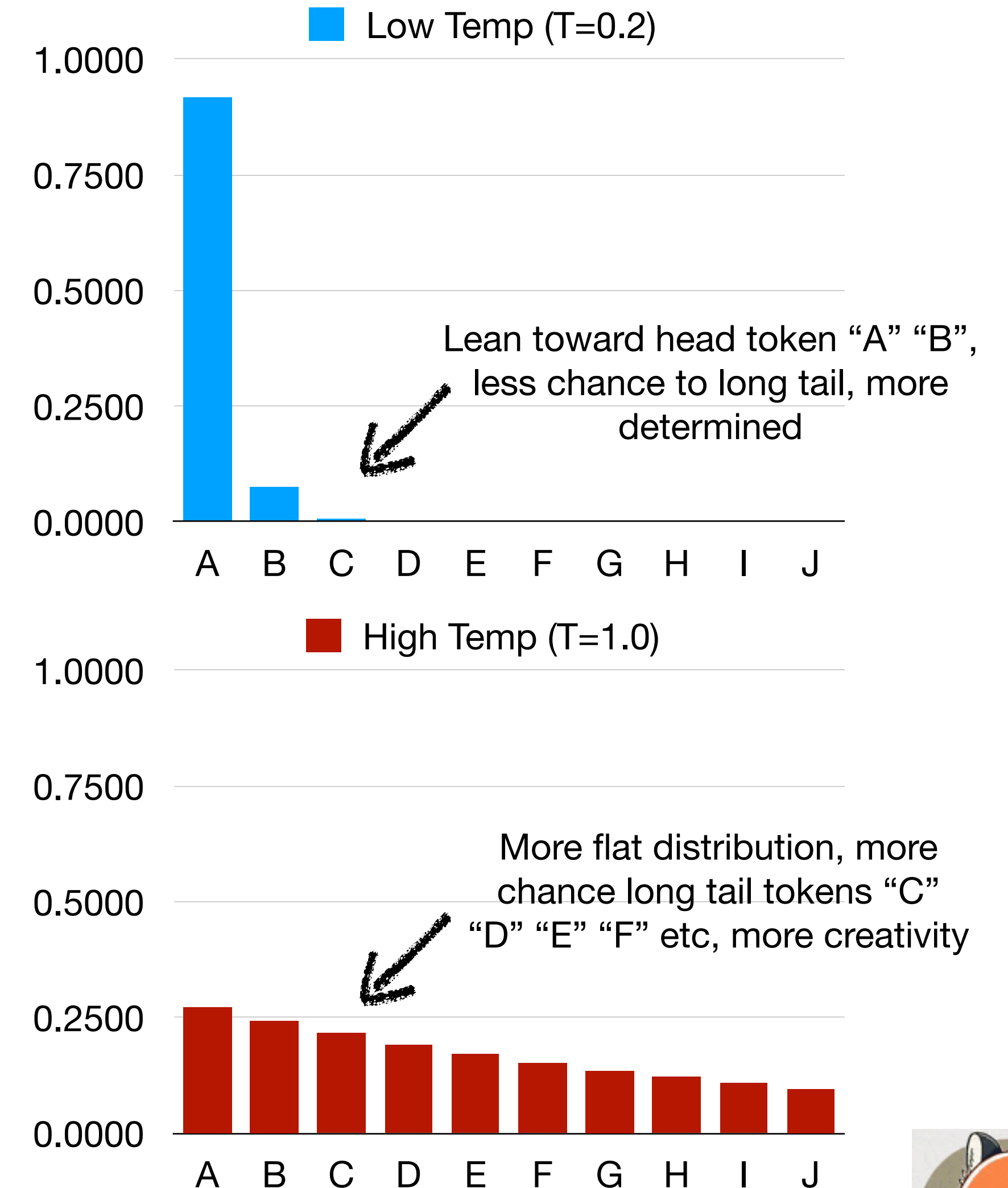
# Understand temperature in LLM

```
response = openai.ChatCompletion.create(  
    model='gpt-4o',  
    temperature=0.7,  
    max_tokens=30,  
    messages=[  
        {'role': 'user',  
         'content': question  
    }],  
)
```

