# StatML: Introduction to Large Language Models

History, Transformer, Inference, and Post-training

Simon Vary

StatML @ Imperial College London

30 April 2026

# The plan for today

- Part 1: A (Very) Brief History of Probabilistic Language Modelling
  - Word embeddings, sequence-to-sequence, GPT
- Part 2: Decoder Transformer Architecture
  - Tokenizer
  - Multi-headed Attention
  - Positional Encodings
- Part 3: Architecture Variations, Inference, Post-Training
  - Architecture choices (shapes, normalizations, activations)
  - Inference: prefill vs decode, attention variants (MQA, GQA, MLA)
  - Post-training: SFT vs RLHF

# Part 1: A (Very) Brief History of Probabilistic Language Modelling

# (Probabilistic) Language Models

- Probability of word sequence $w_{1:T}$

$$p(w_{1:T}) = \prod_{t=1}^{T} p(w_t \mid w_{1:t-1})$$

where $w_{i:j} = \{w_i, w_{i+1}, \ldots, w_{j-1}, w_j\}$.

# (Probabilistic) Language Models

- Probability of word sequence $w_{1:T}$

$$p(w_{1:T}) = \prod_{t=1}^{T} p(w_t \mid w_{1:t-1})$$

  where $w_{i:j} = \{w_i, w_{i+1}, \ldots, w_{j-1}, w_j\}$.
- With $p(w_{1:T})$ we can:
  - rank sequences (likelihood / perplexity),
  - translation via $p(w_{1:T} \mid s_{1:T}) \propto p(s_{1:T} \mid w_{1:T})\, p(w_{1:T})$,
  - generate by sampling $p$.

# (Probabilistic) Language Models

- Probability of word sequence $w_{1:T}$

$$p(w_{1:T}) = \prod_{t=1}^{T} p(w_t \mid w_{1:t-1})$$

  where $w_{i:j} = \{w_i, w_{i+1}, \ldots, w_{j-1}, w_j\}$.
- With $p(w_{1:T})$ we can:
  - rank sequences (likelihood / perplexity),
  - translation via $p(w_{1:T} \mid s_{1:T}) \propto p(s_{1:T} \mid w_{1:T})\, p(w_{1:T})$,
  - generate by sampling $p$.

**Problem:** data sparsity + curse of dimensionality

- Huge vocabulary – many different words
- Long contexts – many different word sequences

# (Probabilistic) Language Models

- Probability of word sequence $w_{1:T}$

$$p(w_{1:T}) = \prod_{t=1}^{T} p(w_t \mid w_{1:t-1})$$

  where $w_{i:j} = \{w_i, w_{i+1}, \ldots, w_{j-1}, w_j\}$.
- With $p(w_{1:T})$ we can:
  - rank sequences (likelihood / perplexity),
  - translation via $p(w_{1:T} \mid s_{1:T}) \propto p(s_{1:T} \mid w_{1:T})\, p(w_{1:T})$,
  - generate by sampling $p$.

**Problem:** data sparsity + curse of dimensionality

- Huge vocabulary – many different words
- Long contexts – many different word sequences

- N-grams: Markov assumption

$$p(w_t \mid w_{1:t-1}) \approx p(w_t \mid w_{t-N+1:t-1})$$

  but it destroys long-range context.

# Early Word Embeddings **(Bengio et al., 2003)**

- Learn an embedding matrix $C \in \mathbb{R}^{V \times D}$ and a predictor $f_\Theta$

$$p(w_t \mid w_{1:t-1}) \approx f_\Theta\big(c_{I(w_{t-1})}, \ldots, c_{I(w_{t-N})}\big),$$

where $I(w) \in \{1, \ldots, V\}$ indexes the vocabulary and $c_{I(w)}$ is the corresponding row of $C$.

# Early Word Embeddings **(Bengio et al., 2003)**

- Learn an embedding matrix $C \in \mathbb{R}^{V \times D}$ and a predictor $f_\Theta$

$$p(w_t \mid w_{1:t-1}) \approx f_\Theta\big(c_{I(w_{t-1})}, \ldots, c_{I(w_{t-N})}\big),$$

  where $I(w) \in \{1, \ldots, V\}$ indexes the vocabulary and $c_{I(w)}$ is the corresponding row of $C$.

**Idea 1**: Smoothness $\Rightarrow$ generalization

- Discrete words have no notion of "small change"; vectors do.
- If "cat" and "dog" have close embeddings and $f_\Theta$ is smooth, then probability mass transfers to *neighbors* in sentence space. *Cat is walking on a road* $\approx$ *Dog is walking on a road*.

# Early Word Embeddings **(Bengio et al., 2003)**

- Learn an embedding matrix $C \in \mathbb{R}^{V \times D}$ and a predictor $f_\Theta$

$$p(w_t \mid w_{1:t-1}) \approx f_\Theta\big(c_{I(w_{t-1})}, \ldots, c_{I(w_{t-N})}\big),$$

where $I(w) \in \{1, \ldots, V\}$ indexes the vocabulary and $c_{I(w)}$ is the corresponding row of $C$.

## **Idea 1**: Smoothness ⇒ generalization

- Discrete words have no notion of "small change"; vectors do.
- If "cat" and "dog" have close embeddings and $f_\Theta$ is smooth, then probability mass transfers to *neighbors* in sentence space. *Cat is walking on a road* ≈ *Dog is walking on a road*.

## Training and early limitations

- **Approach:** Next token prediction from $N$ previous; optimize cross-entropy over $V$ words.
- **The N-gram Era:** Pre-GPU compute and shallow architectures:
  - *"[N]eural probabilistic language models remain far less widely used than N-gram models due to their notoriously long training times..."* (Mnih & Teh, 2012)
- Overcoming required new hardware (GPUs) and architectures (RNNs, Transformers).

# word2vec **(Mikolov et al., 2013)**

- **The AlexNet (2012):** Scale efficient architecture + GPU training.
- **But** early neural language models too expensive for large vocabularies.

# word2vec **(Mikolov et al., 2013)**

- **The AlexNet (2012):** Scale efficient architecture + GPU training.
- **But** early neural language models too expensive for large vocabularies.

**Idea 2:** Efficiency $\Rightarrow$ Train on massive data $\Rightarrow$ Rich representations

- **Simple architecture:** Remove non-linear hidden layers (only a log-linear model).
- **Trick for "cheap" softmax:** Compute probabilities only for the vocabulary $V$ that is important.

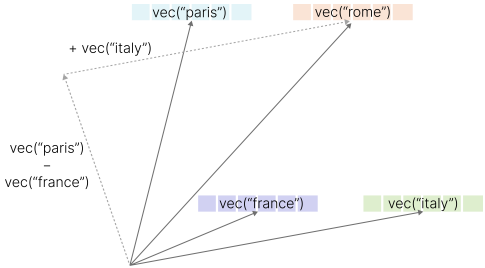
# word2vec **(Mikolov et al., 2013)**

- **The AlexNet (2012):** Scale efficient architecture + GPU training.
- **But** early neural language models too expensive for large vocabularies.

**Idea 2:** Efficiency ⇒ Train on massive data ⇒ Rich representations

- **Simple architecture:** Remove non-linear hidden layers (only a log-linear model).
- **Trick for "cheap" softmax:** Compute probabilities only for the vocabulary $V$ that is important.

- **Emergent Structure:** Simple dot products learned complex geometry.
- Analogies became linear math:
  $\text{vec(King)} - \text{vec(Man)} + \text{vec(Woman)} \approx \text{vec(Queen)}$




Linear structure emerges in the embedding space.

# Sequence-to-Sequence Models **(Sutskever et al., 2014)**

## **Problem:** Variable Length Context

- word2vec embeddings are learned using a *fixed context window*
- no mechanism to capture variable-length, sentence-level dependencies.

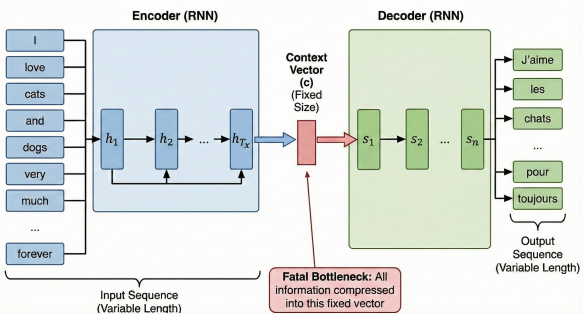
# Sequence-to-Sequence Models **(Sutskever et al., 2014)**

## **Problem:** Variable Length Context

- word2vec embeddings are learned using a *fixed context window*
- no mechanism to capture variable-length, sentence-level dependencies.

## **Idea 3:** Learn embeddings for variable length contexts

- **seq2seq:** Encoder–Decoder to learn a variable-length input $X$ to a variable-length output $Y$.



# Adaptive Context via Attention **(Bahdanau et al., 2014)**

**Problem:** Long sequences don't fit into a single state

- A single vector $c$ lacks the capacity for complex, long-range meaning.

# Adaptive Context via Attention **(Bahdanau et al., 2014)**

**Problem:** Long sequences don't fit into a single state

- A single vector $c$ lacks the capacity for complex, long-range meaning.

**Idea 4:** Don't try to compress into a single state

- Keep encoder states for *all* tokens and *let the decoder decide* what is relevant.
- Dynamic $c_i$ as a weighted sum of all token hidden states $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

# Adaptive Context via Attention **(Bahdanau et al., 2014)**

**Problem:** Long sequences don't fit into a single state

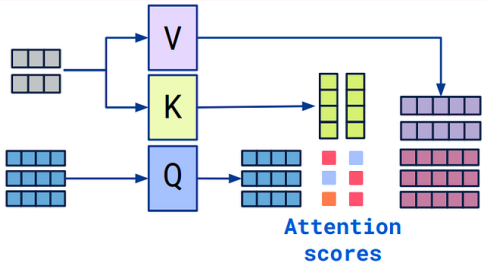
- A single vector $c$ lacks the capacity for complex, long-range meaning.

