A Practical Guide to Large Language Model Training

Xiaobo Yang

1800010745@pku.edu.cn

November 14, 2024

Overview

Data format

Large Language Model (LLM)

Feed prompts in, generate responses by predicting the next token.

- Tokenization
  - Each word, subword, or character is mapped to an integer
- Next token prediction
  - Feature and label are shifted by one token
  - Feature: [batch size, seq len]
  - Label: [batch size, seq len]
- Data Source
  - Raw text data for pre-training
  - Instruction data for instruction tuning
  - Human preference data for RLHF

Transformer

- Core Components
  - Embedding
  - Attention Mechanism
  - MLP Layers
  - Layer Normalization
  - Residual Connections
- LLM Key Features
  - Decoder-only
  - Pre-LN

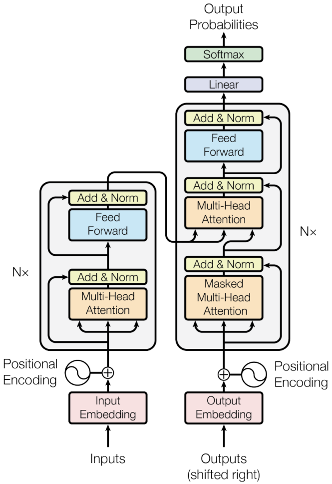


Figure 1: The Transformer - model architecture.

Embedding

- Token embedding
  - Maps each token x_i to a dense vector e_i
  - Learned during training
- Position embedding
  - Encodes position information
  - Maps each position i to a dense vector p_i
- Embedding concatenation
  - Combines token and position embeddings
  - $h_i = e_i + p_i$

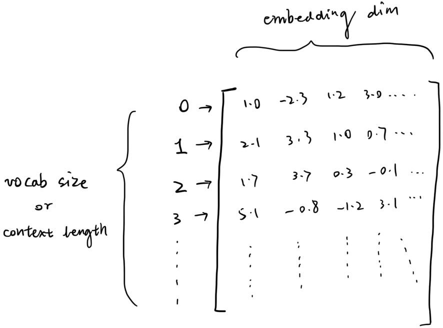


Figure: Embedding visualization

Attention Mechanism

- Self-attention computation
  - Maps each embedding h_i to query, key, value vectors $q_i^T = h_i^T W_Q, k_i^T = h_i^T W_K, v_i^T = h_i^T W_V$.
  - Those vectors constitute Query, Key, Value matrices $Q^T = [q_1, q_2, \ldots, q_n], K^T = [k_1, k_2, \ldots, k_n], V^T = [v_1, v_2, \ldots, v_n]$.
  - Masked attention for causal modeling $\text{mask}(QK^T)_{ij} = -\inf$ for $j > i$.
  - Attention scores: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{\text{mask}(QK^T)}{\sqrt{d_k}})V$, where d_k is queries and keys dimension.
  - Each embedding uses his query to match all keys, and uses the similarity to allocate weights to all values for predict the next token.
- Multi-head attention
  - Concatenate all heads: $o_i = \text{fc}(\text{Concat}(\text{Attention}_1(h_i), \text{Attention}_2(h_i), \ldots, \text{Attention}_h(h_i)))$.
  - Multiple parallel attention computations, different representation subspaces

MLP Layers

- Feed-forward network structure
  - Two linear transformations
  - Non-linear activation in between
- Dimensionality
  - Input/Output: model dimension
  - Hidden: typically 4x model dimension

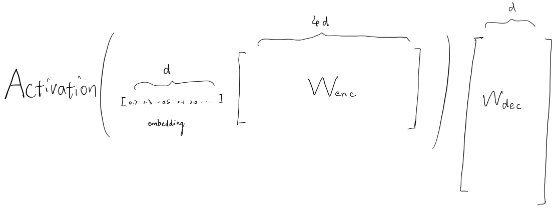


Figure: MLP architecture

Activation Functions

- ReLU
  - Simple and effective: $f(x) = \max(0, x)$
- GELU
  - Smooth approximation of ReLU: $\text{GELU}(x) = x\Phi(x) = xP(X \leq x)$
  - $\Phi(x)$ is the cumulative distribution function of the standard normal distribution
  - Smooth derivative near zero

$$\frac{d}{dx}\text{GELU}(x) = \Phi(x) + x\phi(x) = \Phi(x) + x\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$$

