

# HW11 writing -- KIMI K2

## Participation A

### Prompt

Now you need to solve homework in a deep learning course. I will give you photos of problems, you should provide clear step-by-step reasoning and precise answers

### Response

Sure, please upload the photos of the problems and I'll help you solve them step by step.

### Prompt

Screen shot question 1

This is question 1

### Response

Part (a): Underfitting with LoRA – what to adjust?

Observation: Model underfits → it's too “simple” for the new task.

LoRA-specific knobs:

- **Rank  $k$**  controls expressiveness: larger  $k \Rightarrow$  richer adaptation.
- **Learning rate** for A, B: too low  $\Rightarrow$  updates too small.
- **Number of LoRA layers**: maybe only a few layers have LoRA modules.

Actionable fix:

Increase the rank  $k$  (or add LoRA modules to more layers).

This directly enlarges the subspace in which the weight can move, giving the model more capacity to fit the new data.

Part (b): A and B initialized to all zeros – why problematic??

Let  $W = W_0 + AB$ .

If  $A = 0$  and  $B = 0$ , then  $W = W_0$  exactly; the forward pass is unchanged.

Gradient w.r.t. any loss  $L$ :

$$\partial L / \partial A = (\partial L / \partial W) \cdot B^T \quad \text{and} \quad \partial L / \partial B = A^T \cdot (\partial L / \partial W)$$

With  $A = B = 0$ , both gradients are **zero**.

SGD (or Adam) will therefore **never update** A or B; they stay zero forever.

Conclusion: **LoRA adaptation never starts**—the network remains frozen at the pre-trained weight



**Part (c): Xavier-init both A and B – why might fine-tuning still struggle?**

Xavier initialization is designed for **standard** weight matrices that are **updated directly**.

In LoRA the **effective update** is  $\Delta W = AB$ .

If both A and B are Xavier-scaled independently, the **magnitude of  $\Delta W$  at init** can be **much larger** than intended.

Empirically,  $\|\Delta W\|$  can be  $O(\sqrt{ml})$  (because each element is a dot-product of two Xavier vectors).

This causes:

- **Large initial output shift** → activations blow up → training instabilities.
- **Very large gradients** → possible divergence at the first step.

A safer practice is to **initialize one matrix to zero** (or very small) and the other with a suitable variance so that  $\Delta W \approx 0$  at the start, keeping the effective weight close to  $W_0$ .

**Part (d): Memory footprint comparison**

LoRA weights:

- A:  $m \times k$  floats →  $4mk$  bytes
  - B:  $k \times l$  floats →  $4kl$  bytes
- Total =  $4k(m + l)$  bytes.

Full-rank adaptation matrix  $\Delta W_{\text{full}}$ :

$m \times l$  floats →  $4ml$  bytes.

Memory ratio:

$$\text{LoRA} / \text{full} = 4k(m+l) / 4ml = k(m+l)/(ml) = k(1/l + 1/m).$$

Because  $k \ll \min(m, l)$ , LoRA needs **orders-of-magnitude less** extra memory.

**Annotation:**

This question KIMI K2 answers perfectly.

**Prompt**

Screen shot question 2(a)

This is question 2(a)

**Response**



Question 2(a) asks for the **mathematical expression for the final logits** of a **zero-layer Transformer** and then wants a plain-language description of what that model is doing.

1. Zero-layer Transformer: only an embedding lookup followed by an unembedding.

Variables

$T \in \mathbb{R}^{\text{vocab} \times n_{\text{ctx}}}$  – one-hot matrix; each column is a token.

$W_E \in \mathbb{R}^{\{d_{\text{model}} \times \text{vocab}\}}$  – token embedding matrix.

$W_U \in \mathbb{R}^{\text{vocab} \times d_{\text{model}}}$  – unembedding matrix (maps  $d_{\text{model}} \rightarrow \text{logits}$ ).

Step-by-step forward pass

a. Embed:

$X^0 = W_E T$  (shape  $d_{\text{model}} \times n_{\text{ctx}}$ )

b. Unembed:

$L = W_U X^0 = W_U W_E T$  (shape  $\text{vocab} \times n_{\text{ctx}}$ )

That is the **entire computation**; no layers, no attention, no MLPs.



Answer to (i)

Final logits

$L = W_U W_E T \in \mathbb{R}^{\text{vocab} \times n_{\text{ctx}}}$

Answer to (ii)

Algorithm implemented:

“Look up each token’s embedding vector, then score every vocabulary word by its dot-product similarity with that vector.”

Information the prediction at position  $t$  depends on:

**Only the identity of the single token at position  $t$  itself**—no context from any other positions is used.

Thus the zero-layer model is nothing more than a **bigram/unigram classifier** that predicts the next token based solely on the current token’s learned embedding.

### Annotation:

This question KIMI K2 answers perfectly.

### Prompt

Screen shot question 2(b)

This is question 2(b)

### Response



We are asked to prove that the original “concatenate-then-project” multi-head attention in Vaswani et al. is algebraically identical to the “additive / independent-circuits” view used in the mechanistic-interpretability literature.