**Idea 4:** Don't try to compress into a single state

- Keep encoder states for *all* tokens and *let the decoder decide* what is relevant.
- Dynamic $c_i$ as a weighted sum of all token hidden states $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

**Translating to modern QKV (Cross-Attention):**

- **Query ($q_i$):** What the decoder wants ← Previous decoder state $s_{i-1}$
- **Key/Value ($k_j, v_j$):** What the encoder holds ← Encoder states $h_j$



# The Transformer **(Vaswani et al., 2017)**

- **Sequential representations:** RNNs decoders are inherently sequential ($h_t$ depends on $h_{t-1}$). This precludes parallelization within a sequence, strictly limiting training scale.
- **Idea 5:** *"Attention is All You Need."* Remove recurrence entirely.

# The Transformer **(Vaswani et al., 2017)**

- **Sequential representations:** RNNs decoders are inherently sequential ($h_t$ depends on $h_{t-1}$). This precludes parallelization within a sequence, strictly limiting training scale.
- **Idea 5:** *"Attention is All You Need."* Remove recurrence entirely.

## Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

# The Transformer **(Vaswani et al., 2017)**

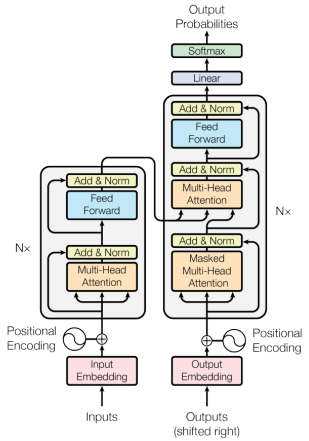
- **Sequential representations:** RNNs decoders are inherently sequential ($h_t$ depends on $h_{t-1}$). This precludes parallelization within a sequence, strictly limiting training scale.
- **Idea 5:** *"Attention is All You Need."* Remove recurrence entirely.

## Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

**Architectural changes:**

- **Self-Attention:** Remove RNN. Words attend to other words in the *same* sequence to build representations.
- **Positional Encoding:** Attention processes everything at once, must add positions.
- **Multi-Head:** Multiple attention representations in parallel.



# Generative Pre-Training **(Radford et al., 2019 - GPT-2)**

- **Massive data:** Use massive unlabeled web corpora (instead of labeled).
- **Decoder Only:** Train decoder-only Transformer using *next-word prediction*.

# Generative Pre-Training **(Radford et al., 2019 - GPT-2)**

- **Massive data:** Use massive unlabeled web corpora (instead of labeled).
- **Decoder Only:** Train decoder-only Transformer using *next-word prediction*.

## **The Objective:** Maximum Likelihood Estimation

Maximize the probability of the next token $w_t$ given all previous context $w_{<t}$:

$$\max_{\Theta} \sum_{t=1}^{T} \log p_{\Theta}(w_t \mid w_{1:t})$$

# Generative Pre-Training **(Radford et al., 2019 - GPT-2)**

- **Massive data:** Use massive unlabeled web corpora (instead of labeled).
- **Decoder Only:** Train decoder-only Transformer using *next-word prediction*.

## **The Objective:** Maximum Likelihood Estimation

Maximize the probability of the next token $w_t$ given all previous context $w_{<t}$:

$$\max_{\Theta} \sum_{t=1}^{T} \log p_{\Theta}(w_t \mid w_{1:t})$$

- **Allows for scale:** No human labeling is required.

# Generative Pre-Training **(Radford et al., 2019 - GPT-2)**

- **Massive data:** Use massive unlabeled web corpora (instead of labeled).
- **Decoder Only:** Train decoder-only Transformer using *next-word prediction*.

## **The Objective:** Maximum Likelihood Estimation

Maximize the probability of the next token $w_t$ given all previous context $w_{<t}$:

$$\max_{\Theta} \sum_{t=1}^{T} \log p_{\Theta}(w_t \mid w_{1:t})$$

- **Allows for scale:** No human labeling is required.
- **Takeaway:** Data scale $+$*pure* next word prediction $\rightarrow$ learn *syntax*, *semantics*, and *world knowledge* to perform translation, summarization, and QA.

# Generative Pre-Training **(Radford et al., 2019 - GPT-2)**

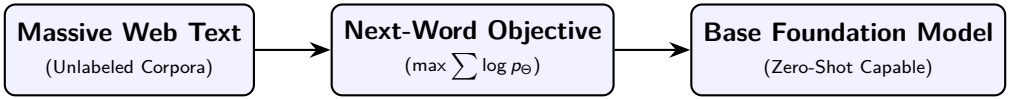
- **Massive data:** Use massive unlabeled web corpora (instead of labeled).
- **Decoder Only:** Train decoder-only Transformer using *next-word prediction*.

## **The Objective:** Maximum Likelihood Estimation

Maximize the probability of the next token $w_t$ given all previous context $w_{<t}$:

$$\max_{\Theta} \sum_{t=1}^{T} \log p_{\Theta}(w_t \mid w_{1:t})$$

- **Allows for scale:** No human labeling is required.
- **Takeaway:** Data scale $+$*pure* next word prediction $\rightarrow$ learn *syntax*, *semantics*, and *world knowledge* to perform translation, summarization, and QA.



# Alignment via RLHF **(Ouyang et al., 2022 - InstructGPT)**

- **The Problem:** Base models predict statistical likelihood, not human values (can be toxic, unhelpful, or ignore instructions).

# Alignment via RLHF **(Ouyang et al., 2022 - InstructGPT)**

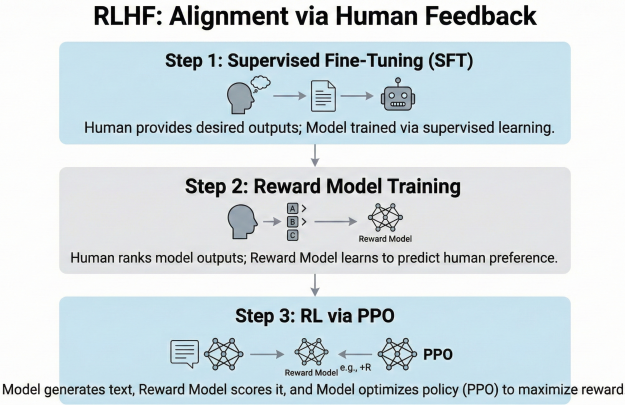
- **The Problem:** Base models predict statistical likelihood, not human values (can be toxic, unhelpful, or ignore instructions).
- **The Solution:** RLHF shifts the training objective from *likelihood* to *human preference*.

# Alignment via RLHF **(Ouyang et al., 2022 - InstructGPT)**

- **The Problem:** Base models predict statistical likelihood, not human values (can be toxic, unhelpful, or ignore instructions).
- **The Solution:** RLHF shifts the training objective from *likelihood* to *human preference*.

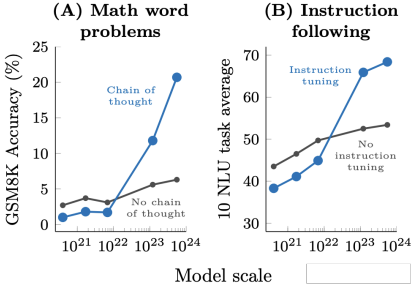
## Three steps:

1. **SFT:** Supervised fine-tuning on high-quality human demonstrations.
2. **Reward Model:** Train a "critic" network on human rankings to predict preference.
3. **RL (PPO):** Optimize the model's policy to maximize that reward score.



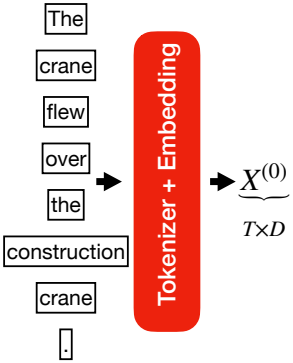
# Emergent Abilities of LLMs **(Wei et al., 2022)**

- **Emergent ability:** Not present at small scale.
- **Phase Shift:** A few-shot prompted tasks sharply works well for large models.
- **Measuring Scale:** Training FLOPS.
- **Unpredictability:** Cannot be predicted looking at scaling plots.



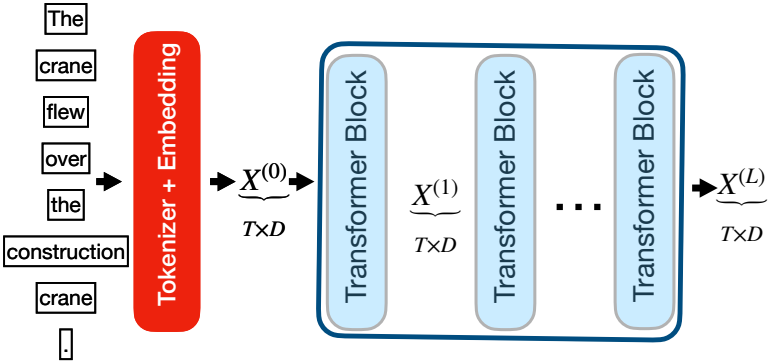
# Part 2: Decoder Transformer Architecture