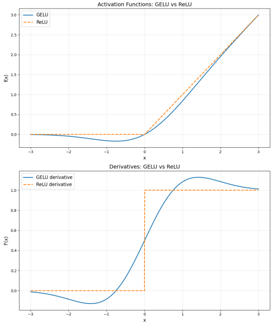


Figure: GELU vs ReLU

Normalization

- Normalized input

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta,$$

  where γ and β are learnable parameters ($4dL + 2d$ parameters). Input has batch size B, sequence length T, and feature dimension D.

- Batch normalization: normalizes across batch dimension $(B, \_, D)$, unstable for small batch size.
- Layer normalization: normalizes across feature dimension $(\_, \_, D)$, independent of batch size, low communication for parallel training(e.g. DDP).

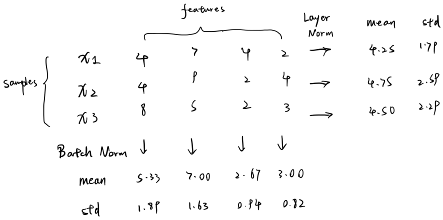
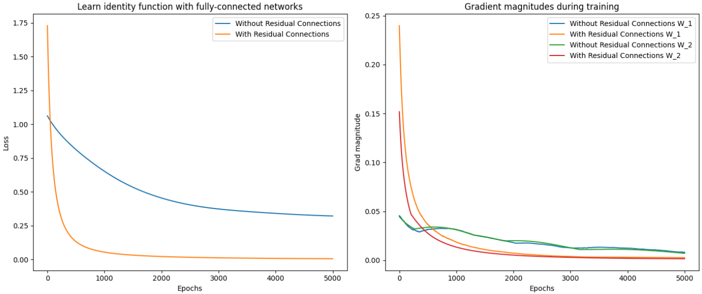


Figure: Layer Norm vs Batch Norm

Residual Connections

- Skip connections
  - Connects layer input directly to output: $y = F(x) + x$
- Benefits
  - Mitigates vanishing gradients, enables training of very deep networks
  - Learn residual functions instead of the original function
  - Easier to learn identity function



Loss function: Cross Entropy

- Measure the distance between the predicted next token distribution and the true distribution

$$H(p, q) = E_{x \sim p}[-\log q(x)] = -\sum_{x} p(x) \log q(x)$$

- Let $X \in \mathbb{R}^{B \times T}$ be the feature with batch size B and sequence length T, and $Y \in \mathbb{R}^{B \times T}$ be the label, the loss function is defined as:

$$\mathcal{L}(X, Y) = H(Y, p_\theta(X)) = -\frac{1}{B \cdot T} \sum_{i=1}^{B} \sum_{j=1}^{T} \log[p_\theta(X_{ij})]_{Y_{ij}}$$

- Equivalent to Maximum Likelihood Estimation (MLE)

Tokenizer Training

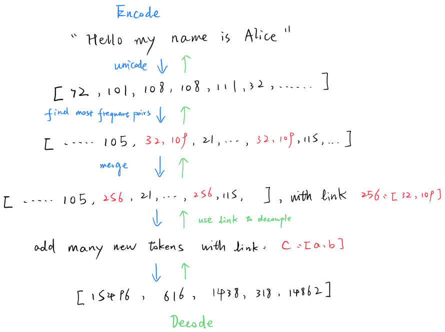


Figure: BPE algorithm

- Byte pair encoding(BPE) algorithm
  - Starts with base vocabulary (0-255)
  - Merges most frequent pairs into new tokens
  - Use regular expression to cut off unnecessary pairs
  - Training data may not be included in the pre-training phase.
- Compression vs. Training difficulty trade-off
  - Higher compression = more context in same length
  - But harder to train compressed tokens