Notation (consistent with the prompt)

- $H$  heads
- $d_{\text{head}} = d_{\text{model}} / H$
- For head  $h$ :  
 $r^h \in \mathbb{R}^{d_{\text{head}} \times n_{\text{ctx}}}$  (the value-mix output of that head)
- Concatenation view:  
 $R = [r^1; r^2; \dots; r^H] \in \mathbb{R}^{(H \cdot d_{\text{head}}) \times n_{\text{ctx}}}$  (vertical stack)  
 $\text{Output} = W R$  with  $W \in \mathbb{R}^{d_{\text{model}} \times (H \cdot d_{\text{head}})}$
- Additive view:  
 Each head has its own independent output matrix  $W_O^h \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$   
 $\text{Output} = \sum_{h=1..H} W_O^h r^h$

Part (i) – show equivalence by constructing  $W$

Define the block-row matrix

$$W \triangleq [W_O^1 \mid W_O^2 \mid \dots \mid W_O^H] \quad (\text{block row of } H \text{ blocks, each } d_{\text{model}} \times d_{\text{head}})$$

Then

$$\begin{aligned} W R &= [W_O^1 \mid \dots \mid W_O^H] [r^1; \dots; r^H] \\ &= W_O^1 r^1 + W_O^2 r^2 + \dots + W_O^H r^H \\ &= \sum_h W_O^h r^h \end{aligned}$$

Hence the two expressions are identical.

Conversely, any block-row  $W$  can be split into  $H$  blocks  $W_O^h$ , giving the additive form.

Part (ii) – advantages / disadvantages

Concatenation view (Vaswani)

- Single matrix multiply  $WR$  is highly optimized on GPUs (one large GEMM).
- Standard in libraries; no code change needed when  $H$  varies.
  - Conceptually treats heads as one big lump; harder to analyze individual head contributions.
  - $W$  has  $d_{\text{model}}^2$  parameters (same total count as additive, but feels monolithic).

Additive / independent view

- Each head's contribution is explicitly isolated; aligns with mechanistic interpretability ("circuits").
- Easy to ablate, probe, or mask single heads without touching others.
  - $H$  separate smaller GEMMs; can be slightly less efficient on modern hardware unless fused.
  - More boiler-plate code if  $H$  is large.

---

Bottom line

The two formulas are literally the same computation reshuffled; choosing one over the other is purely about software engineering and interpretability, not expressive power.

#### Annotation:

This question KIMI K2 answers perfectly.

#### Prompt

Screen shot question 2(c)

This is question 2(c)

#### Response



Question 2(c) separates an attention head into two conceptual circuits:

- **QK circuit** – decides *where* to look (attention pattern)
- **OV circuit** – decides *what* to copy (value information)

Here we analyze only the **QK circuit**.

(i) Derive the pre-softmax score  $S_{ij}$  as a bilinear form in the residual-stream vectors  $X_i, X_j$  and a single virtual matrix  $W_{QK}$ .

Definitions (per token columns of the residual stream):

$$\begin{aligned} Q_i &= W_Q X_i && \in \mathbb{R}^{d_{\text{head}}} \\ K_j &= W_K X_j && \in \mathbb{R}^{d_{\text{head}}} \end{aligned}$$

Then

$$\begin{aligned} S_{ij} &= Q_i^T K_j \\ &= (W_Q X_i)^T (W_K X_j) \\ &= X_i^T W_Q^T W_K X_j \end{aligned}$$

Define the **virtual QK matrix**:

$$W_{QK} \triangleq W_Q^T W_K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$$



(1)  $W_{QK} = I$

High score when  $X_i^T X_j$  is large → **cosine similarity is large**.

Head “looks for” tokens whose residual vectors point in the **same direction** as the current token.

(2)  $W_{QK} = -I$

High score when  $X_i^T X_j$  is **large negative** → **opposite direction**.

Head attends to tokens that are **anti-correlated** with the current token.

(3)  $W_{QK} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$  (rank-1 projector onto 1st coordinate)

$$S_{ij} = X_{i1} X_{j1}$$

High score when both tokens have **large 1st-component** with **same sign**.

Head ignores dimensions 2 and 3; it purely **matches on the first feature**.

Difference from identity: identity uses **all dimensions equally**; this head uses **only one**.

(4)  $W_{QK} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$  (skew-symmetric)

$$S_{ij} = X_{i2} X_{j1} - X_{i1} X_{j2}$$

This is the **2-D cross-product** (or oriented area).

High score when the vector  $(X_{j1}, X_{j2})$  is  **$\pi/2$  ahead** of  $(X_{i1}, X_{i2})$  in the (1,2)-plane.

After softmax, the head **attends most to tokens that are “90° to the left”** of the current token in that 2-D subspace; it creates a **fixed cyclic bias** in position-based patterns.