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_{j=1}^N \exp(z_j/T)}$$

T = 0.2

T = 1.0



Softmax with Temperature

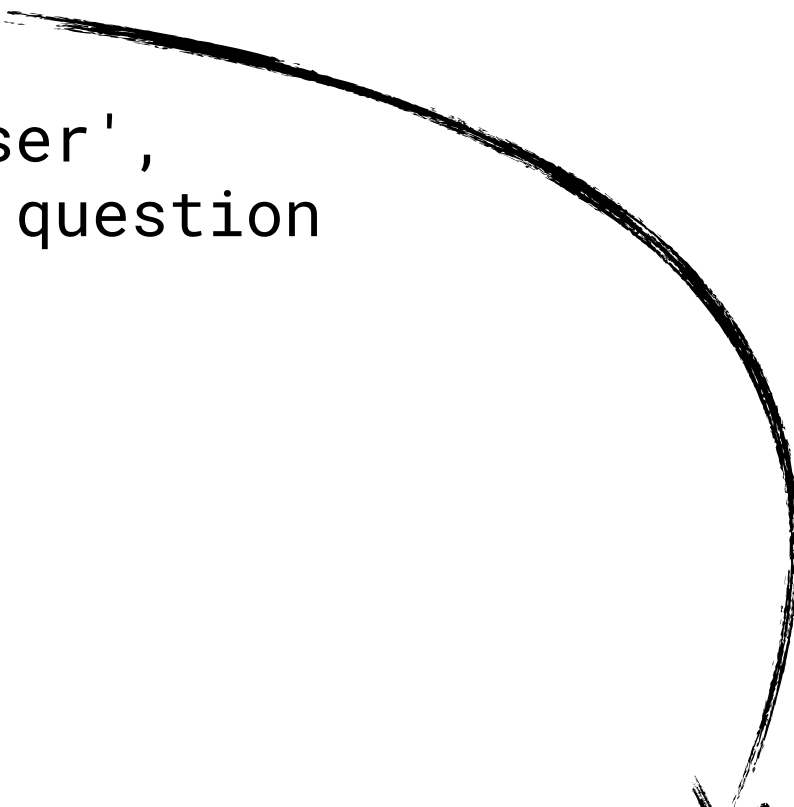
Hinton et al "Distilling the Knowledge in a Neural Network" 2015