Pre-training

- Raw text processing
  - Unstructured text data
  - Minimal cleaning required
  - Cost major portion of training time
- Token coverage challenges
  - Some tokens may be undertrained
  - Can lead to unexpected behaviors

Figure: Undertrained tokens with weird behaviors

Instruction Tuning

- Structured dialogue format
  - Assistant/User turn-taking
  - Specific formatting requirements
- LoRA adaptation
  - Freezes base parameters
  - Trains low-rank approximation

$$W = W_{\text{frozen}} + AB,$$

  where $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{r \times k}$ are low rank matrices.
  - Efficient fine-tuning approach

A: I'm worried about something.
B: What's that?
A: Well, I have to drive to school for a meeting this morning, and I'm going to end up getting stuck in rush-hour traffic.
B: That's annoying, but nothing to worry about. *Just breathe deeply when you feel yourself getting upset*.
A: Ok, I'll try that.
B: Is there anything else bothering you?
A: Just one more thing. A school called me this morning to see if I could teach a few classes this weekend and I don't know what to do.
B: Do you have any other plans this weekend?
A: I'm supposed to work on a paper that'd due on Monday.
B: *Try not to take on more than you can handle*.
A: You're right. I probably should just work on my paper. Thanks!

Figure: DailyDialog dataset

Human Preference Data Collection

1. Generate response pairs y, y';
2. Human annotators rank preferences $y_w > y_l$;
3. Build comparison dataset $\mathcal{D} = \{(y_w^i, y_l^i)\}_{i=1}^n$.

Figure: GPT Preference Visualization

Reward Model Training

- Bradley-Terry model implementation

$$\max_{\theta} P_\theta(y_w > y_l) = \frac{\exp(r_\theta(y_w))}{\exp(r_\theta(y_w)) + \exp(r_\theta(y_l))},$$

  where $r_\theta(y)$ is the reward score of response y.
- Architecture
  - Base LLM model with additional linear head
  - Scalar reward output
  - Loss function: binary cross entropy

$$L_{\text{reward}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(r_\theta(y_w^i) - r_\theta(y_l^i)),$$

  where $y_{ij} = 1$ if $y_w > y_l$ otherwise 0, $\sigma(x)$ is the sigmoid function.

RL Training: PPO

- Optimization objective:

$$\max_{\theta} E_{x\sim\mathcal{D}_{\text{prompt}}} E_{y\sim\pi_\theta(\cdot|x)} \left[r(y|x) - \beta \text{KL}(\pi_\theta(y|x), \pi_{\text{ref}}(y|x)) \right],$$

  where $r(y|x) = \sum_t \gamma^t r(y_t|x, y_{<t})$ are discounted rewards of each token in the response sequence, $KL(\pi_\theta(y|x), \pi_{\text{ref}}(y|x)) = \sum_t E_{y_t\sim\pi_\theta(\cdot|x,y_{<t})} \log\left(\frac{\pi_\theta(y_t|x,y_{<t})}{\pi_{\text{ref}}(y_t|x,y_{<t})}\right)$ are the KL divergences between the policy model and the reference model.
- Four-model system architecture:
  - Policy model $\pi_\theta(y|x)$ (actively trained), an LLM;
  - Reference model $\pi_{\text{ref}}(y|x)$ (frozen), an LLM;
  - Value model $v_\mu(y|x)$ (actively trained), an LLM + linear head;
  - Reward model $r(y|x)$ (frozen), an LLM + linear head.

PPO Loss Function I

Let T be the number of tokens in the response sequence. For LLM, states s_t are defined by the previous tokens $[x, y_{<t}]$ and actions a_t are defined by the next token y_t.