# GPT2-like Decoder Transformer




Tokenizer + Embedding : $\Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

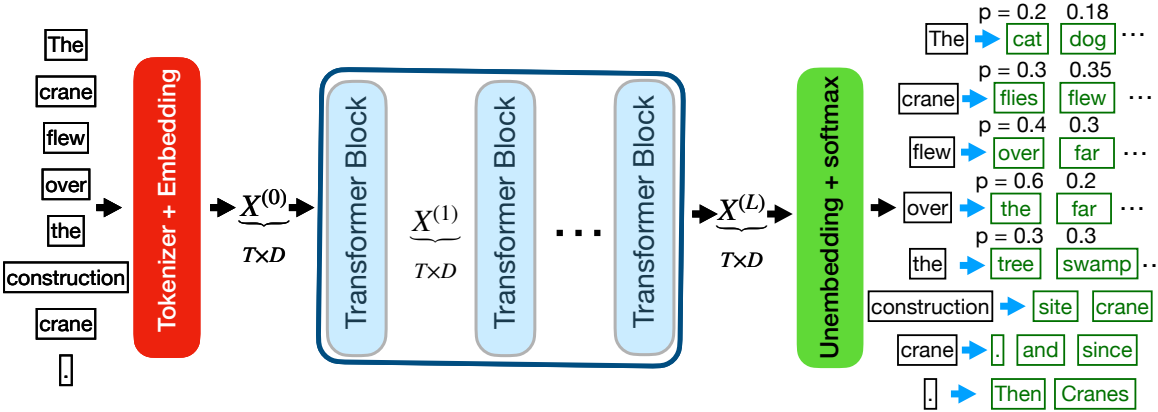
# GPT2-like Decoder Transformer




Tokenizer + Embedding : $\Sigma^* \rightarrow \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \rightarrow \mathbb{R}^{T \times D}$ sequence embeddings

# GPT2-like Decoder Transformer




Tokenizer + Embedding : $\Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ sequence embeddings

Unembedding + Softmax : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times |V|}$ probability over token vocabulary

# Step 1: Tokenizer + Embedding

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units $\rightarrow$ *tokenization*
- The set of possible tokens defines the model's **vocabulary**

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

## Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

## Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']
- **UTF-8 code level**, e.g., "café" → [99, 97, 102, 195, 169]

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

## Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']
- **UTF-8 code level**, e.g., "café" → [99, 97, 102, 195, 169]
- **Word-level**, e.g., "The dog runs fast." → ['The', 'dog', 'runs', 'fast', '.']

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

## Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']
- **UTF-8 code level**, e.g., "café" → [99, 97, 102, 195, 169]
- **Word-level**, e.g., "The dog runs fast." → ['The', 'dog', 'runs', 'fast', '.']
- **Subword-level**, e.g., *Byte Pair Encoding*, "happening" → ['happen', 'ing']

# Step 1: Tokenization, vocabulary vs. length tradeoff

## Question

What is the right granularity to discretize text at?

- Raw text must be converted into discrete units → *tokenization*
- The set of possible tokens defines the model's **vocabulary**

## Tokenization ideas

- **Character-level**, e.g., "dog" → ['d', 'o', 'g']
- **UTF-8 code level**, e.g., "café" → [99, 97, 102, 195, 169]
- **Word-level**, e.g., "The dog runs fast." → ['The', 'dog', 'runs', 'fast', '.']
- **Subword-level**, e.g., *Byte Pair Encoding*, "happening" → ['happen', 'ing']

## The tradeoff

**Vocabulary size** $V$ vs **Sequence length** $T$ (and attention cost $\sim T^2$).

# Step 1: Byte Pair Encoding (BPE): Learning the vocabulary

**Idea:** Learn the vocabulary from data

Start from a fine-alphabet (e.g. bytes) and iteratively *merge frequent adjacent pairs*.

# Step 1: Byte Pair Encoding (BPE): Learning the vocabulary

## **Idea:** Learn the vocabulary from data

Start from a fine-alphabet (e.g. bytes) and iteratively *merge frequent adjacent pairs*.

## BPE toy example)

**1. Corpus with counts:**

```
"hug": 10, "pug": 5, "pun": 12, "bun": 4,
"hugs": 5
```

**2. Split into characters:**

```
("h", "u", "g"): 10
("p", "u", "g"): 5
("p", "u", "n"): 12 ...
```

**3. Find the most frequent pair:**

The pair `("u", "g")` appears $10 + 5 + 5 = 20$ times.

**4. Merge Rule #1:** `"u" + "g"` $\rightarrow$ `"ug"`

**Updated Corpus:**

```
("h", "ug"): 10
("p", "ug"): 5
("p", "u", "n"): 12 ...
```

## BPE Algorithm

Initialize vocab $V$ with all base bytes/chars.

1. Count adjacent pairs $(a, b)$.
2. Pick the most frequent pair.
3. Create new token $c \leftarrow ab$.
4. Repeat $M$ times.

Final vocab size: $|V_{\text{initial}}| + M$.

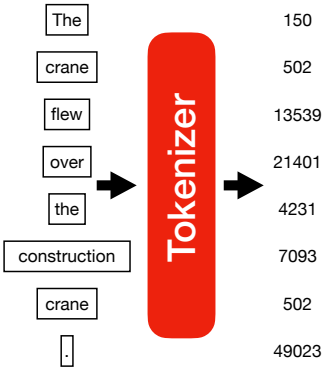
# Step 1: After tokenization: IDs $\rightarrow$ vectors

- Tokenizer outputs integer IDs: $\mathrm{tok}_t \in \{1, \ldots, V\}$.
- Show https://tiktokenizer.vercel.app/ or the Jupyter Notebook
- Convert IDs to vectors via an embedding table $E \in \mathbb{R}^{V \times d}$:

$$\text{one-hot}(\mathrm{tok}_t)^\top E \;=\; E_{\mathrm{tok}_t} \in \mathbb{R}^d \quad \text{(implemented as a lookup)}.$$

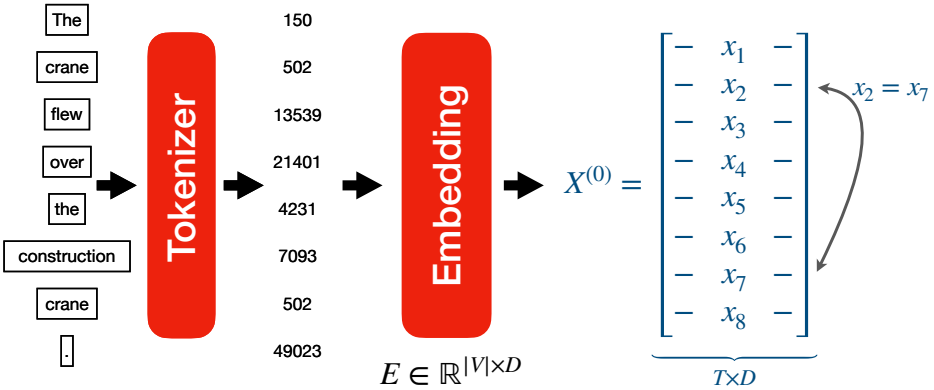
- Add positional information (absolute or relative) before attention.
- Transformer maps embeddings $\rightarrow$ logits in $\mathbb{R}^V$ for next-token prediction.

# Step 1: Tokenizer + Embedding




$$\text{Tokenizer} : \Sigma^* \rightarrow [1,\ldots,|V|]^T$$

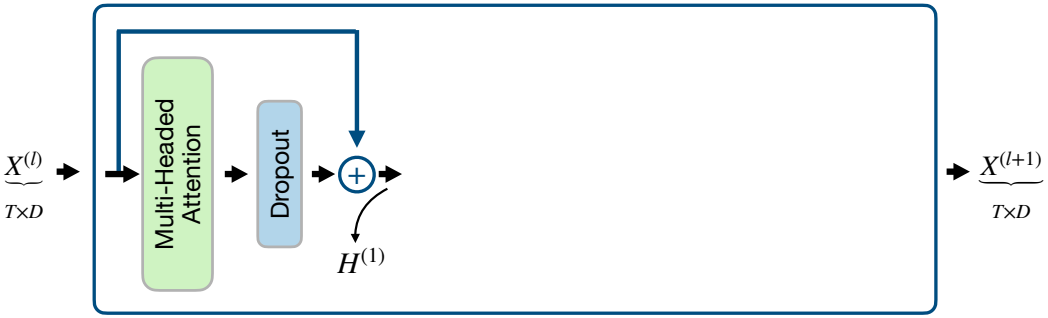
# Step 1: Tokenizer + Embedding




Tokenizer : $\Sigma^* \to [1,\ldots,|V|]^T$ Embedding : $[1,\ldots,|V|]^T \to \mathbb{R}^{T \times D}$

# Step 2: Transformer Decoder Block

# Step 2: Transformer (Decoder) Block



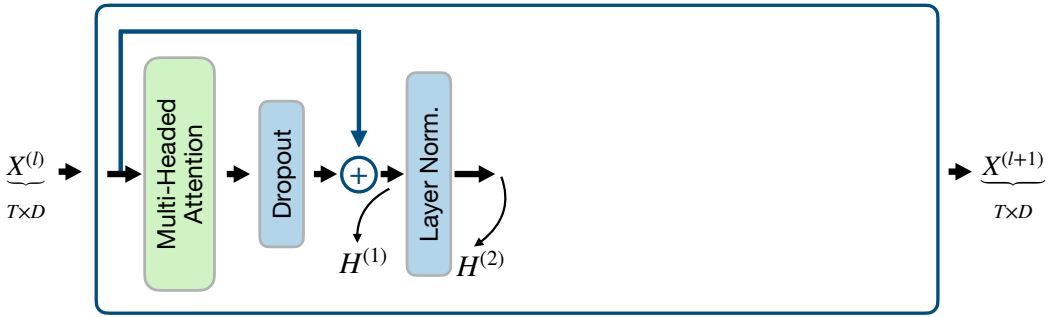

$\text{MHA} : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$ Multi-Headed Attention

$\text{Dropout} : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$ Set each entry to zero with prob. p

$\text{Residual connection} \oplus : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$ Addition with the input

$$H^{(1)} = \text{Dropout}_p\left(\text{MHA}(X^{(l)})\right) + X^{(l)} \in \mathbb{R}^{T\times D}$$

# Step 2: Transformer (Decoder) Block




$\mathrm{LN} : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$

Layer Norm. forces each token embedding (not the sequence) to have zero mean and st. dev 1

$$\underbrace{H^{(2)}}_{T\times D} = \mathrm{LN}_\varepsilon\left(H^{(1)}\right) = \left(\frac{H^{(1)} - \mathbf{1}^\top \mu_{[1:T]}}{\sqrt{\varepsilon + \mathbf{1}^\top \sigma^2_{[1:T]}}}\right) \cdot \gamma + \beta$$