### Annotation:

For part1, KIMI K2 answers perfectly. However part2, it is wrong. I found that KIMI K2 wrongly scanned the matrix of photo. After I provide correct matrix, it turn out to the correct answer



### Additional Prompt

I think in (2)  $W_{QK} = \begin{bmatrix} 1, 0, 0 \\ 0, 0, 0 \\ 0, 0, 0 \end{bmatrix}$

in (3)  $W_{QK} = \begin{bmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, -1 \end{bmatrix}$

in (4)  $W_{QK} = -I$

### Additional Response

Same setup:  $d_{\text{model}} = 3$ ,  $S_{ij} = X_i^T W_{QK} X_j$

(1)  $W_{QK} = I$

High score  $\Leftrightarrow X_i$  and  $X_j$  are **similar** (same direction).

Head looks for “tokens like me”.

(2)  $W_{QK} = \text{diag}(1, 0, 0)$

$S_{ij} = X_{i1} X_{j1}$

High score when **first components** have large equal sign; other dimensions ignored.

Head **matches only on feature 1**.

(3)  $W_{QK} = \text{diag}(1, 1, -1)$

$S_{ij} = X_{i1} X_{j1} + X_{i2} X_{j2} - X_{i3} X_{j3}$

High score when the first two coordinates are **similar** and the third is **opposite**.

Head looks for “same on dims 1&2, opposite on dim 3”.

(4)  $W_{QK} = -I$

$S_{ij} = -X_i^T X_j$

High score when  $X_i$  and  $X_j$  are **anti-aligned** (cosine  $\approx -1$ ).

After softmax, head **attends most to the least similar token**.

### Prompt

Screen shot question 2(d)

This is question 2(d)

### Response



Question 2(d) – the OV (output-value) circuit that actually **moves information** once the QK circuit has decided **where** to look.

(i) Attention-weighted average

Given

$X \in \mathbb{R}^{d_{\text{model}} \times n_{\text{ctx}}}$  (column  $j$  = vector at src position  $j$ )

$A \in \mathbb{R}^{n_{\text{ctx}} \times n_{\text{ctx}}}$   $A_{ij}$  = weight from dest  $i$  to src  $j$

We need the  $i$ -th column of  $XA^T$ .

Compute:

$$\begin{aligned}(XA^T)_{\bullet i} &= X(A^T)_{\bullet i} = X(\text{row } i \text{ of } A) \\ &= \sum_{j=1}^{n_{\text{ctx}}} A_{ij} X_{\bullet j}\end{aligned}$$

which is exactly an **attention-weighted average** of the source vectors, with weights  $\{A_{ij}\}_j$ .

(ii) Final residual stream after one attention layer ( $H$  heads, no layernorm, no MLPs)

Initial stream:  $X^0 = W_E T$

Each head  $h$  contributes

$$h_h(X^0) = W_{\text{OV}}^h X^0 (A^h)^T \in \mathbb{R}^{d_{\text{model}} \times n_{\text{ctx}}}$$

Additive residual update:

$$X^{\text{final}} = X^0 + \sum_{h=1}^H W_{\text{OV}}^h X^0 (A^h)^T$$

(iii) Update at token position  $t$  lies in  $\text{Col}(W_{\text{OV}}^h)$

The update vector added to position  $t$  by head  $h$  is column  $t$  of the head's output:

$$\begin{aligned}\delta_t^h &= h_h(X^0)_{\bullet t} \\ &= W_{\text{OV}}^h [X^0 (A^h)^T]_{\bullet t} \\ &= W_{\text{OV}}^h v_t \quad \text{where } v_t = \sum_j A^h_{tj} X^0_{\bullet j} \in \mathbb{R}^{d_{\text{model}}}\end{aligned}$$

Thus  $\delta_t^h$  is **literally**  $W_{\text{OV}}^h$  acting on some vector  $v_t$ , hence

$$\delta_t^h \in \text{Col}(W_{\text{OV}}^h)$$

Since  $W_{\text{OV}}^h$  has shape  $d_{\text{model}} \times d_{\text{model}}$  but **rank**  $\leq d_{\text{head}} \ll d_{\text{model}}$ , its column space is a **low-dimensional subspace** (dimension  $d_{\text{head}}$ ).

Therefore each head can **write only into this fixed subspace**, no matter what the attention patte



### Annotation:

This question KIMI K2 answers perfectly.

### Prompt

Screen shot question 2(e)

This is question 2(e)



## Response



Question 2(e) – use the SVD of

$$W_{OV} = U \Sigma V^T \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$$

to pin down exactly **what** an attention head can read from a source token and **where** it can write in the destination residual stream.

---

(i)  $\text{rank}(W_{OV}) \leq d_{\text{head}}$

By construction

$$W_{OV} = W_O W_V \quad \text{with} \quad W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}, \quad W_V \in \mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$$

Matrix product rank is at most the minimum inner dimension, so

$$\text{rank}(W_{OV}) \leq d_{\text{head}} \ll d_{\text{model}}$$

$\Rightarrow W_{OV}$  is **low-rank** (typical heads use  $d_{\text{head}} = 64$ ,  $d_{\text{model}} = 512 \dots 4096$ ).

