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

RTX 4090
24 GB

tuning a 2B model

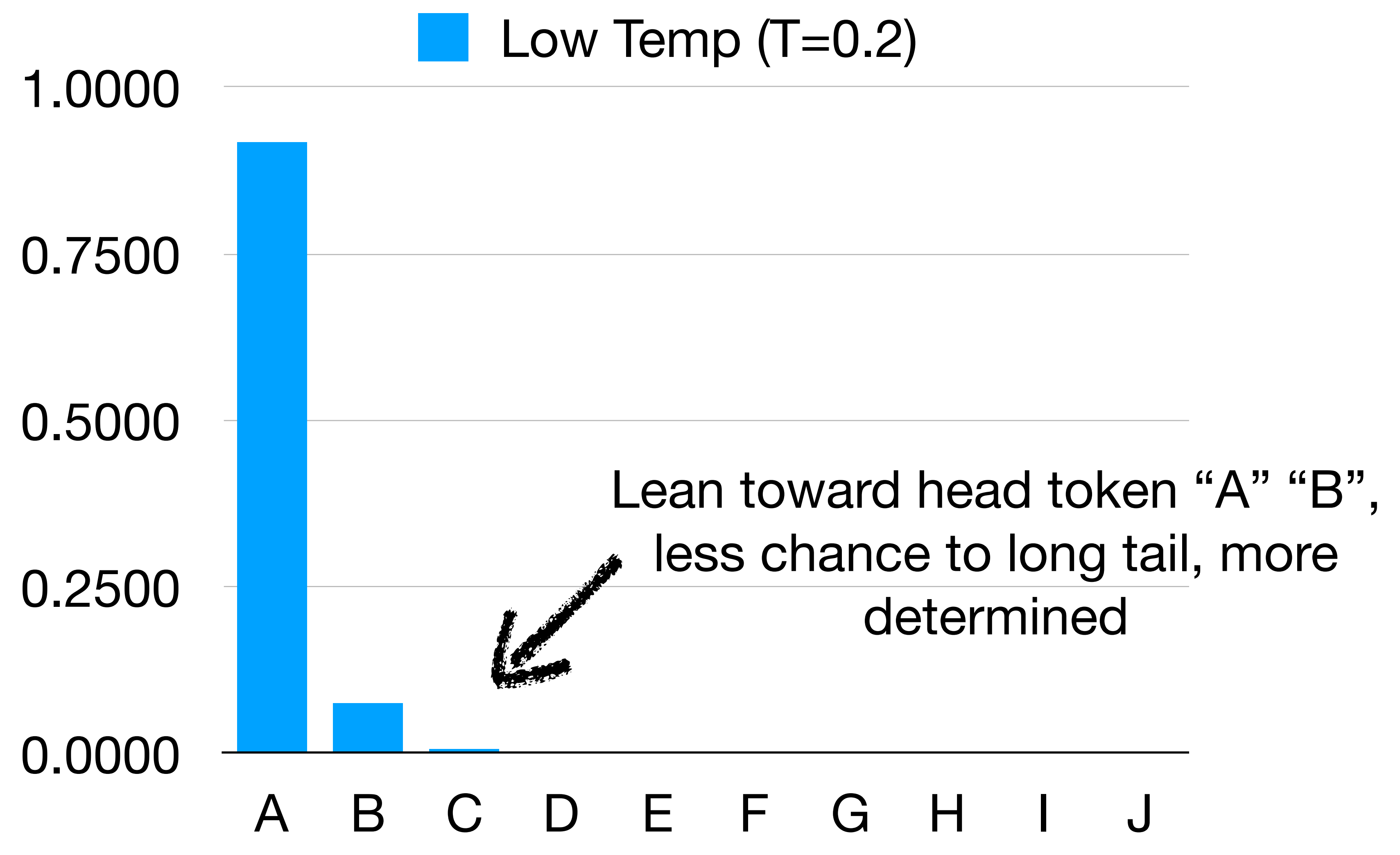
4GB	**Original model**	2B (FP16) = 2*16/8 = 4GB
4GB	**Gradient**	Same size with the model
8 GB	**Optimizer**	Two tensors: mean + variance
~6GB	**Activation**	Model architecture, batch size, sequence length etc, ~1.5x model size
	Something else	