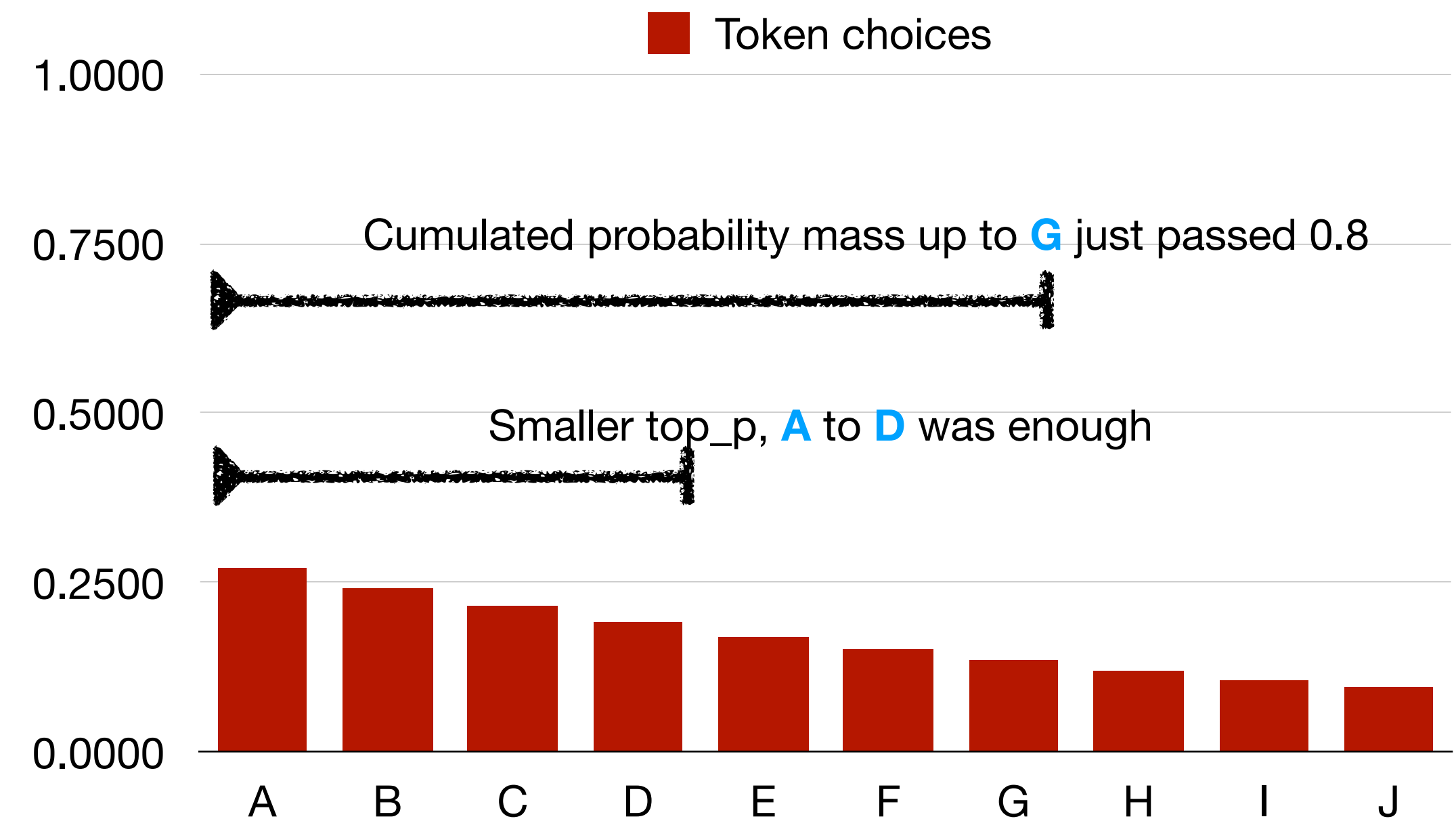
<https://arxiv.org/abs/1503.02531>



# Understand **top\_p** in LLM

```
response = openai.ChatCompletion.create(  
    model='gpt-4o',  
    max_tokens=30,  
    top_p=0.8,  
    messages=[  
        {'role': 'user',  
         'content': question  
        }],  
)
```

$$\sum_{x \in V(p)} P(x|x_{1:i-1}) \geq p$$




More flexible than top K to the shape of the probability distribution, across different contexts.

Top P or "Nucleus sampling"

Holtzman et al, "The Curious Case of Neural Text Degeneration" 2019

<https://arxiv.org/abs/1904.09751>

<https://www.linkedin.com/in/liuhongliang/>



# Understand Boltzmann distribution and neural networks

## What computer scientists see

Probability of class  $i$  given input  $\mathbf{x}$  in a neural network

$$P(\text{class } i | \mathbf{x}) = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

logits (pre-softmax activations)

interpret neural network input as probabilities instead of numbers

Optimization and generalization can make statistical sense.

## What physicists see

Probability of a certain state

$$P(\mathbf{s}) = \frac{1}{Z} \exp \left( -\frac{E(\mathbf{s}; \boldsymbol{\theta})}{T} \right)$$

Energy of the state

Partition function (normalization)