---

(ii) SVD expansion of the OV action on a single source vector  $x_{\text{src}}$

$$\begin{aligned} W_{OV} x_{\text{src}} &= U \Sigma V^T x_{\text{src}} \\ &= \sum_{k=1}^r \sigma_k (v_k^T x_{\text{src}}) u_k \quad (r = \text{rank} \leq d_{\text{head}}) \end{aligned}$$

Each term is a **scalar projection** of  $x_{\text{src}}$  onto the **right singular vector**  $v_k$ , scaled by  $\sigma_k$ , and placed in the **left singular vector** direction  $u_k$  in the destination space.

(iii) Read and write subspaces

(a) **Read subspace**

$\text{span}\{v_1, \dots, v_r\}$  (columns of  $V$ )

Only components of  $x_{\text{src}}$  lying in this subspace produce non-zero  $(v_k^T x_{\text{src}})$ ; information outside is **invisible** to the head.

(b) **Write subspace**

$\text{span}\{u_1, \dots, u_r\}$  (columns of  $U$ )

Every output vector  $W_{OV} x_{\text{src}}$  is a linear combination of these  $u_k$ 's, so the head can **deposit information only into this fixed  $r$ -dimensional subspace** of the residual stream.

(c) Information orthogonal to the read subspace ( $x_{\text{src}} \perp \text{span}\{v_k\}$ ) gives  $v_k^T x_{\text{src}} = 0$  for all  $k$ , hence **zero contribution** to the output; it is **completely ignored**.

---

(iv) Relation to value projection  $W_V$

The value vector actually fed to attention is  $v = W_V x_{\text{src}} \in \mathbb{R}^{d_{\text{head}}}$ .

Because  $W_V$  maps into a  $d_{\text{head}}$ -dimensional space, **at most  $d_{\text{head}}$  independent linear features** of  $x_{\text{src}}$  can influence  $v$ .

The read subspace  $\text{span}\{v_k\}$  is exactly the **pre-image** of these features under  $W_V$ , so it is natural that its dimension is  $\leq d_{\text{head}}$ .

(v) Synthesis: SVD as a complete "communication channel" picture

- $V$  selects which **features** (directions) of the source token are read.
- $\Sigma$  scales their **importance**.
- $U$  places the scaled features into a specific **target subspace** of the destination residual stream.

The attention pattern  $A$  then **re-weights** how much each destination position listens to each source position, but **cannot change the linear subspaces** that are read from or written to; those are hard-wired by the SVD of  $W_{OV}$ .

### Annotation:

This question KIMI K2 answers perfectly.

### Prompt

Screen shot question 5(b)

This is question 5(b)

### Response



We are given the Chinchilla scaling law:

$$L(N, D) = A N^{-\alpha} + B D^{-\beta} + L_0 \quad (\alpha = 0.34, \beta = 0.28, A = 406.4, B = 410.7, L_0 = 1.69)$$

and the compute budget:

$$C = C_0 N D \quad \text{with} \quad C_0 = 6 \text{ FLOP} / (\text{param} \cdot \text{token}).$$

Goal: for fixed  $C$ , minimise  $L$  w.r.t.  $N$  and  $D$  under the equality  $C_0 N D = C$ .

Step 1 – eliminate  $D$

From the constraint:  $D = C / (C_0 N)$ .

Insert into  $L$ :

$$\begin{aligned} L(N) &= A N^{-\alpha} + B (C / (C_0 N))^{-\beta} + L_0 \\ &= A N^{-\alpha} + B (C_0 / C)^{\beta} N^{\beta} + L_0. \end{aligned}$$

Step 2 – minimise w.r.t.  $N$

$$dL/dN = -\alpha A N^{-\alpha-1} + \beta B (C_0 / C)^{\beta} N^{\beta-1} = 0$$

Multiply by  $N^{\alpha+1}$ :

$$-\alpha A + \beta B (C_0 / C)^{\beta} N^{\alpha+\beta} = 0$$



$$\Rightarrow N^{\alpha+\beta} = (\alpha A) / (\beta B) \cdot (C / C_0)^{\beta}$$

Hence the **symbolic Chinchilla-optimal** values:

$$N_{\text{opt}}(C) = [ (\alpha A) / (\beta B) ]^{1/(\alpha+\beta)} \cdot (C / C_0)^{\beta/(\alpha+\beta)}$$

$$\begin{aligned} D_{\text{opt}}(C) &= C / (C_0 N_{\text{opt}}) \\ &= [ (\beta B) / (\alpha A) ]^{1/(\alpha+\beta)} \cdot (C / C_0)^{\alpha/(\alpha+\beta)} \end{aligned}$$

Step 3 – plug in the given numbers

$$\alpha = 0.34, \beta = 0.28 \Rightarrow \alpha + \beta = 0.62$$

$$\alpha A / (\beta B) = (0.34 \times 406.4) / (0.28 \times 410.7) \approx 1.188$$

Exponents:

$$\beta/(\alpha+\beta) \approx 0.452 \quad \alpha/(\alpha+\beta) \approx 0.548$$

Therefore

$$N_{\text{opt}}(C) \approx 1.29 \times (C / 6)^{0.452} \quad (\text{parameters})$$

$$D_{\text{opt}}(C) \approx 0.78 \times (C / 6)^{0.548} \quad (\text{tokens})$$

(With C in FLOP, N in parameters, D in tokens.)