Understand temperature in LLM

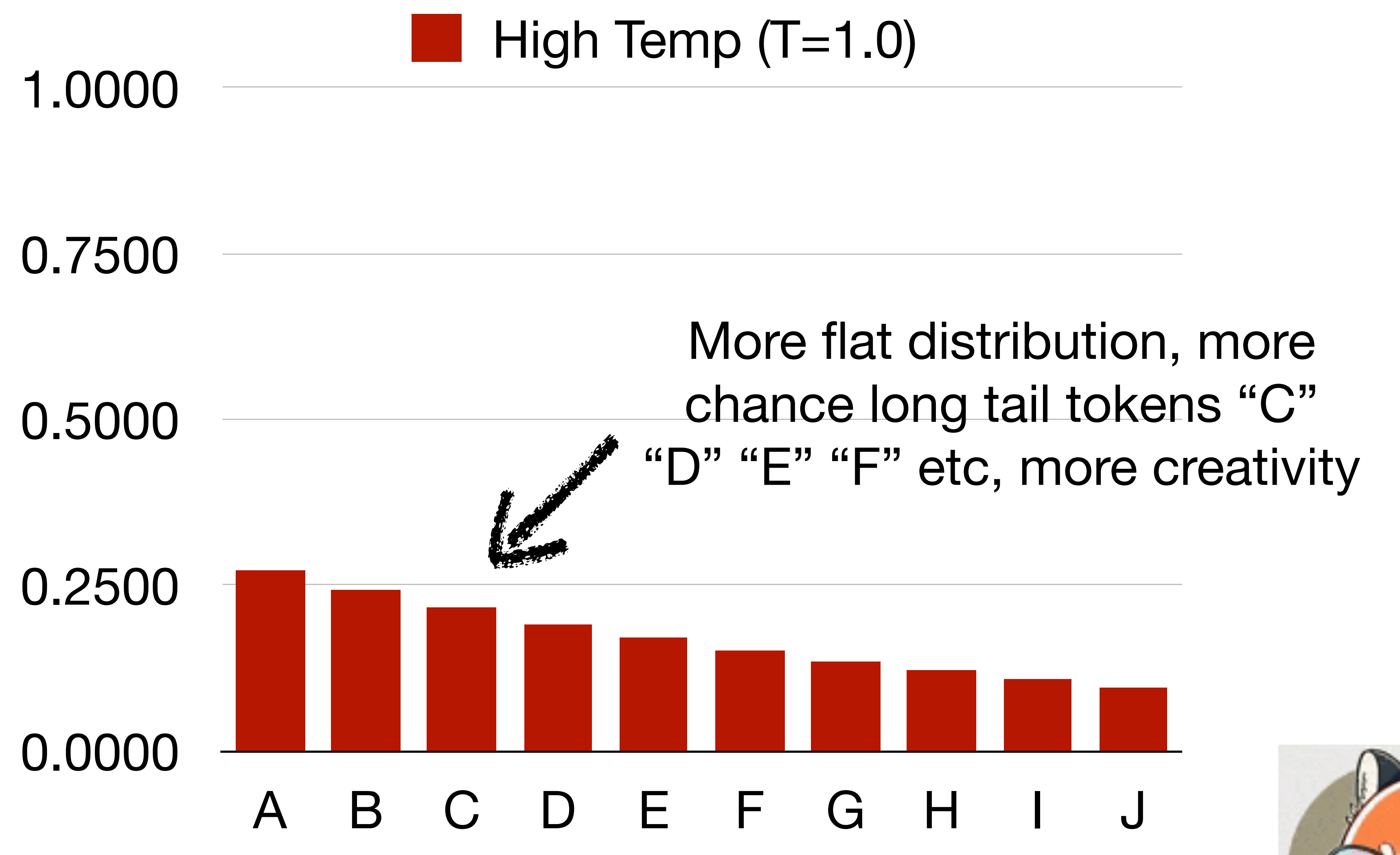
```
response = openai.ChatCompletion.create(
    model='gpt-4o',
    temperature=0.7,
    max_tokens=30,
    messages=[{
        'role': 'user',
        'content': question
    }],
)
```