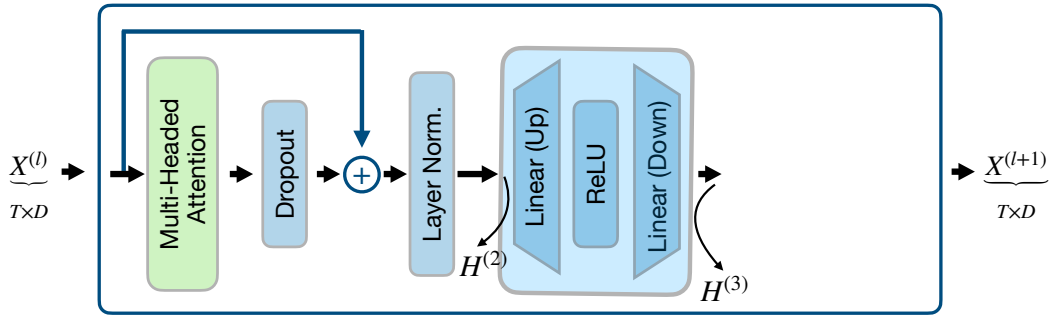
means: $\mu_{[1:T]} \in \mathbb{R}^T$

variances: $\sigma^2_{[1:T]} \in \mathbb{R}^T$

$\varepsilon = 10^{-6}$

learnable parameters: $\gamma, \beta$

# Step 2: Transformer (Decoder) Block




$\text{FFN} : \mathbb{R}^{T\times D} \to \mathbb{R}^{T\times D}$ Feed Forward Network

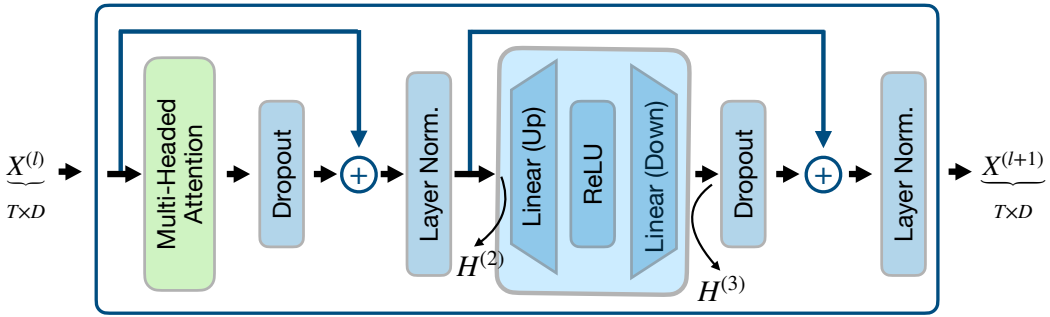
$$H^{(3)} = \text{FFN}(H^{(2)}) = \text{ReLU}(H^{(2)}W_1 + \mathbf{1}b_1^T)W_2 + \mathbf{1}b_2^T$$

$W_1 \in \mathbb{R}^{D\times 4D}$, $b_1 \in \mathbb{R}^{4D}$

$\text{ReLU}(x) = \max\{0, x\}$

$W_2 \in \mathbb{R}^{4D\times D}$, $b_2 \in \mathbb{R}^{D}$

# Step 2: Transformer (Decoder) Block




$$X^{(l+1)} = \mathrm{LN}_{\varepsilon}\left(\mathrm{Dropout}_{p}\left(H^{(3)}\right) + H^{(2)}\right) \in \mathbb{R}^{T \times D}$$

# Step 2: Multi-Headed Attention (MHA) with causal mask

$\underbrace{X^{(l)}}_{T \times D}$

The 3 aspects of every token

- Query $\left(Q \in \mathbb{R}^{T \times D}\right)$: “What information I look for?”
- Key $\left(K \in \mathbb{R}^{T \times D}\right)$: “What information do I offer?”
- Value $\left(V \in \mathbb{R}^{T \times D}\right)$: “What is my meaning?”

$\underbrace{\mathrm{MHA}(X^{(l)})}_{T \times D}$

# Step 2: Multi-Headed Attention (MHA) with causal mask

$$\underbrace{X^{(l)}}_{T\times D}$$

$$X^{(l)}W_K \to K = \underbrace{\begin{bmatrix} - & k_1 & - \\ - & k_2 & - \\ & \vdots & \\ - & k_T & - \end{bmatrix}}_{T\times D}$$

$$X^{(l)}W_Q \to Q = \underbrace{\begin{bmatrix} - & q_1 & - \\ - & q_2 & - \\ & \vdots & \\ - & q_T & - \end{bmatrix}}_{T\times D}$$

$$X^{(l)}W_V \to V = \underbrace{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \\ - & v_T & - \end{bmatrix}}_{T\times D}$$

The 3 aspects of every token

- Query $\left(Q \in \mathbb{R}^{T\times D}\right)$: "What information I look for?"
- Key $\left(K \in \mathbb{R}^{T\times D}\right)$: "What information do I offer?"
- Value $\left(V \in \mathbb{R}^{T\times D}\right)$: "What is my meaning?"

Parameters $W_Q, W_K, W_V \in \mathbb{R}^{D\times D}$

$$\underbrace{\mathrm{MHA}(X^{(l)})}_{T\times D}$$

# Step 2: Multi-Headed Attention (MHA) with causal mask

$\underbrace{X^{(l)}}_{T\times D}$

$X^{(l)}W_K \to K = \underbrace{\begin{bmatrix} - & k_1 & - \\ - & k_2 & - \\ & \vdots & \\ - & k_T & - \end{bmatrix}}_{T\times D}$

$X^{(l)}W_Q \to Q = \underbrace{\begin{bmatrix} - & q_1 & - \\ - & q_2 & - \\ & \vdots & \\ - & q_T & - \end{bmatrix}}_{T\times D}$

$X^{(l)}W_V \to V = \underbrace{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \\ - & v_T & - \end{bmatrix}}_{T\times D}$

The 3 aspects of every token

- Query $\left(Q \in \mathbb{R}^{T\times D}\right)$: "What information I look for?"
- Key $\left(K \in \mathbb{R}^{T\times D}\right)$: "What information do I offer?"
- Value $\left(V \in \mathbb{R}^{T\times D}\right)$: "What is my meaning?"

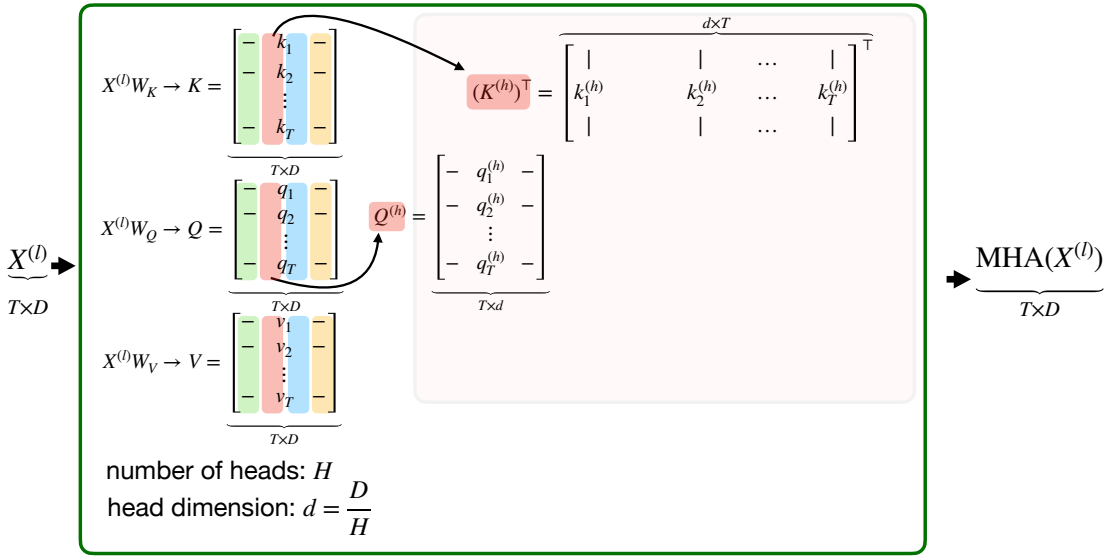
Parameters $W_Q, W_K, W_V \in \mathbb{R}^{D\times D}$

Multiple roles of every token

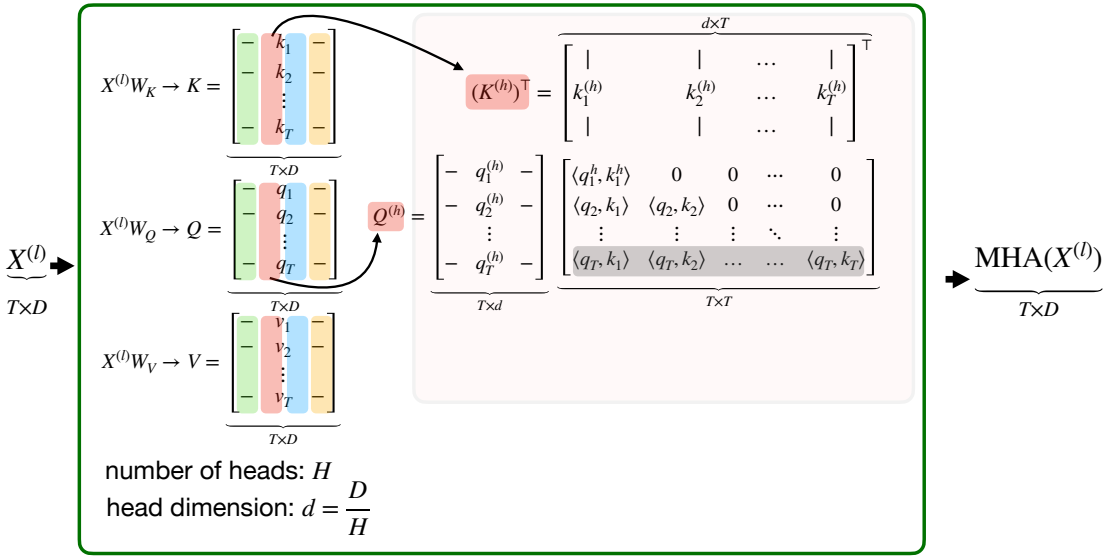
- "Grammar", "Semantic", "Punctuation"
- Split to $H$ different $d = D/H$-dim. "heads"
- $\left\{Q^{(h)}, K^{(h)}, V^{(h)}\right\}_{h=1}^{H}$, where $Q^{(h)}, K^{(h)}, V^{(h)} \in \mathbb{R}^{T\times d}$