### Annotation:

This question KIMI K2 answers perfectly. The result is same as the solution if you put 6 out.

### Prompt

Screen shot question 5(c)(d)(e)

This is question 5(c)(d)(e)

### Response



Let's finish the Fermi-estimation homework in one clean sweep.

---

(c) Compute & dataset for  $N = 100$  trillion ( $10^{14}$ ) params

Use the **direct Chinchilla fits** quoted in the paper (already rounded):

$$N_{\text{opt}}(C) \approx 1.26 \times (C / 6)^{0.50} \quad (\text{parameters})$$

$$D_{\text{opt}}(C) \approx 0.89 \times (C / 6)^{0.50} \quad (\text{tokens})$$

Invert the first equation to get  $C$  when  $N$  is fixed:

$$(C / 6)^{0.50} = N / 1.26$$

$$\Rightarrow C = 6 \times (N / 1.26)^2$$

$$= 6 \times (10^{14} / 1.26)^2$$

$$\approx 3.8 \times 10^{28} \text{ FLOP}$$

Insert into  $D$  equation:

$$D = 0.89 \times (10^{14} / 1.26)$$

$$\approx 7.1 \times 10^{13} \text{ tokens} \quad (71 \text{ trillion tokens})$$

---

(d) How much text is 71 trillion tokens?

- 1 token  $\approx 0.71$  English word  $\Rightarrow$   
Words  $\approx 7.1 \times 10^{13} \times 0.71 \approx 5.0 \times 10^{13}$  words (50 trillion words)
- 1 page  $\approx 400$  words  $\Rightarrow$   
Pages  $\approx 1.25 \times 10^{11}$  pages
- 1 book  $\approx 300$  pages  $\Rightarrow$   
Books  $\approx 4.2 \times 10^8$  books  $\approx$  **420 million books**

Context

- Library of Congress  $\approx 40$  million books
- Google Books corpus  $\approx 40$  million books

So Chinchilla-optimal training data for 100-T-param GPT-6  $\approx 10\times$  the entire Library of Congress or  $10\times$  Google Books.

(e) 16-bit (2-byte) parameter memory

- 1 billion params:  $10^9 \times 2 \text{ B} = 2 \times 10^9 \text{ B} = \mathbf{2 \text{ GB}}$
- 1 trillion params:  $10^{12} \times 2 \text{ B} = 2 \times 10^{12} \text{ B} = \mathbf{2 \text{ TB}}$
- 100 trillion params:  $10^{14} \times 2 \text{ B} = \mathbf{200 \text{ TB}}$

H200 GPU VRAM  $\approx 100 \text{ GB} \rightarrow \mathbf{2 \text{ TB model needs}}$

$2 \text{ TB} / 100 \text{ GB} = 20 \text{ GPUs}$  **just to hold the weights** (training would need many more for activations/gradients).

---

One-sentence summary

A Chinchilla-optimal 100-T-param “GPT-6” costs  $\approx 4 \times 10^{28}$  FLOP, needs  $\approx 70 \text{ T}$  tokens ( $\approx 400 \text{ M}$  books), and **200 TB** of 16-bit param memory—about **20 H200 GPUs** for bare-metal storage alone.

### Annotation:

For question(e) KIMI K2 answers perfectly.

However, for question(c)(d), it didn’t fully understand question correctly.

### Additional Prompt

You should use  $N_{\text{opt}}(C) = 0.1 C^{0.5}$ ;  $D_{\text{opt}}(C) = 1.7 C^{0.5}$  rather than part(b)

### Additional Response



Got it — we’ll use the **direct Chinchilla fits** given in the problem statement:

$$N_{\text{opt}}(C) = 0.1 C^{0.5}, \quad D_{\text{opt}}(C) = 1.7 C^{0.5}$$

with  $C$  in FLOPs,  $N$  in parameters, and  $D$  in tokens.

---

(c) Compute and dataset for  $N = 100 \text{ trillion} = 1e14$  parameters

Solve for  $C$  :

$$1e14 = 0.1 C^{0.5} \Rightarrow C^{0.5} = 1e15 \Rightarrow C = 1e30 \text{ FLOPs}$$

Then:

$$D = 1.7 C^{0.5} = 1.7 \times 1e15 = 1.7e15 \text{ tokens}$$

### Additional Prompt

question (d) is asking 1000 trillion tokens

### Additional Response



You're right — question (d) specifically asks about **1000 trillion tokens**, not the 1.7 trillion we derived earlier. Let's redo **only part (d)** with  $D = 1000 \text{ trillion} = 1e15 \text{ tokens}$ .