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

T = 0.2



T = 1.0



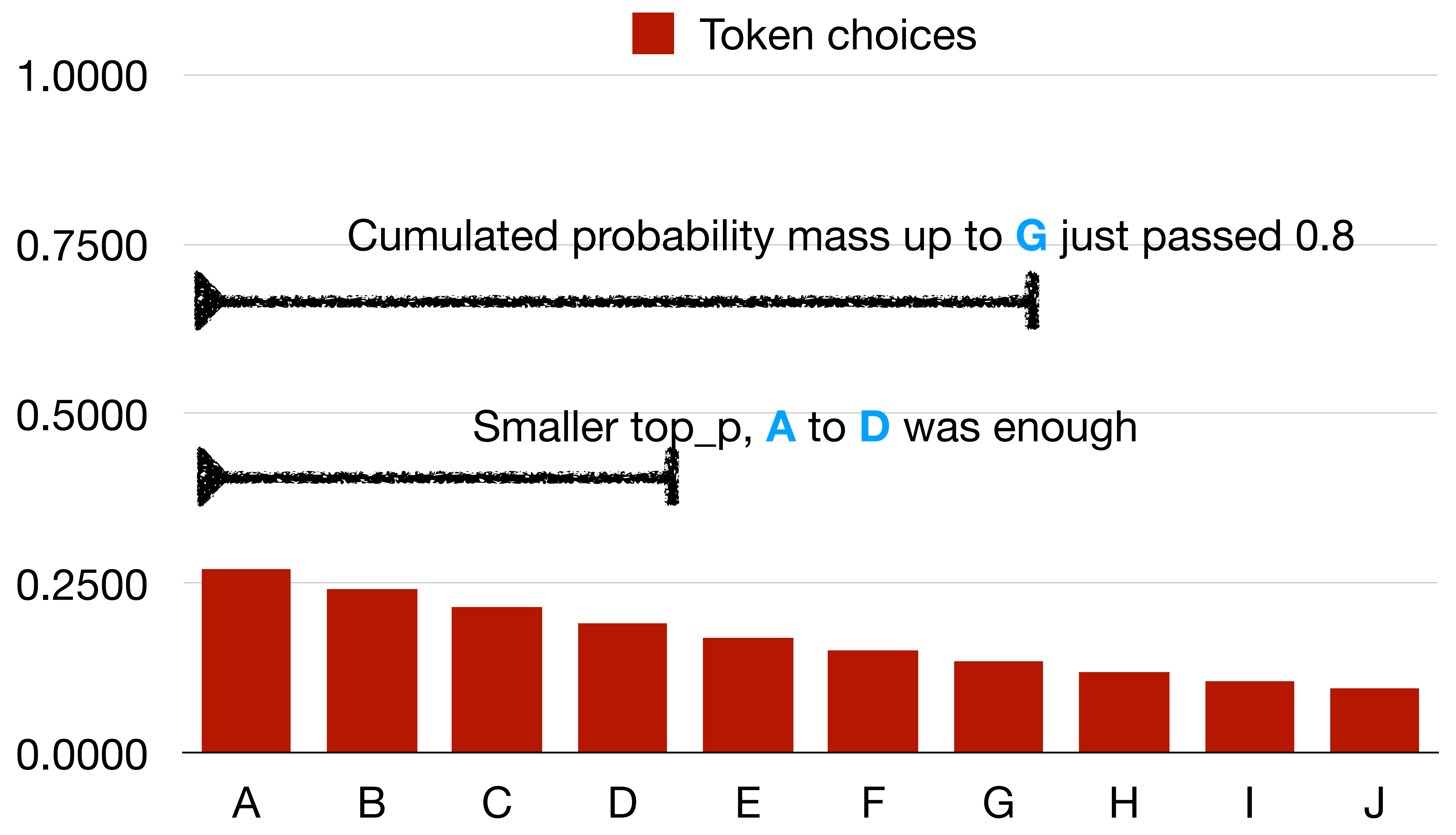
Softmax with Temperature
Hinton et al “Distilling the Knowledge in a Neural Network“ 2015
https://arxiv.org/abs/1503.02531