1. Merge the reward and KL penalty: $r(y|x, y_{<t}) \leftarrow r(y|x, y_{<t}) - \beta \mathrm{KL}_t$;
2. Policy gradient:

$$
\begin{aligned}
&\nabla_\theta E_{y \sim \pi_\theta}[r(y)] = E[\nabla_\theta \log \pi_\theta(y) \cdot r(y)] \\
&= \sum_{t=1}^{T} E_y \left[\nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \sum_{k=t}^{T} \gamma^k r(y_k|x, y_{<k}) \right] \\
&= \sum_{t=1}^{T} E_{y_{<t}} E_{y_t|y_{<t}} \left[\nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \left(E_{y_{>t}|y_{\leq t}} \left[\sum_{k=t}^{T} \gamma^k r(y_k|x, y_{<k}) \right] - E_{y_{\geq t}|y_{<t}} \left[\sum_{k=t}^{T} \gamma^k r(y_k|x, y_{<k}) \right] \right) \right] \\
&:= \sum_{t=1}^{T} E_{y_{<t}} E_{y_t|y_{<t}} \left[\nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \cdot \gamma^t \cdot (Q^{\pi_\theta}(y_t, [x, y_{<t}]) - V^{\pi_\theta}([x, y_{<t}])) \right] \\
&:= \sum_{t=1}^{T} E_{y_{<t}} E_{y_t|y_{<t}} \left[\nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \cdot \gamma^t \cdot A^{\pi_\theta}(a_t, s_t) \right]
\end{aligned}
$$

   where $A_t := A^{\pi_\theta}(a_t, s_t) = Q^{\pi_\theta}(a_t, s_t) - V^{\pi_\theta}(s_t)$ is the advantage function by standard definition of RL. We will use a neural network $v_\mu(s)$ to estimate the value function $V^{\pi_\theta}(s)$.
3. Estimate policy gradient:

$$
\widehat{\nabla}_\theta = \frac{1}{B} \sum_{i=1}^{B} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t^i|x^i, y_{<t}^i) \cdot \gamma^t \cdot \hat{A}_t(x^i, y^i), \quad y^i \sim \pi_{\theta_{\text{old}}}(\cdot|x^i) \text{ in last iteration;}
$$

PPO Loss Function II

4. Estimate KL divergence by token-level sampling: $\widehat{\mathrm{KL}} = \sum_{t=1}^{T} \widehat{\mathrm{KL}}_t = \sum_{t=1}^{T} \log\left(\frac{\pi_\theta(y_t^i|x^i, y_{<t}^i)}{\pi_{\mathrm{ref}}(y_t^i|x^i, y_{<t}^i)}\right)$;
5. Estimate advantage function $\hat{A}_t$ using Generalized Advantage Estimation (GAE):
   - Accumulation(low bias but high variance): $\hat{A}_t(x, y) = \sum_{k=t}^{T} \gamma^{k-t} r(y_k|x, y_{<k}) - v_\mu(s_t)$;
   - TD residual(low variance but high bias): $\hat{A}_t(x, y) = r(y_t|x, y_{<t}) + \gamma v_\mu(s_{t+1}) - v_\mu(s_t)$.

   Consider interpolation with multi-step TD residuals:

$$
\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma v_\mu(s_{t+1}) - v_\mu(s_t); \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 v_\mu(s_{t+2}) - v_\mu(s_t); \\
\hat{A}_t^{(3)} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 v_\mu(s_{t+3}) - v_\mu(s_t); \\
\hat{A}_t^{(k)} &= r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k v_\mu(s_{t+k}) - v_\mu(s_t).
\end{aligned}
$$

   GAE use λ-exponentially weighted of infinite multi-step TD residuals:

$$
\begin{aligned}
\hat{A}_t(x, y) &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \cdots + \lambda^{k-1}\hat{A}_t^{(k)} + \cdots\right) \\
&= \delta_t + \gamma\lambda\delta_{t+1} + \cdots + (\gamma\lambda)^k \delta_{t+k} + \cdots,
\end{aligned}
$$

   where $\delta_t = r(y_t|x, y_{<t}) + \gamma v_\mu(s_{t+1}) - v_\mu(s_t)$. Take $\hat{A}_t = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \delta_{t+k}$ for finite horizon T.