(d) Dataset size: **1e15 tokens**

- **Tokens to words:**  
 $1 \text{ token} \approx 0.71 \text{ English words}$   
 $\rightarrow 1e15 \times 0.71 \approx \mathbf{7.1e14 \text{ words}}$
- **Pages** (400 words/page):  
 $7.1e14 / 400 = \mathbf{1.775e12 \text{ pages}}$
- **Books** (300 pages/book):  
 $1.775e12 / 300 \approx \mathbf{5.9 \text{ billion books}}$

Context Check

- **Library of Congress:** ~40 million books
- **Google Books:** ~40 million books

So 1000 trillion tokens  $\approx$  5.9 billion books  $\approx$  150 $\times$  the entire Library of Congress or Google Bo...



### Prompt

Screen shot question 5(f)

This is question 5(f)

### Response



Question 5(f) – yearly \$ cost to store a **100 trillion-parameter** model (200 TB) in 2025 prices.

Storage size

100 T params  $\times$  2 bytes = **200 TB** =  $2 \times 10^{11}$  MB.

2025 unit prices (from table)

- **SSD/Flash:** \$0.00005 / MB
- **DRAM:** \$0.0040 / MB

Assume media is replaced **every year** (no amortisation beyond 1 yr).

Yearly cost

SSD:  $2 \times 10^{11} \text{ MB} \times \$0.00005 / \text{MB} = \mathbf{\$10 \text{ million}}$

DRAM:  $2 \times 10^{11} \text{ MB} \times \$0.0040 / \text{MB} = \mathbf{\$800 \text{ million}}$

Bottom line

- **SSD:** ~\$10 M per year
- **DRAM:** ~\$800 M per year



### Annotation:

The answer is wrong. It is too big.

### Additional Prompt

carefully calculate it again, you are bigger than answer

### Additional Response



You're right — let's **carefully recalculate** with **exact units and no rounding errors**.

Given:

- 100 trillion parameters =  $1 \times 10^{14}$  parameters
- 16-bit = 2 bytes per parameter
- Total size =  $2 \times 10^{14}$  bytes = 200 TB

Convert to megabytes:

$$200 \text{ TB} = 200 \times 1024^2 \text{ MB} = 200 \times 1,048,576 = 209,715,200 \text{ MB}$$

2025 Prices (from table):

- SSD/Flash: \$0.00005 per MB
- DRAM: \$0.0040 per MB

Yearly Cost:

SSD:

$$209,715,200 \text{ MB} \times \$0.00005/\text{MB} = \$10,485.76$$

DRAM:

$$209,715,200 \text{ MB} \times \$0.0040/\text{MB} = \$838,860.80$$



**Final Answer:**

- SSD: \$10,486 per year
- DRAM: \$838,861 per year

### Prompt

Screen shot question 5(g)

This is question 5(g)

## Response



Let's do this step-by-step, keeping only the figures that matter for an **order-of-magnitude** answer.

### 1. Minimal latency for one forward pass (bandwidth-bound)

model size = 100 T params  $\times$  2 B = 200 TB

H200 bandwidth = 4.8 TB s<sup>-1</sup>

latency = 200 TB / 4.8 TB s<sup>-1</sup>  $\approx$  42 s

$\rightarrow \approx$  **40 s per forward pass** for GPT-6 on one GPU.

### 2. Autoregressive tokens per minute

One token = one full forward pass

tokens min<sup>-1</sup> = 60 s / 42 s  $\approx$  1.4

$\rightarrow \approx$  **1 token per minute** for GPT-6 on a single H200.

For GPT-3 (175 B params  $\rightarrow$  350 GB)

latency = 0.35 TB / 4.8 TB s<sup>-1</sup>  $\approx$  0.073 s

$\rightarrow \approx$  **14 tokens s<sup>-1</sup>  $\approx$  800 tokens min<sup>-1</sup>**.

### 3. Activation memory for **one token** in GPT-3

Per-layer activation volume (key quantities):

- 96 layers
- 96 heads  $\times$  128 d\_head = 12 288 attention head dims
- d\_model = 12 288 (so same number)
- 16-bit  $\rightarrow$  2 B per float

Activations we must keep per token:

- Q, K, V, O for each layer: 4  $\times$  d\_model = 49 152 floats
- Add residual & FFN ( $\approx$  4  $\times$  d\_model extra)  $\rightarrow$  total  $\approx$  10  $\times$  d\_model floats per layer

$\approx$  10  $\times$  12 288  $\times$  96 = 1.18  $\times$  10<sup>7</sup> floats

$\times$  2 B  $\rightarrow \approx$  **24 MB per token** (round to 20 MB).

### 4. Batch size where activations = model size

model memory = 350 GB

activations GB = (20 MB)  $\times$  B = 350 GB

B = 350 GB / 20 MB = 350 000 / 20  $\approx$  **17 500 tokens**

$\rightarrow$  Batch of  $\approx$  **20 000 tokens** makes activation traffic rival parameter traffic; beyond that, **activation bandwidth** becomes the new bottleneck.

## Annotation:

This question KIMI K2 answers perfectly.

### Prompt

Screen shot question 5(h)

This is question 5(h)

### Response



Let's work through the arithmetic carefully.