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

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 **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

Energy of the state

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

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.

Task gradient

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

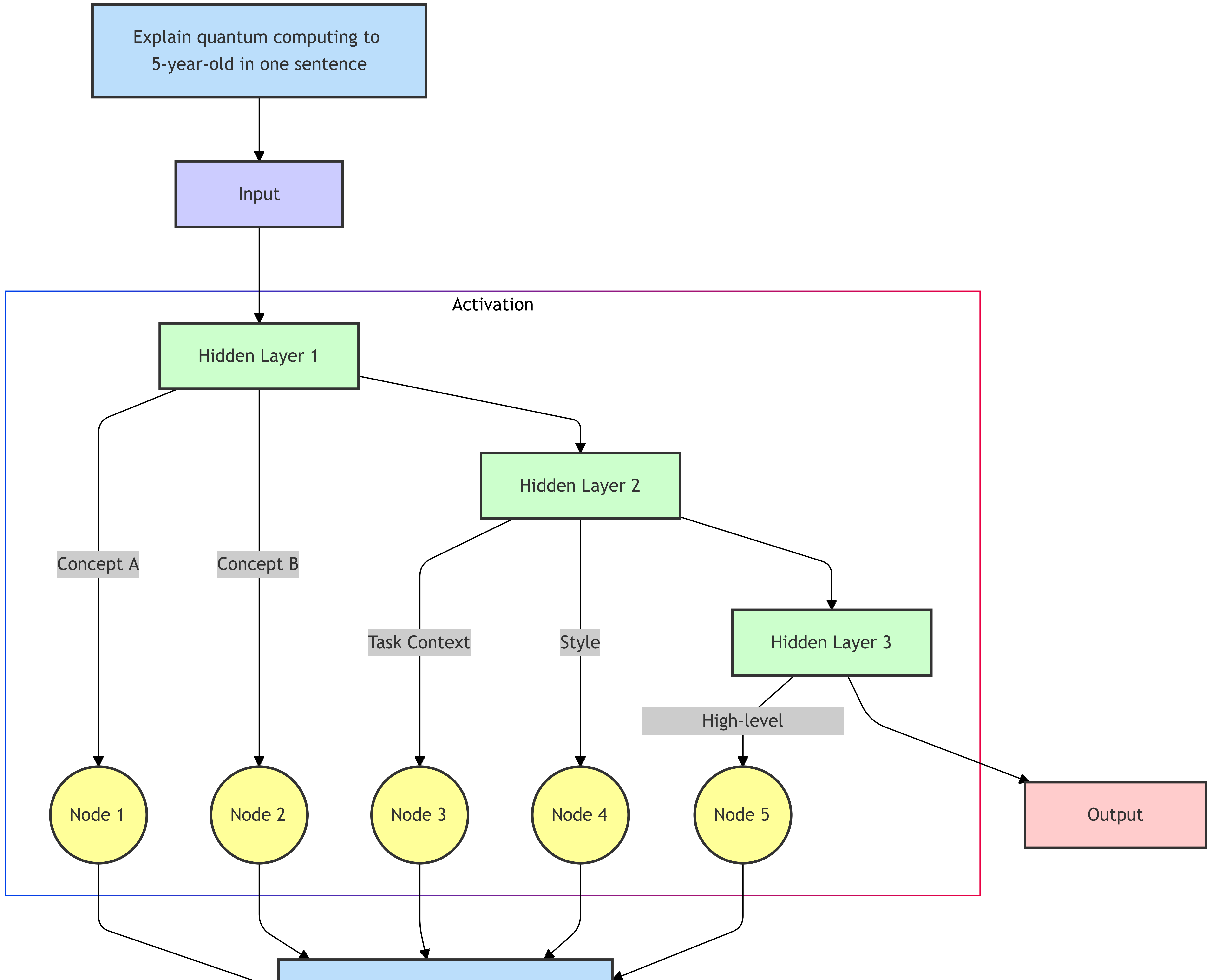
$$P(y_t|\text{prompt}, x) = \text{softmax}(f(h_L + \beta \nabla_{\text{task}})/\tau)$$