Temperature parameter



# Understand prompting

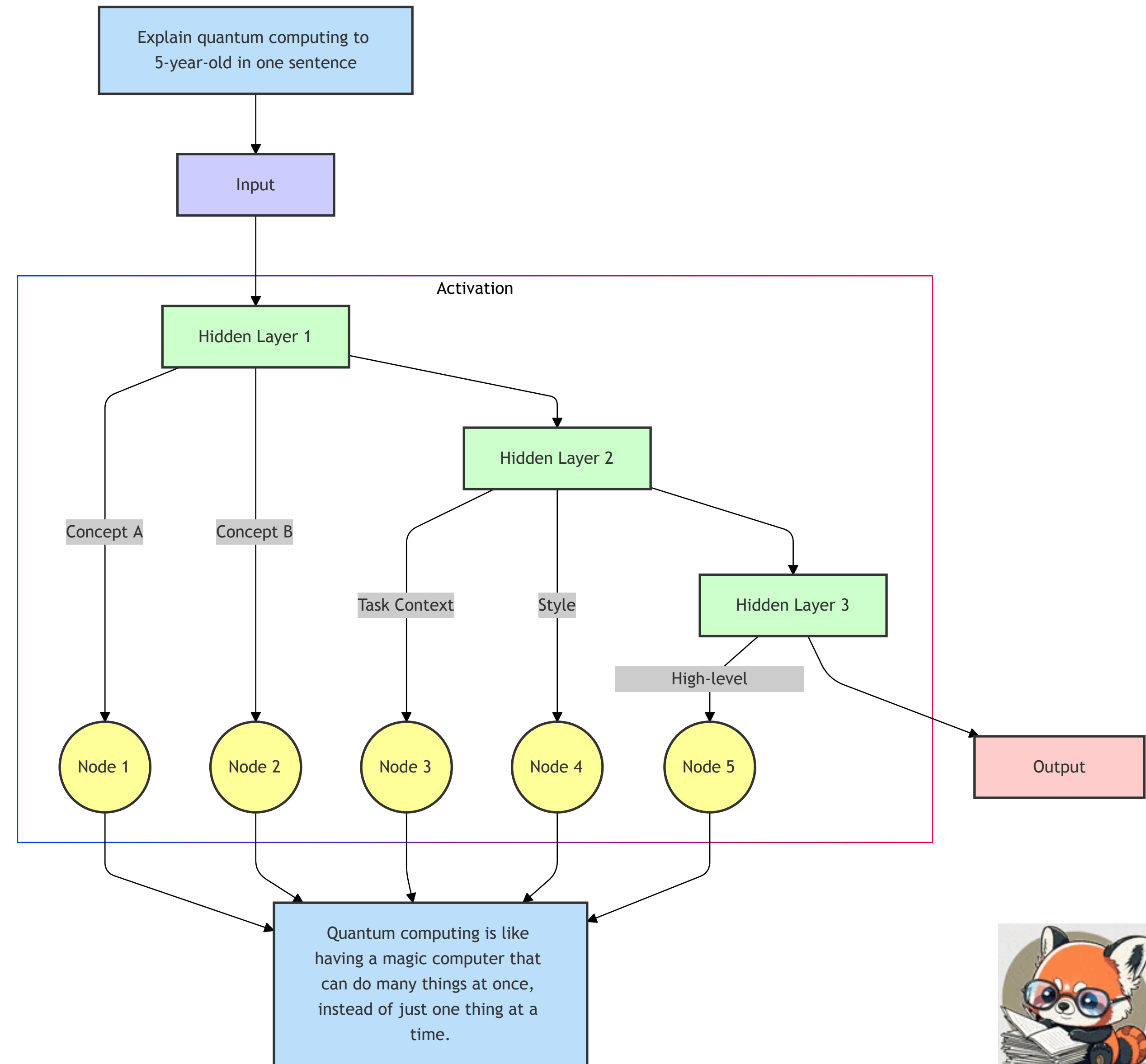
Prompting introduces a **task-specific bias** to the model's output distribution by its **activation patterns**, effectively **aligning the target task** with the LLM's pre-trained task manifold.

$$P(y_t | \text{prompt}, x) = \text{softmax}(f(h_L + \beta \nabla_{\text{task}}) / \tau)$$

$\nabla_{\text{task}} = \frac{\partial \mathcal{L}_{\text{task}}}{\partial h_L}$  Task gradient

$h_L = g_L(\dots g_2(g_1([E_{\text{prompt}}; E_x])))$  Hidden states

$\beta = \gamma \cdot \text{sim}(h_L, h_{\text{task}})$  Alignment strength





# Understand tool using

LLMs translate language tasks into tool actions by computing  $P(\text{tool}|\text{context})$  through **attention-based alignment** between task requirements and tool capabilities.

$$P(\text{action}|\text{state}) = \int P(\text{action}|\text{intention})P(\text{intention}|\text{state})dI$$

Connect state observation to action selection

$$P(\text{tool}|\text{context}) = \text{softmax}\left(\frac{h_{\text{tool}}^{\text{ctx}} W_{\text{out}}}{\tau}\right)$$