---

Base unit: 1 petaFLOP-day  
=  $1e15 \text{ FLOP/s} \times 86\,400 \text{ s}$   
= **8.64e19 FLOP**

H200 theoretical speed: 1.98 petaFLOP/s  
 $\Rightarrow$  1 petaFLOP-day needs  
 $8.64e19 \text{ FLOP} / 1.98e15 \text{ FLOP/s} \approx 4.36e4 \text{ s}$   
 $\approx$  **12.1 GPU-hours** (one H200).

Rental cost:  $12.1 \text{ h} \times \$1.50 \text{ h}^{-1} \approx$  **\$18 per petaFLOP-day**.

---

GPT-3: 3 640 petaFLOP-days

Cost @ 100 % utilisation:  $3\,640 \times \$18 \approx$  **\$65 000** (rental).

(With 30 % utilisation and  $\sim 2\times$  restarts the real bill would be  $\sim 6\times$  higher, but the question asks for the ideal figure.)

---

GPT-6: 1e30 FLOP (from 5c)

Convert to petaFLOP-days:

$1e30 \text{ FLOP} / 8.64e19 \text{ FLOP/petaFLOP-day} \approx 1.16e11 \text{ petaFLOP-days}$

Cost:  $1.16e11 \times \$18 \approx$  **\$2.1 trillion** (rental, 100 % utilisation).



---

Wall-clock time with 30 million H200s

Aggregate speed =  $30 \times 10^6 \times 1.98$  petaFLOP/s  
=  $5.94 \times 10^7$  petaFLOP/s  
=  $5.94 \times 10^{22}$  FLOP/s

Time =  $1 \times 10^{30}$  FLOP /  $5.94 \times 10^{22}$  FLOP/s  
 $\approx 1.68 \times 10^7$  s

$\approx$  **4 650 hours (~6.5 months)** of continuous, failure-free training.

---

Context check

- iPhone 1: \$0.15 B
- 5-nm chip: \$0.5 B
- A380 / Manhattan: ~\$20 B
- Apollo: \$178 B

Our **ideal** GPT-6 training bill (**\$2 T**) is **10× Apollo** and **100× the largest civilian R&D projects**.

Even spread across a 10 GW fleet of 30 M GPUs, the **wall-clock time is half a year**—a scale that only a **state-level actor** (or a coalition of the very largest tech firms) could contemplate.



#### Annotation:

This question KIMI K2 answers perfectly.

#### Prompt

Screen shot question 5(j)(k)(l)

This is question 5(j)(k)(l)

#### Response



Let's do each part step-by-step with **exact numbers** and **clear units**.

---

#### (j) Inference Cost

Given:

- GPT-5: 1 trillion params =  $1 \times 10^{12}$
- Inference: 2 FLOPs per param per token
- Tokens: 1 million =  $1 \times 10^6$
- H200 speed: 1.98 petaFLOP/s =  $1.98 \times 10^{15}$  FLOP/s
- H200 rental: \$1.50/hr
- OpenAI price: \$120 per 1M tokens
- Training cost: \$1 billion
- Essay: 1000 words  $\times$  1.4 tokens/word = 1400 tokens



FLOPs for 1M tokens:

$$2 \times 1 \times 10^{12} \times 1 \times 10^6 = 2 \times 10^{18} \text{ FLOPs}$$

Time on H200:

$$\frac{2 \times 10^{18}}{1.98 \times 10^{15}} \approx 1010 \text{ seconds} \approx 16.8 \text{ minutes}$$

Cost on H200:

$$1010 \text{ sec} \div 3600 \times \$1.50 \approx \$0.42$$

Tokens to recoup \$1B:

$$\frac{\$1 \times 10^9}{\$120} \times 1 \times 10^6 = 8.33 \times 10^{12} \text{ tokens}$$

Essays equivalent:

$$\frac{8.33 \times 10^{12}}{1400} \approx 6 \times 10^9 \text{ essays}$$

---

(k) Energetic Cost (Landauer Limit)

Given:

- Landauer limit:  $E = k_B T \ln 2 \approx 3 \times 10^{-21}$  J per bit erased
  - 1 FLOP: 64 bits  $\rightarrow$  32 bits  $\rightarrow$  32 bits erased
  - H200 speed: 1.98 petaFLOP/s =  $1.98 \times 10^{15}$  FLOP/s
  - Temperature: 300 K
- 

Energy per FLOP:

$$32 \times 3 \times 10^{-21} = 9.6 \times 10^{-20} \text{ J}$$

Power for 1.98e15 FLOP/s:

$$9.6 \times 10^{-20} \times 1.98 \times 10^{15} \approx 1.9 \times 10^{-4} \text{ W} = \mathbf{0.19 \text{ mW}}$$

Compare to actual:

- Actual: 700 W
- Landauer minimum: 0.00019 W
  - $\rightarrow$  Factor:  $\frac{700}{0.00019} \approx 3.7 \times 10^6$
  - $\rightarrow$  Real GPU is ~3.7 million times less efficient than the Landauer limit.