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

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(tool|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

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

Context-aware **tool selection**

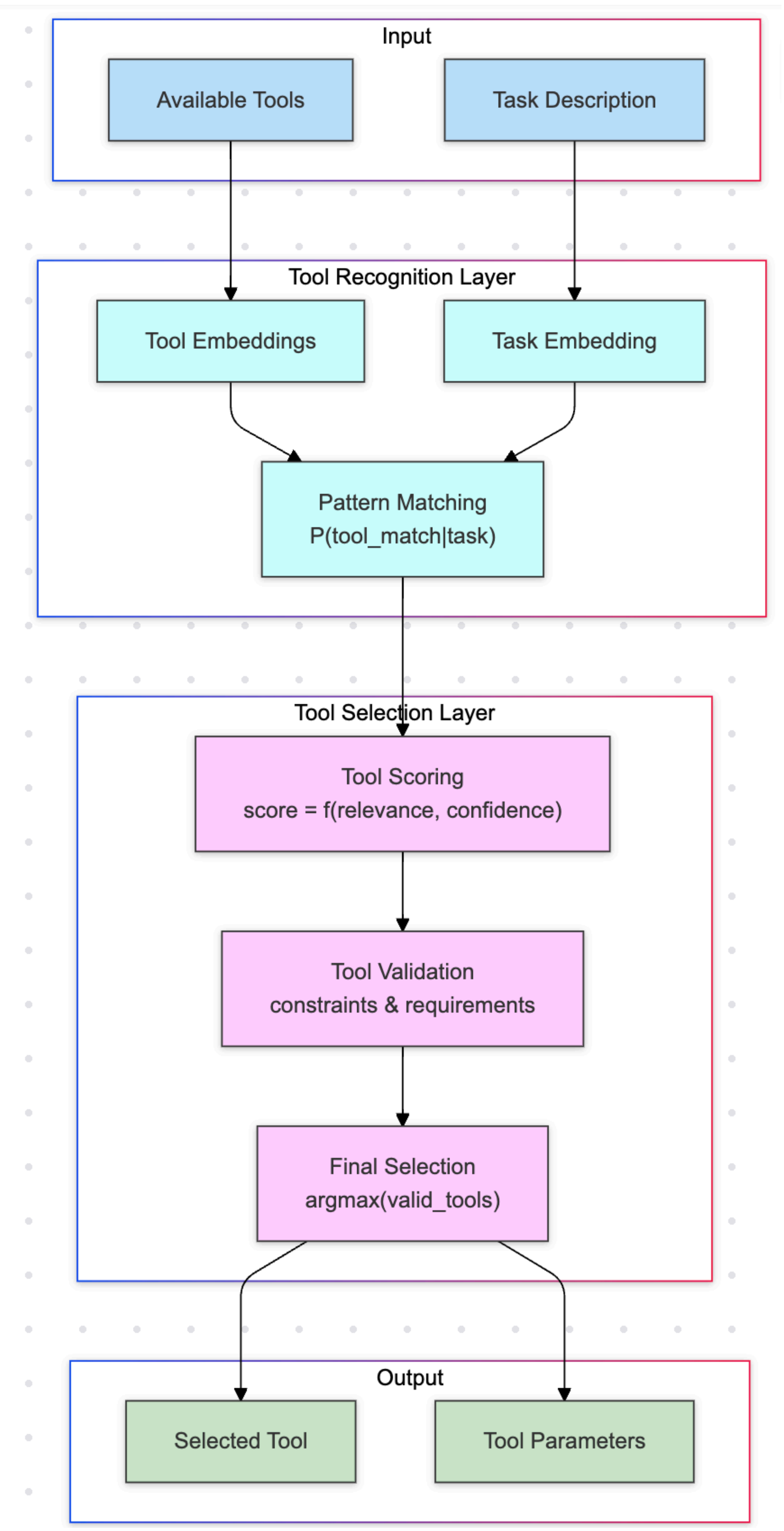
Temperature control

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

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

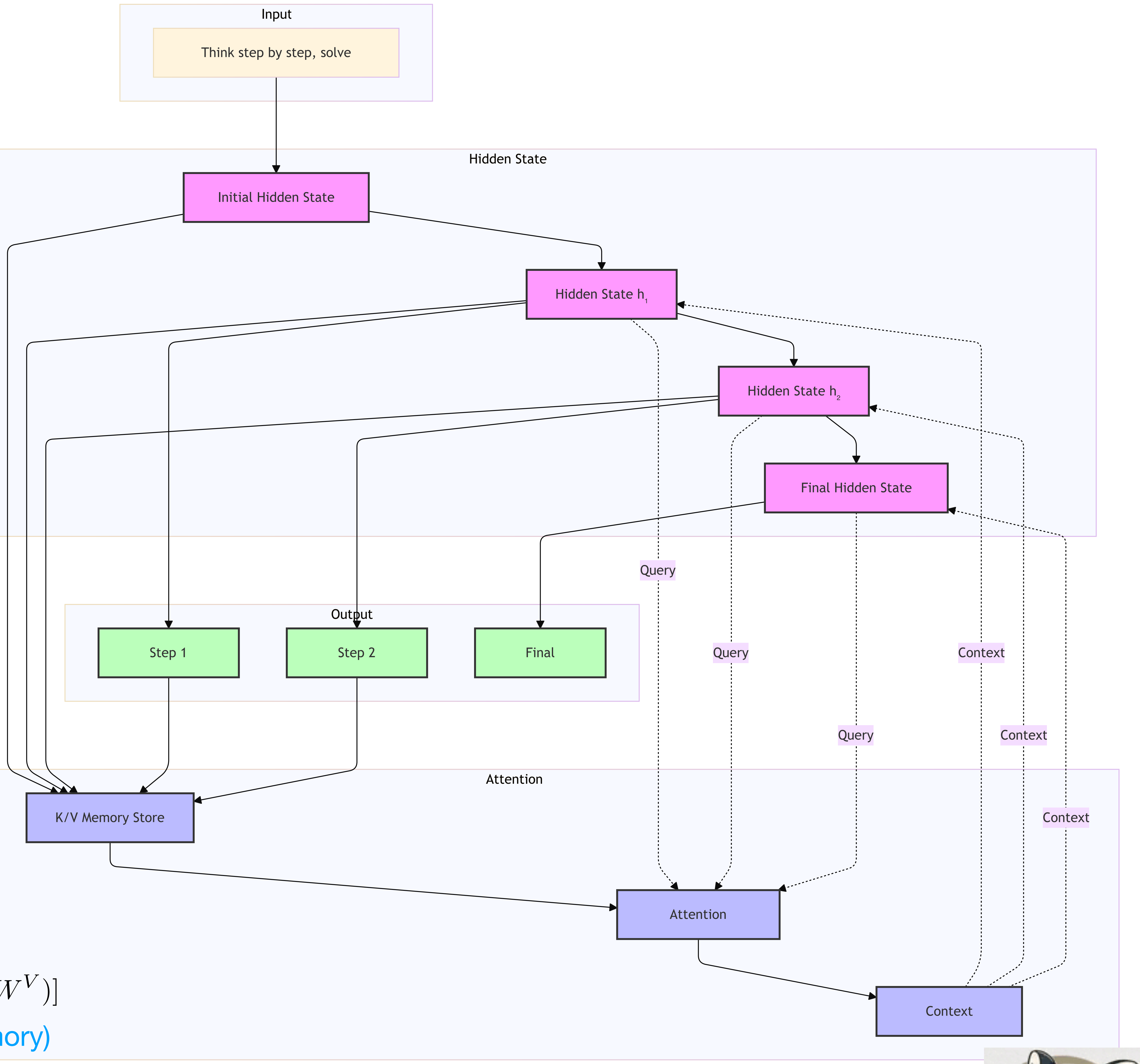
$$c_t = \text{softmax}\left(\frac{h_t W^Q (M_t^K)^T}{\sqrt{d}}\right) M_t^V$$