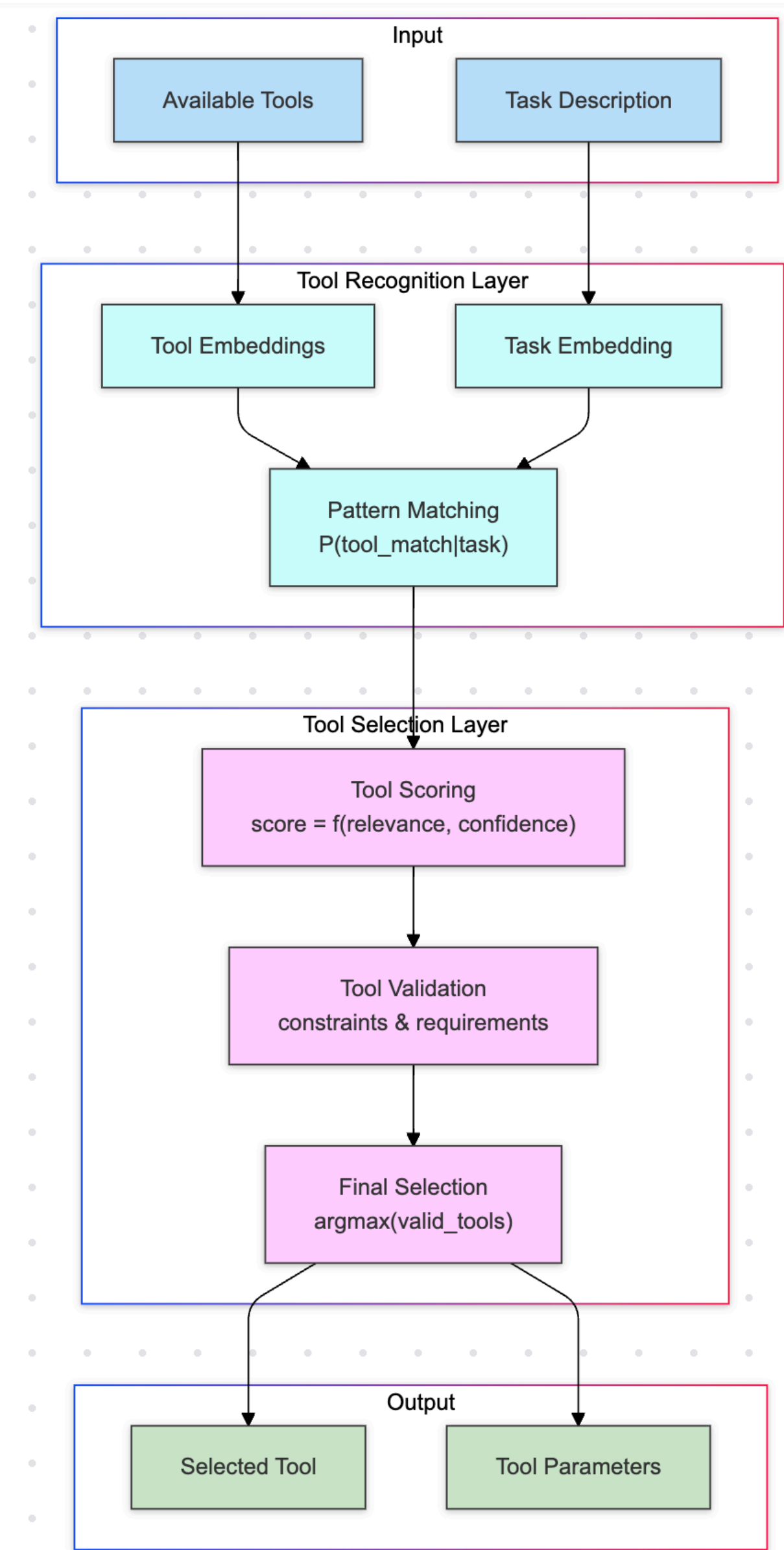
Context-aware **tool selection**      Temperature control

$$\alpha_{\text{tool}} = \text{softmax}\left(\frac{h_{\text{context}} W_Q (h_{\text{tool}} W_K)^T}{\sqrt{d_k}}\right)$$

**Attention** of tools

$$h_{\text{context}} = \text{LayerNorm}([h_{\text{task}}; h_{\text{tools}}])$$

Hidden states of **context**



# Understand chain-of-thoughts (CoT)

Chain-of-thought (CoT) emerges from attention mechanism building up **a working memory** of key-value pairs from each reasoning step, while **hidden states evolve** by attending to this memory to compute next steps, creating **a computational cycle** where each step can query and build upon previous computations.

Attention-based Memory Access

$$c_t = \text{softmax} \left( \frac{h_t W^Q (M_t^K)^T}{\sqrt{d}} \right) M_t^V$$

Hidden State Evolution

$$h_t = f(\text{attention}(h_{t-1}, [h_1, \dots, h_{t-1}]))$$

$$M_t = [K_t, V_t] = [(h_1 W^K, h_1 W^V), (h_2 W^K, h_2 W^V), \dots, (h_t W^K, h_t W^V)]$$

KV Memory Collection (working memory)