$\underbrace{\mathrm{MHA}(X^{(l)})}_{T\times D}$

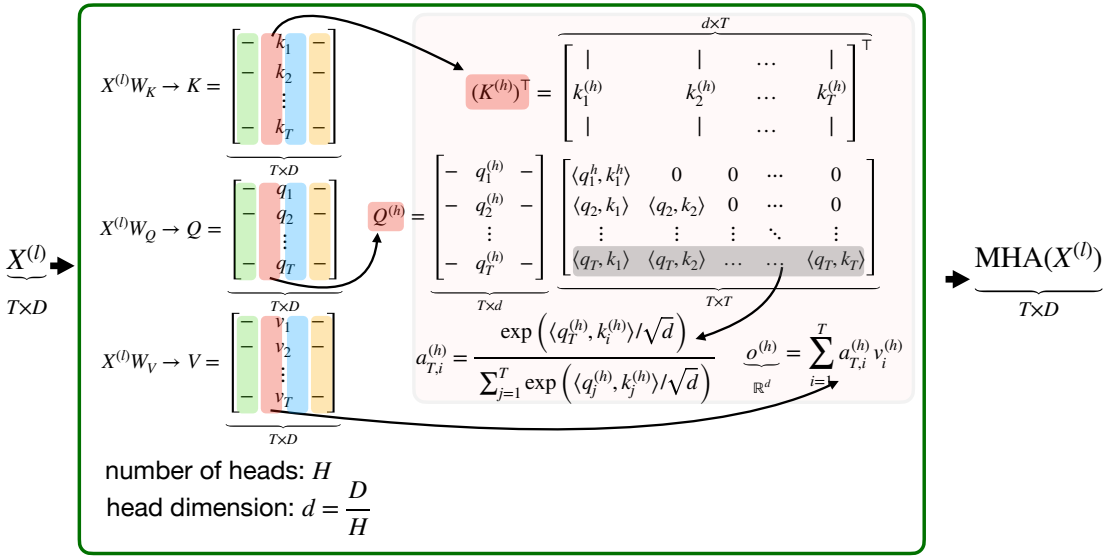
# Step 2: Multi-Headed Attention (MHA) with causal mask



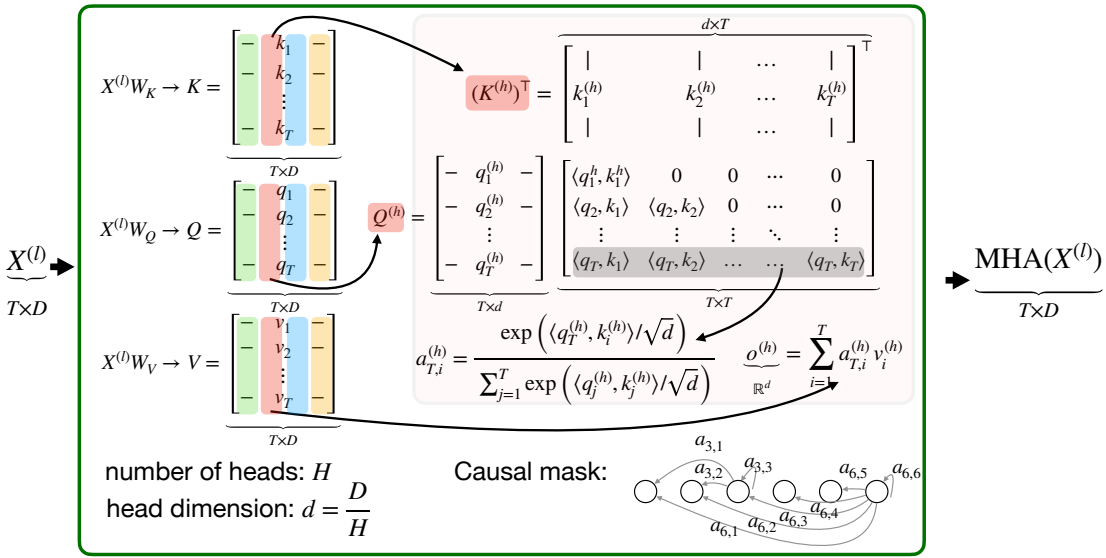
# Step 2: Multi-Headed Attention (MHA) with causal mask



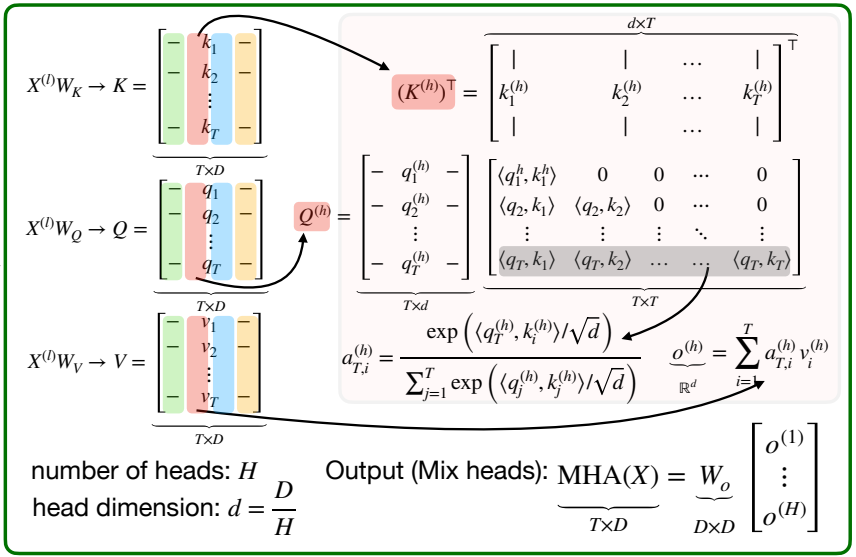
# Step 2: Multi-Headed Attention (MHA) with causal mask



# Step 2: Multi-Headed Attention (MHA) with causal mask

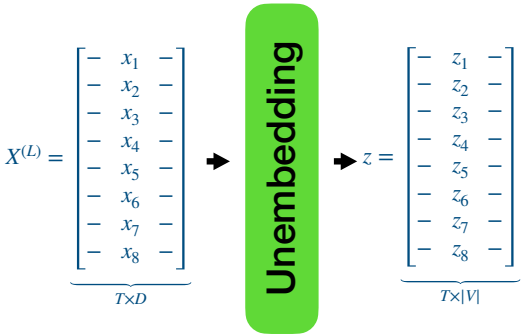


# Step 2: Multi-Headed Attention (MHA) with causal mask



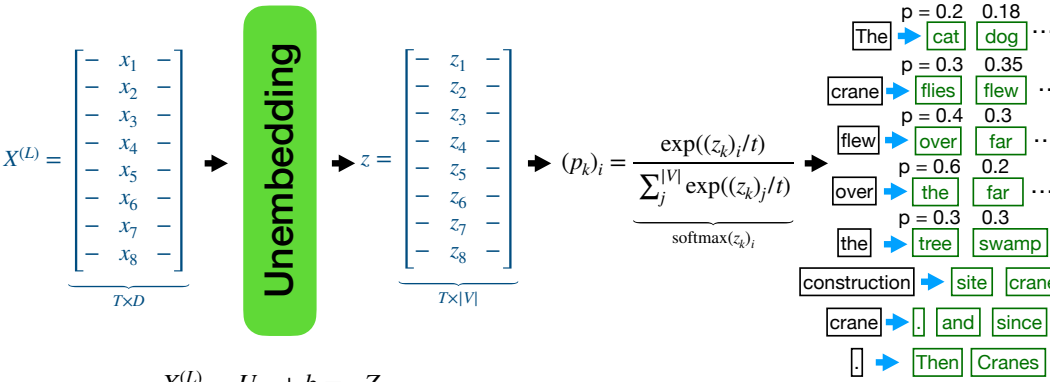
## Step 3: Unembedding + softmax + training