Hidden State Evolution

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

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

KV Memory Collection (working memory)



Understand LoRA ranks

$d_{\text{head}} = k$

$d_{\text{model}} = d$

$A \in \mathbb{R}^{d \times r}$

rank

$B \in \mathbb{R}^{r \times k}$

rank

$$\Delta W = AB$$

LoRA fine tune weight update

Rank = 4

$$\Delta W_{low} = A_{d \times 4} B_{4 \times k}$$

$$|\Delta W|* = \sum_{i=1}^{4} \sigma_i \leq 4|\Delta W|F$$

spread limit = 4

simpler, more concentrated updates
Better preservation of base model knowledge
Low risk of overfitting

Rank = 32

$$\Delta W_{high} = A_{d \times 32} B_{32 \times k}$$

$$|\Delta W|* = \sum_{i=1}^{32} \sigma_i \leq 32|\Delta W|F$$

spread limit = 32

more complex, distributed changes
More expressive power and complex adaptations
High risk of overfitting

Nuclear Norm (sum all singular values)

$$\frac{|\Delta W|* = \sum_{i=1}^{\min(d,k)} \sigma_i}{|\Delta W|F = \sqrt{\sum_{i=1}^{\min(d,k)} \sigma_i^2}} \leq r$$

Frobenius Norm as
sqrt(sum of squared singular values)

spread limit or "rank"
(power of LoRA fine-tune)