(l) Environmental Cost (GPT-3)

Given:

- GPT-3 CO<sub>2</sub>: 552 tCO<sub>2</sub>
  - Carbon price: \$112 per tCO<sub>2</sub>
  - A100 training cost: \$6M
  - Beef: 50 kg CO<sub>2</sub> per kg → 113 g per burger → 5.65 kg CO<sub>2</sub> per burger
- 

Carbon cost:

$$552 \times 112 = \$61,824$$

% of training cost:

$$\frac{61,824}{6 \times 10^6} \approx 1.03\%$$

Burgers equivalent:

$$\frac{552,000}{5.65} \approx 97,700 \text{ burgers}$$

---

**Annotation:**

This question KIMI K2 answers perfectly.

**Prompt**

Screen shot question 6(a)

This is question 6(a)

**Response**



We are **training only the 5 soft-prompt vectors** (tokens 1–5).

The rest of the model is **frozen**.

Cross-entropy is computed between the **model's predicted next-token logits** and the **target sequence**, which is the input shifted **right by one position**.

Input positions (what the model sees):

1–5 soft prompt

6–50 question

51–70 chain-of-thought (CoT)

71 answer token

72 newline

73–100 padding

Target positions (what we ask the model to predict):

1–4 soft-prompt tokens (but we **do not** have ground-truth labels for these)

5–49 question tokens (frozen embeddings, no label)

50–69 CoT tokens (we **do** want the soft prompt to help generate these)

70 answer token (critical supervisory signal)

71 newline (cheap to predict, but still part of the parsing format)

72–99 padding tokens (usually masked out)



Because **only the soft-prompt parameters are trainable**, gradients must flow **back through every position whose prediction we supervise**.

Therefore we include **all positions that we want the model to produce correctly**:

- CoT positions 50–69 (20 tokens)
- Answer position 70
- Newline position 71

We **mask out** positions 1–49 (no labels) and 72–99 (padding).

Hence the **loss is computed on target positions**:

50, 51, ..., 71

i.e. the **chain-of-thought tokens**, the **answer token**, and the **terminating newline token**.

### Annotation:

This question KIMI K2 answers perfectly.

### Prompt

Screen shot question 6(b)(c)(d)

This is question 6(b)(c)(d)

### Response



Let's do each part cleanly.

(b) How many **trainable** parameters?

Only the **5 soft-prompt vectors** are updated; everything else is frozen.

Each vector lives in the **token-embedding space**, dimension **E**.

Trainable params =

$$5 \times E$$

(All other symbols – S, V, H, L, D – are irrelevant for this count.)

(c) True / False with one-sentence justification

(i) **True**

Autoregressive GPT caches previous activations; prompt tokens never change, so their hidden states can be computed once and reused for every sequence in the batch.

(ii) **False**

A soft prompt has **E real numbers per token**, whereas a hard prompt is restricted to **V discrete choices**; the continuous space is strictly richer, so the **best** soft prompt can't do worse than the **best** hard prompt of the same length.

(iii) **False**

Full fine-tuning can **over-fit** the small prompt-tuning set; with limited data the **inductive bias** of soft prompts often gives **higher validation** accuracy.

(iv) **False**

Catastrophic forgetting applies to **shared model weights**; here the **GPT body is frozen** and only **task-specific soft prompts** change, so Task B's prompt is completely unaffected by Task A's tuning.

(d) Meta-learning soft prompts (MAML-style)

Learn a **single initialisation** for the soft-prompt vectors that can be **quickly adapted** (a few gradient steps) to any new task in the family.

### Training loop

1. Sample a task  $\tau$  from the family.
2. Copy the meta-prompt, fine-tune on  $\tau$ 's support set, yielding task-specific prompt  $\theta_\tau$ .
3. Evaluate  $\theta_\tau$  on  $\tau$ 's query set; back-propagate the loss **through the adaptation steps** into the **meta-prompt parameters**.
4. Repeat over many tasks; the meta-prompt becomes a **universal starting point**.

### At test time

Given an unseen task, plug in the meta-prompt, run **5–10 gradient updates** on its small support set, and use the resulting prompt for inference.

No change to the frozen GPT; only the **soft prompts** are meta-learned.

**Annotation:**

KIMI K2 answers perfectly except c(ii).

### Additional Prompt

double check(ii) If you compare the validation-set performance of the best possible K-token hard prompt to the best possible K-vector soft prompt, the soft-prompt performance will always be equal or better.

### Additional Response



The statement is **TRUE**, but the justification has to be slightly sharpened.

- A **K-token hard prompt** is a sequence of exactly K **discrete tokens** chosen from the vocabulary  $V$ .
- A **K-vector soft prompt** is a sequence of K **continuous vectors** in  $\mathbb{R}^E$ , which are **free parameters** and can implement **any real-vector embedding**, including the embeddings of the best hard prompt if that were optimal.

Hence the search space for the soft prompt **contains the hard-prompt space** as a subset.

Therefore the **global optimum** over soft prompts can only do **at least as well** as the best hard prompt; it can do **strictly better** whenever the optimal prompt is **not realisable as a single token embedding**.

So: **True** – the best possible K-vector soft prompt can **never under-perform** the best possible K-token hard prompt on the same validation set.