# Step 3: Unembedding + softmax + training

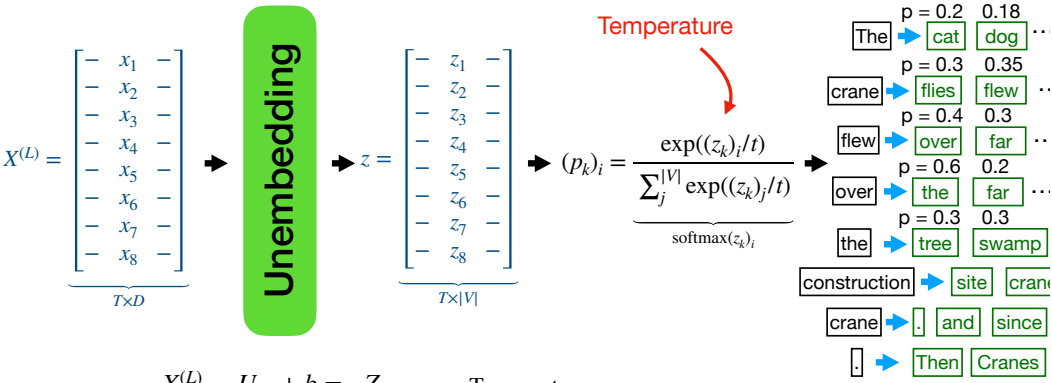



$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

# Step 3: Unembedding + softmax + training



# Step 3: Unembedding + softmax + training

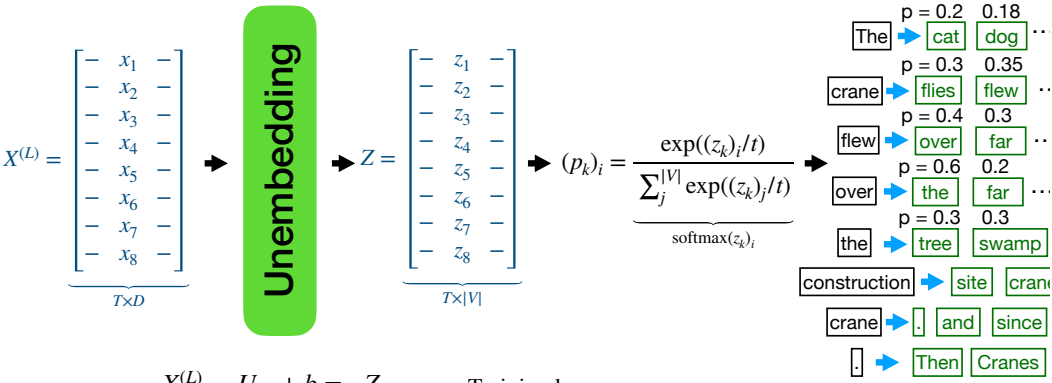



$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Temperature:

- t < 1: logits become larger relative to each other → sharper distribution → more deterministic.
- t > 1: logits shrink → softer distribution → more random.

# Step 3: Unembedding + softmax + training

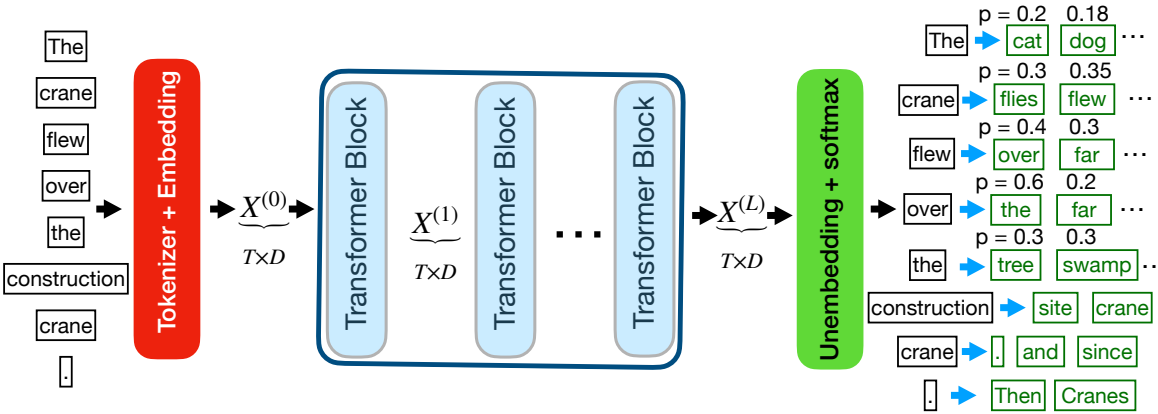



$$\underbrace{X^{(L)}}_{T\times D} \cdot \underbrace{U}_{D\times|V|} + b = \underbrace{Z}_{T\times|V|}$$

Training loss:

$$L(\theta) = -\sum_{t=1}^{7} \log p_\theta(x_{t+1} \mid x_{1:t}) = -\sum_{t=1}^{7} \log(p_k)_{x_{k+1}}$$

# Summary: Decoder Transformer



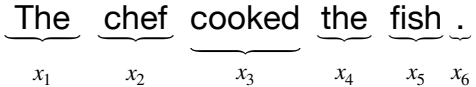

Tokenizer + Embedding : $\Sigma^* \to \mathbb{R}^{T \times D}$ string to sequence embedding

Transformer Block : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times D}$ sequence embeddings

Unembedding + Softmax : $\mathbb{R}^{T \times D} \to \mathbb{R}^{T \times |V|}$ probability over token vocabulary

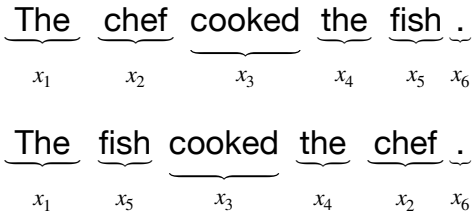
# Position Embeddings

# Position embedding: Absolute (learned) embedding



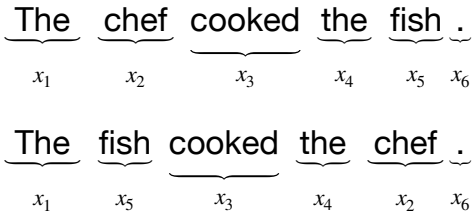

$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

# Position embedding: Absolute (learned) embedding



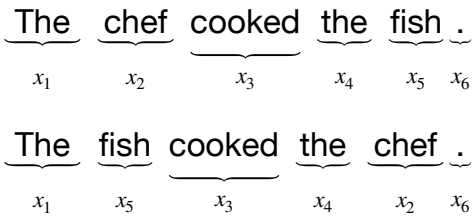

$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

# Position embedding: Absolute (learned) embedding

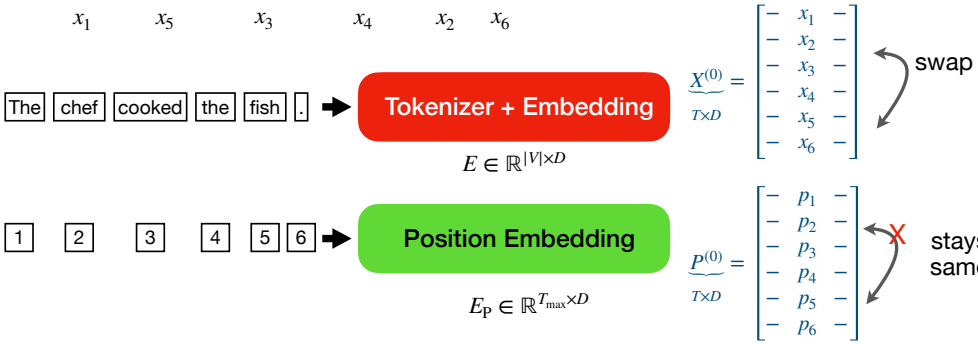



$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$

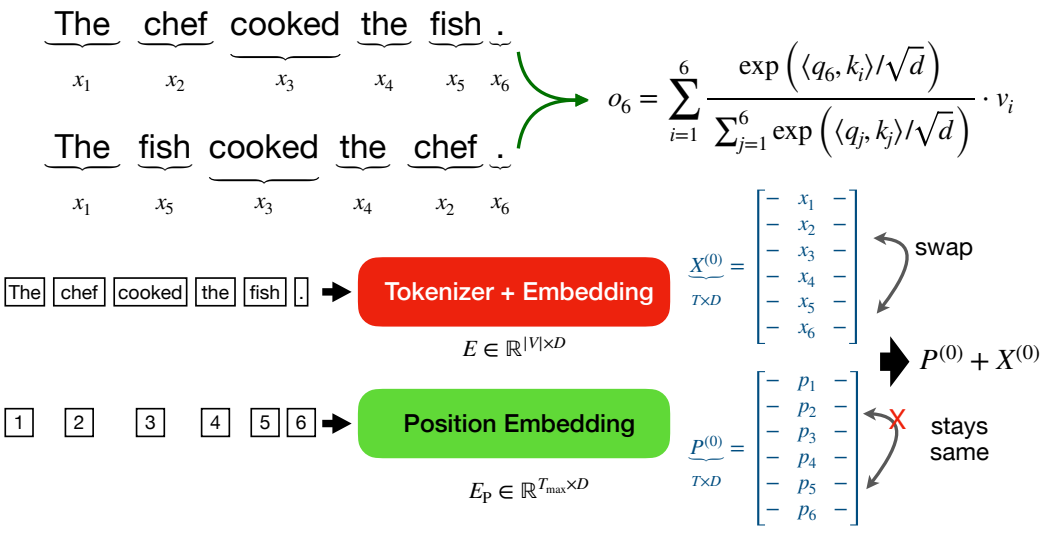
# Position embedding: Absolute (learned) embedding




$$o_6 = \sum_{i=1}^{6} \frac{\exp\left(\langle q_6, k_i \rangle / \sqrt{d}\right)}{\sum_{j=1}^{6} \exp\left(\langle q_j, k_j \rangle / \sqrt{d}\right)} \cdot v_i$$



# Position embedding: Absolute (learned) embedding



# Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding

# Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position

# Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position
- **Relative embeddings:**
  - Add a position vector *inside* of the attention computation

# Variations of position embeddings

- **Absolute (learned) embeddings:**
  - Add a position vector (learned) to the embedding
- **Sinusoidal embeddings:**
  - Add a position vector based on a frequency of position
- **Relative embeddings:**
  - Add a position vector *inside* of the attention computation

**Want:** Inner product invariant to *absolute position* shifts

The constructed pos. embedding $f : \mathbb{R}^D \times \mathbb{N} \to \mathbb{R}^D$ has

$$\langle f(x_1, i), f(x_2, j) \rangle = g(x_1, x_2, i - j) \qquad \text{for some } g : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{N} \to \mathbb{R}.$$

- The inner product depends only on:
  - token embeddings $x_1, x_2$
  - relative distance $i - j$