PPO Loss Function III

6. Maximize the advantage (Actor loss):

$$\max_{\theta} L_{policy}(\theta) = \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{T}\hat{A}_t^i \cdot \frac{\pi_\theta(y_t^i|x^i, y_{<t}^i)}{\pi_{\theta_{\text{old}}}(y_t^i|x^i, y_{<t}^i)} \approx E_{y\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T}\hat{A}_t\right];$$

7. Minimize the value function approximation error (Critic loss):

$$\min_{\mu} L_{value}(\mu) = \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{T}(v_\mu([x^i, y_{<t}^i]) - R_t(x^i, y^i))^2,$$

where $R_t(x^i, y^i) = \hat{A}_t(x^i, y^i) + v_{\mu_{\text{old}}}([x^i, y_{<t}^i])$ is the returns in last iteration.

PPO Loss Function

$$L_{PPO}(\theta, \mu) = -L_{policy}(\theta) + c \cdot L_{value}(\mu),$$

RL Training: DPO

1. Solve PPO optimization problem:

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

2. Reparameterize the reward model:

$$r(x,y) = \beta\log\frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta\log Z(x)$$

3. Plug in the reparameterized reward model into reward model training loss function $L_{reward} = -E[\log\sigma(r(y_w|x) - r(y_l|x))]$:

DPO Loss Function

$$L_{\text{DPO}}(\pi_\theta;\pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)}\right)\right]$$

DPO Loss Function

Solve PPO optimization problem:

$$\max_{\pi} E_{y\sim\pi(\cdot|x)}\left[r(x,y)-\beta\log\left(\frac{\pi(y|x)}{\pi_{ref}(y|x)}\right)\right] \qquad \text{subject to} \int \pi(y|x)dy = 1.$$

1. Construct Lagrangian:

$$\begin{aligned} L(\pi,\lambda) &= E_{y\sim\pi(\cdot|x)}\left[r(x,y)-\beta\log\left(\frac{\pi(y|x)}{\pi_{ref}(y|x)}\right)\right]+\lambda\left[\int \pi(y|x)dy-1\right] \\ &= \int\left(r(x,y)-\beta\log\left(\frac{\pi(y|x)}{\pi_{ref}(y|x)}\right)\right)\pi(y|x)dy+\lambda\left(\int\pi(y|x)dy-1\right). \end{aligned}$$

2. Apply KKT conditions:

$$0=\nabla_q L=\int\nabla_\pi\left[\left(r-\beta\log\frac{\pi}{\pi_{ref}}+\lambda\right)\pi\right]dy=\int\left(-\beta+r-\beta\log\frac{\pi}{\pi_{ref}}+\lambda\right)(\nabla_\pi\pi)(y)dy.$$

   Therefore:

$$\forall z \quad 0=\partial_{q(z)}L=\int\left(-\beta+r-\beta\log\frac{\pi}{\pi_{ref}}+\lambda\right)\delta(z-y)dy=-\beta+r(x,z)-\beta\log\frac{\pi(z|x)}{\pi_{ref}(z|x)}+\lambda.$$

   Thus, $r(x,y)+\beta\log\pi_{ref}(y|x)-\beta\log\pi^*(y|x)+\beta+\lambda^*=0$, which gives:

$$\pi^*(y|x)\propto\pi_{ref}(y|x)e^{\frac{1}{\beta}r(x,y)}.$$

Scaling Laws

- Empirical observation
$$L(N,D) = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + E,$$
where A, B, α, β, E are observed constants, N is the number of model parameters D is number of trained tokens.
- FLOPs Estimation: $C \approx 6ND$ ($2ND$ fwd + $4ND$ bwd)
- Optimal allocation given FLOPs: $(N_{\text{opt}}(C), D_{\text{opt}}(C)) = \arg\min_{6ND=C} L(N,D)$
$$N_{\text{opt}}(C) = G\left(\frac{C}{6}\right)^a, \quad D_{\text{opt}}(C) = G^{-1}\left(\frac{C}{6}\right)^b,$$
where
$$G = \left(\frac{\alpha A}{\beta B}\right)^{\frac{1}{\alpha+\beta}}, \quad a = \frac{\beta}{\alpha+\beta}, \quad b = \frac{\alpha}{\alpha+\beta}.$$