$$\sum_{i=1}^{r} \sigma_i \leq \sqrt{r} \sqrt{\sum_{i=1}^{r} \sigma_i^2}$$

Cauchy-Schwarz Interpretation

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

Understand LLM inference time

Total time

Position embeddings

$$T_{pos} = \mathcal{O}(s \cdot d)$$

$+$

Number of layers $\times$

32 layers in Llama 8B

Self-attention

$$T_{attn} = \mathcal{O}(s \cdot d \cdot h)$$

Feed-forward network

$$T_{ffn} = \mathcal{O}(s \cdot d \cdot 4d)$$

Layer norm

$$T_{ln} = \mathcal{O}(s \cdot d)$$

$+$

Final layer norm

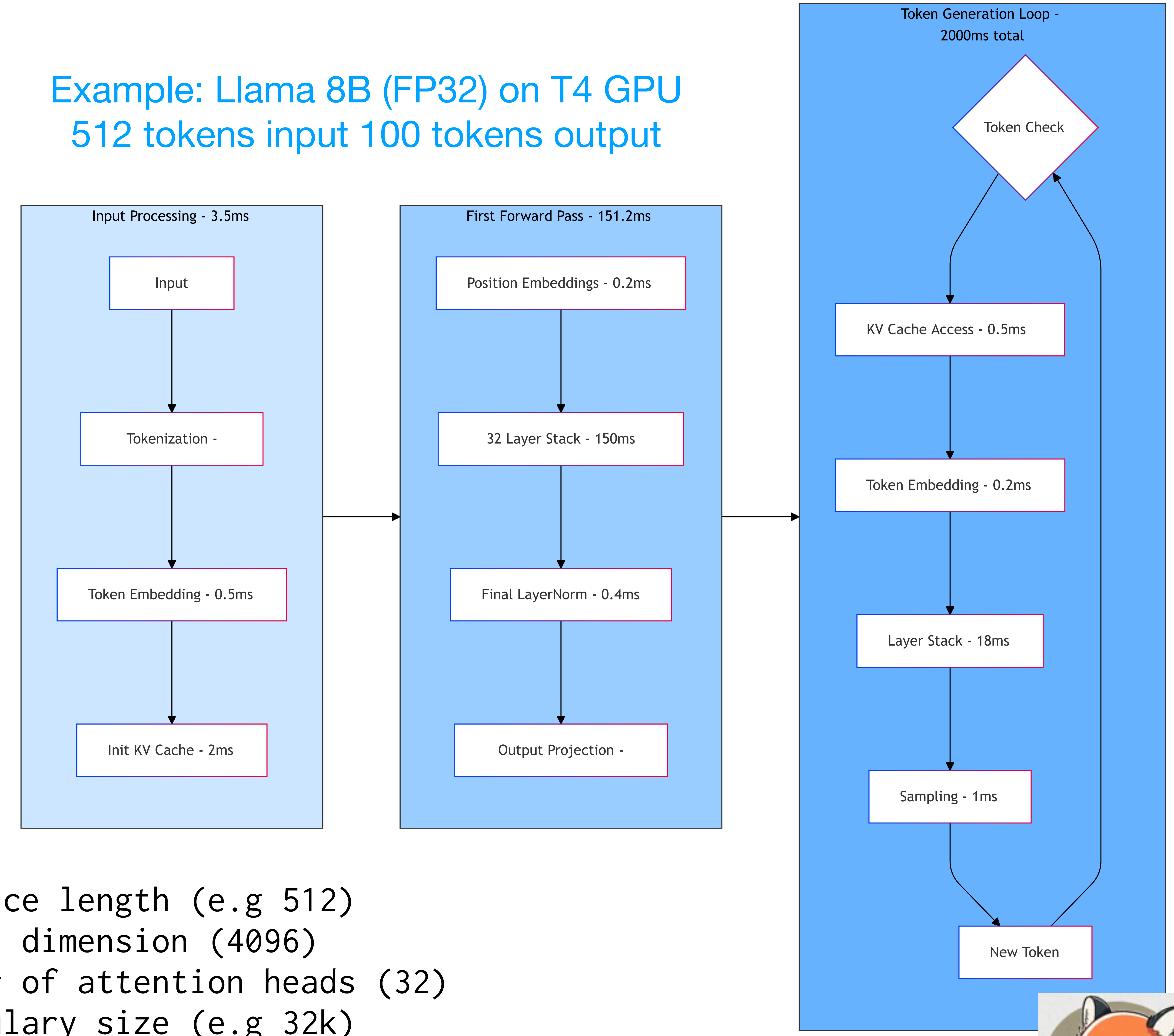
$$T_{ln} = \mathcal{O}(s \cdot d)$$

$+$

Output projection

$$T_{proj} = \mathcal{O}(s \cdot d \cdot v)$$

Example: Llama 8B (FP32) on T4 GPU
512 tokens input 100 tokens output



s = sequence length (e.g 512)
d = hidden dimension (4096)
h = number of attention heads (32)
v = vocabulary size (e.g 32k)

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