# RoPE: Rotary Position Embeddings **(Su et al., 2021)**

**Idea:** Rotation preserves inner products

For positions $p \in \{1, \ldots, T_{\max}\}$, if we have an orthogonal matrices $\{R_p\}_{p=1}^{T_{\max}}$, s.t.,

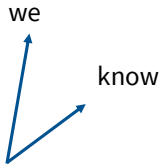
$$R_p^\top = R_{-p} \quad \text{and} \quad R_m^\top R_n = R_{n-m} \quad \Longrightarrow \quad \langle R_m x_1, R_n x_2 \rangle = \langle x_1, R_{n-m} x_2 \rangle.$$

# RoPE: Rotary Position Embeddings **(Su et al., 2021)**

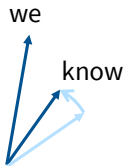
## **Idea:** Rotation preserves inner products

For positions $p \in \{1, \ldots, T_{\max}\}$, if we have an orthogonal matrices $\{R_p\}_{p=1}^{T_{\max}}$, s.t.,

$$R_p^\top = R_{-p} \quad \text{and} \quad R_m^\top R_n = R_{n-m} \quad \Longrightarrow \quad \langle R_m x_1, R_n x_2 \rangle = \langle x_1, R_{n-m} x_2 \rangle.$$

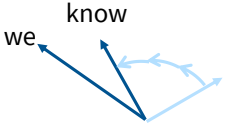



Position independent embedding




Embedding
“we know that”

Rotate we by ‘0 positions’
know by ‘1 positions’




Embedding
“of course we know”

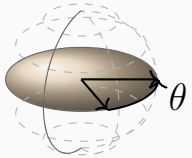
Rotate we by ‘2 positions’
Rotate know by ‘3 positions’

From CS336: Lecture 3 on Architectures

# RoPE: A suitable family of rotations in $\mathbb{R}^D$

## Rotation in 2D subspaces ("Givens rotation")

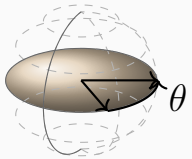
$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$



# RoPE: A suitable family of rotations in $\mathbb{R}^D$

## Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$




## RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high $i$ rotate slower)

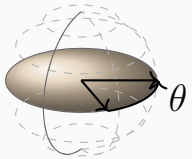
$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i),$$

$$f(x, p) = R_{\theta,D}(p)\, x$$

# RoPE: A suitable family of rotations in $\mathbb{R}^D$

## Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$




## RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high $i$ rotate slower)

$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i),$$

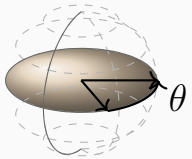
$$f(x, p) = R_{\theta,D}(p)\, x$$

$$R_{\theta,D}(p) = \begin{pmatrix} \cos(p\theta_1) & -\sin(p\theta_1) & & \\ \sin(p\theta_1) & \cos(p\theta_1) & & \\ & & \cos(p\theta_2) & -\sin(p\theta_2) \\ & & \sin(p\theta_2) & \cos(p\theta_2) \\ & & & & \ddots \end{pmatrix}$$

# RoPE: A suitable family of rotations in $\mathbb{R}^D$

## Rotation in 2D subspaces ("Givens rotation")

$$G(i, i+1, \theta) = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & I_{D-i-1} \end{pmatrix}$$




## RoPE

Let $\theta_i = 10\,000^{-2i/D}$ (high $i$ rotate slower)

$$R_{\theta,D}(p) = \prod_{i=1}^{D/2} G(i, i+1, p\theta_i), \qquad f(x, p) = R_{\theta,D}(p)\, x$$

$$R_{\theta,D}(p) = \begin{pmatrix} \cos(p\theta_1) & -\sin(p\theta_1) & & \\ \sin(p\theta_1) & \cos(p\theta_1) & & \\ & & \cos(p\theta_2) & -\sin(p\theta_2) \\ & & \sin(p\theta_2) & \cos(p\theta_2) \\ & & & & \ddots \end{pmatrix}$$

- Can be viewed as sampled Fourier phases of a Dirac delta (point mass) at position $p$.

# RoPE: Where it is used?

## Where it is used?

For hidden states $X \in \mathbb{R}^{T \times D}$, first compute

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V.$$

Then, in each attention head and at each position $t$,

$$\widetilde{q}_t = R_{\theta,d_h}(t)\, q_t, \qquad \widetilde{k}_t = R_{\theta,d_h}(t)\, k_t, \qquad \widetilde{v}_t = v_t.$$

## Effect on attention scores

$$a_{mn} = \frac{\widetilde{q}_m^\top \widetilde{k}_n}{\sqrt{d_h}} = \frac{q_m^\top R_{\theta,d_h}(n-m)\, k_n}{\sqrt{d_h}}.$$

- RoPE is applied **after the** $Q, K, V$ **projections**, separately in **each head**.
- **High-frequency** components vary quickly with position $\Rightarrow$ capture **local** information.
- **Low-frequency** components vary slowly with position $\Rightarrow$ capture **global / long-range**.

# Part 3: Architecture Variations, Inference, Post-Training

# Part 3a) Architecture Variations

# Normalization / Stability

- Representations can grow across many Transformer blocks.

# Normalization / Stability

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.

# Normalization / Stability

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.

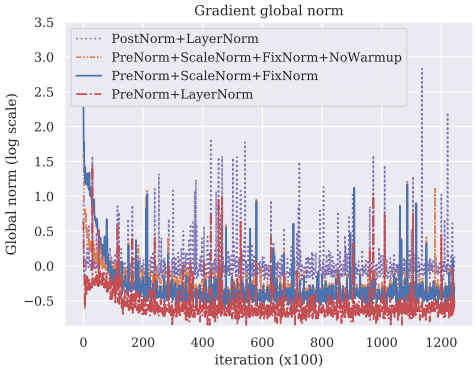
# Normalization / Stability

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.
- Improve:
  - gradient flow,
  - training stability,
  - depth scalability.

# Normalization / Stability

- Representations can grow across many Transformer blocks.
- Large activation scales makes optimization unstable.
- Normalization keeps representations on a controlled scale.
- Improve:
  - gradient flow,
  - training stability,
  - depth scalability.

**Main role in Transformers** Keep each sublayer input well-conditioned before attention / FFN computations.



# Pre-norm vs post-norm

## Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

# Pre-norm vs post-norm

## Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

## Pre-norm (modern LLMs)

$$x_{l+1} = x_l + F(\mathrm{Norm}(x_l))$$

- *Before* MHA (or FFN).
- Cleaner residual path for gradients.
- More stable for deep, large-scale training.

# Pre-norm vs post-norm

## Post-norm (original Transformer)

$$x_{l+1} = \mathrm{Norm}\big(x_l + F(x_l)\big)$$

- *After* the addition and MHA (or FFN).
- Harder to optimize in very deep Transformers.

## Pre-norm (modern LLMs)

$$x_{l+1} = x_l + F(\mathrm{Norm}(x_l))$$

- *Before* MHA (or FFN).
- Cleaner residual path for gradients.
- More stable for deep, large-scale training.

Most modern LLMs use **pre-norm** because it trains more reliably at scale.

# LayerNorm vs RMSNorm

## LayerNorm

Normalizes both mean and variance across features:

$$\mathrm{LayerNorm}(x) = \gamma \odot \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \varepsilon}} + \beta$$

- centers and rescales + learned shift $\beta$,

**Typical examples (LayerNorm):**
Original Transformer, BERT, GPT-2/3, OPT

## RMSNorm

Normalizes only the root-mean-square:

$$\mathrm{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}}$$

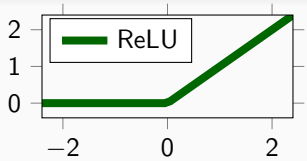
- does *not* subtract the mean,
- simpler and cheaper, widely used in modern LLMs.

**Typical examples (RMSNorm):**
T5, PaLM, LLaMA-family, Gemma

# Common activations: ReLU $\rightarrow$ GeLU

## ReLU (Rectified Linear Unit)

$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$

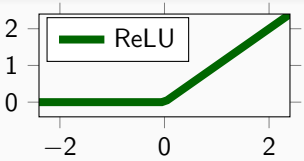



**Models:** Original Transformer, T5, Gopher, Chinchilla, OPT ...

# Common activations: ReLU → GeLU

## ReLU (Rectified Linear Unit)

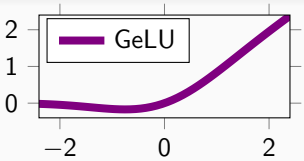
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$




**Models:** Original Transformer, T5, Gopher, Chinchilla, OPT ...

## GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

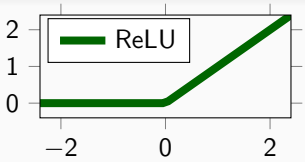



**Models:** GPT1/2/3, GPTJ, BERT, ...

# Common activations: ReLU → GeLU

## ReLU (Rectified Linear Unit)

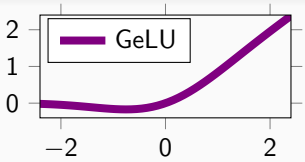
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$




**Models:** Original Transformer, T5, Gopher, Chinchilla, OPT ...

## GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$



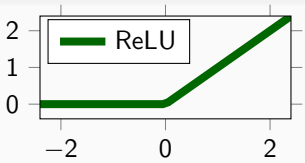

**Models:** GPT1/2/3, GPTJ, BERT, ...

- **Smoother non-linearity:** no non-smooth kink at 0.

# Common activations: ReLU → GeLU

## ReLU (Rectified Linear Unit)

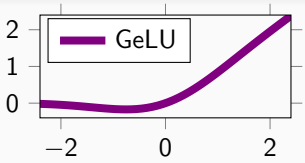
$$\mathrm{FFN}(x) = \max\{0, xW_1\}W_2$$




**Models:** Original Transformer, T5, Gopher, Chinchilla, OPT ...

## GeLU (Gaussian Error Linear Unit)

$$\mathrm{FFN}(x) = \mathrm{GeLU}(xW_1)W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$
$$\Phi(x) \coloneqq \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$




**Models:** GPT1/2/3, GPTJ, BERT, ...

- **Smoother non-linearity:** no non-smooth kink at 0.
- **Soft-gating:** preserves information of small negative activations (instead of setting them 0).