Scaling Laws

- Best performance scaling

$$L(C) = L(N_{\text{opt}}(C), D_{\text{opt}}(C)) \approx O(C^{-\gamma}).$$

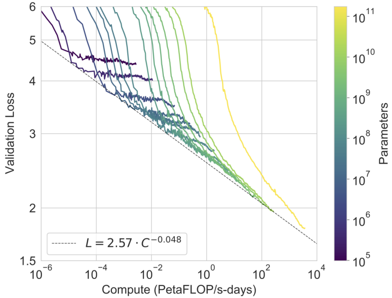


Figure: Compute-optimal frontier from GPT-3 paper

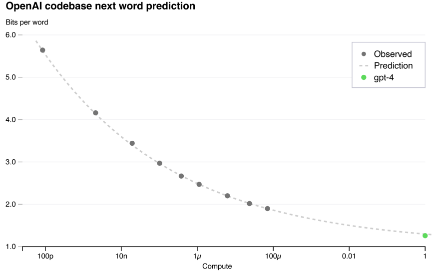


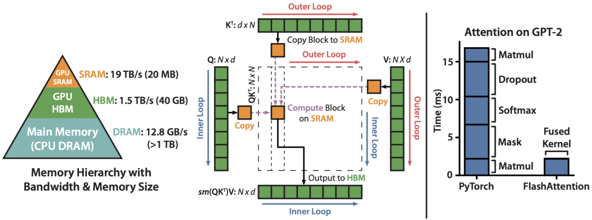
Figure: Scaling laws with GPT-4

Parallelization

- Data Parallel
  - Split batch across devices
  - Full model copy per device
  - Gradient sync and averaging (AllReduce)
- Model Parallel
  - Split model layers across devices
  - Reduce per-device memory usage
- Tensor Parallel
  - Split individual tensors across devices
  - Row/Column parallel

Flash Attention

- Acceleration in both training and inference by fused kernel.
- Decomposes parallel computation into for-loops (softmax decomposition formula):
$$\mathrm{softmax}([x_1; x_2]) = [\alpha_1 \mathrm{softmax}(x_1); \alpha_2 \mathrm{softmax}(x_2)],$$
where $\alpha_i = \frac{\sum_j \exp(x_{ij})}{\sum_k \exp(x_k)}$.
- Loops execute on much faster memory (SRAM) and significantly reduces I/O overhead for better speed.



KV Cache

- Acceleration in inference by reducing memory footprint and computational complexity.
- KV Cache
  - To predict next token only need to use the latest query to lookup against the cached keys and values.
  - Reduce complexity from $O(T^2)$ to $O(T)$.
- Cache reduction
  - Multiple-query attention (MQA), $1/n_{\text{head}}$ KV cache;
  - Grouped-query attention (GQA), $n_{\text{group}}/n_{\text{head}}$ KV cache.

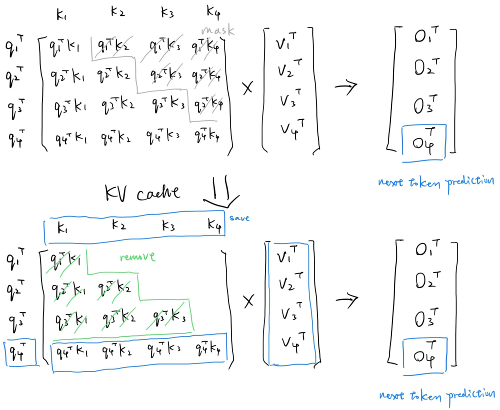


Figure: KV Cache

The End