# Gated activations (Gated Linear Units, *GLU)

**Idea:** Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

# Gated activations (Gated Linear Units, *GLU)

**Idea:** Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

## GeGLU (=GeLU + GLU)

$$\mathrm{FFN}_{\mathrm{GeGLU}}(x) = (\mathrm{GeLU}(xW_1) \otimes (xV))\, W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$

**Models:** T5-v1.1, Gemma, ...

# Gated activations (Gated Linear Units, *GLU)

## **Idea:** Separate "gating" from the linear transform

$$\max\{0, xW_1\} \quad \rightarrow \quad \max\{0, \underbrace{xW_1}_{\text{map for gating}}\} \otimes \underbrace{(xV)}_{\text{linear map on } x}$$

$$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\max\{0, xW_1\} \otimes (xV))\, W_2$$

## GeGLU (=GeLU + GLU)

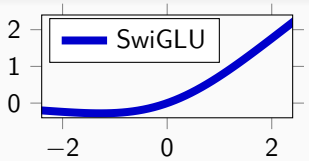
$$\mathrm{FFN}_{\mathrm{GeGLU}}(x) = (\mathrm{GeLU}(xW_1) \otimes (xV))\, W_2$$
$$\mathrm{GeLU}(x) = x\Phi(x)$$

**Models:** T5-v1.1, Gemma, ...

## SwiGLU (Swish Gated Linear Unit)

$\mathrm{Swish}(x) = x\,\mathrm{sigmoid}(x)$, where $\mathrm{sigmoid}(x) = 1/(1+e^{-x})$

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x) = (\mathrm{Swish}(xW_1) \otimes (xV))\, W_2$$




**Models:** LLaMA, PaLM, Mistral, ...

# Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.

# Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.

# Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit $+ \mathrm{Swish}(x)$.

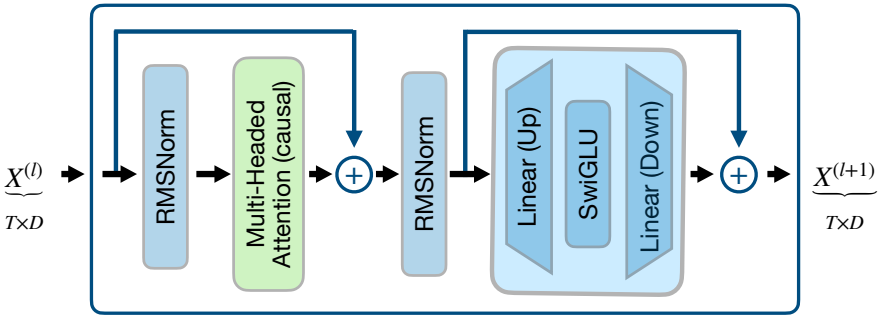
# Takeaways: Modern Transformer Decoder Block

- **Normalization placement:** Post-Norm $\Rightarrow$ **Pre-Norm**
  - Normalizes inputs *before* sublayers $\Rightarrow$ better training stability for deep networks.
- **Normalization type:** Layer Norm $\Rightarrow$ **RMSNorm**
  - Removes the mean-centering $\Rightarrow$ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU $\Rightarrow$ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit $+ \mathrm{Swish}(x)$.
- **Parameters:** Biases $\Rightarrow$ **Remove Biases**
  - Bias terms $(+b)$ are dropped to save memory and increase throughput.

# Takeaways: Modern Transformer Decoder Block

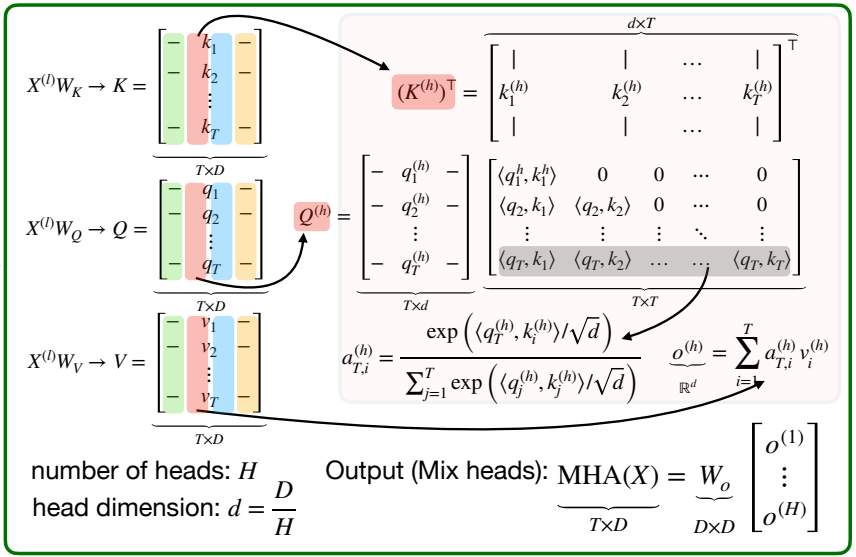
- **Normalization placement:** Post-Norm ⇒ **Pre-Norm**
  - Normalizes inputs *before* sublayers ⇒ better training stability for deep networks.
- **Normalization type:** Layer Norm ⇒ **RMSNorm**
  - Removes the mean-centering ⇒ computationally cheaper while maintaining model quality.
- **Activations in Feed-Forward (FFN):** ReLU ⇒ **SwiGLU**
  - Replaces 2-layer MLP with a 3-layer *gated* linear unit + $\mathrm{Swish}(x)$.
- **Parameters:** Biases ⇒ **Remove Biases**
  - Bias terms ($+b$) are dropped to save memory and increase throughput.
- **Positional Encoding:** Absolute (Input) ⇒ **RoPE** in every head of MHA
  - Injected directly *inside* MHA to model relative distances ⇒ improving long-context modelling.

# Decoder transformer



# Part 3b) Inference: Prefill vs Decode

# Recall: Multi-Headed Attention (MHA) with causal mask



# Metrics for Inference

## Metrics

- *Time-to-first-token (TTFT):* How long user waits before ay generation appears
- *Latency (seconds/token):* How fast tokens appear for user
- *Throughput (tokens/second):* How many tokens the model outputs overall per second (among all users).

# Metrics for Inference

## Metrics

- *Time-to-first-token (TTFT):* How long user waits before ay generation appears
- *Latency (seconds/token):* How fast tokens appear for user
- *Throughput (tokens/second):* How many tokens the model outputs overall per second (among all users).

## What is different now?

- Training (supervised): you feed whole sentences (all tokens) $\Rightarrow$ can parallelize over sequence.
- Inference: you have to generate sequentially $\Rightarrow$ cannot paralelize.

# Metrics for Inference

## Metrics

- *Time-to-first-token (TTFT):* How long user waits before ay generation appears
- *Latency (seconds/token):* How fast tokens appear for user
- *Throughput (tokens/second):* How many tokens the model outputs overall per second (among all users).

## What is different now?

- Training (supervised): you feed whole sentences (all tokens) $\Rightarrow$ can parallelize over sequence.
- Inference: you have to generate sequentially $\Rightarrow$ cannot paralelize.

***Why important?***

- Providers of closed / open models must solve this problem
- Influences architecture

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}, W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$
3. *Write $XW$:* bytes transfered: $2 \cdot B \cdot F$

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$
3. *Write $XW$:* bytes transfered: $2 \cdot B \cdot F$

$$\text{Intensity} = \frac{\text{flops}}{\text{bytes transfered}} = \frac{2BDF}{2BD + 2DF + 2BF} \approx B.$$

How many operations per read/write

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$
3. *Write $XW$:* bytes transfered: $2 \cdot B \cdot F$

$$\text{Intensity} = \frac{\text{flops}}{\text{bytes transfered}} = \frac{2BDF}{2BD + 2DF + 2BF} \approx B.$$

How many operations per read/write

## Theoretical arithmetic-intensity limit of the hardware

- **L40S:** $\approx$ 200 FLOP/byte (TF32), $\approx$ 400 FLOP/byte (BF16/FP16)
- **H100 SXM:** $\approx$ 300 FLOP/byte (TF32), $\approx$ 600 FLOP/byte (BF16/FP16)

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$
3. *Write $XW$:* bytes transfered: $2 \cdot B \cdot F$

$$\text{Intensity} = \frac{\text{flops}}{\text{bytes transfered}} = \frac{2BDF}{2BD + 2DF + 2BF} \approx B.$$

How many operations per read/write

## Theoretical arithmetic-intensity limit of the hardware

- **L40S:** $\approx$ 200 FLOP/byte (TF32), $\approx$ 400 FLOP/byte (BF16/FP16)
- **H100 SXM:** $\approx$ 300 FLOP/byte (TF32), $\approx$ 600 FLOP/byte (BF16/FP16)

- If computation intensity $>$ accelerator intensity, compute-limited (good)
- If computation intensity $<$ accelerator intensity, memory-limited (bad)

# Why we broadcast over batches?

**Arithmetic Intensity:** for $X \cdot W$, where $X \in \mathbb{R}^{B \times D}$, $W \in \mathbb{R}^{D \times F}$

1. *Read $X$ and $W$:* bytes transfered: $2 \cdot B \cdot D + 2 \cdot D \cdot F$
2. *Compute $XW$:* flops: $2 \cdot B \cdot D \cdot F$
3. *Write $XW$:* bytes transfered: $2 \cdot B \cdot F$

$$\text{Intensity} = \frac{\text{flops}}{\text{bytes transfered}} = \frac{2BDF}{2BD + 2DF + 2BF} \approx B.$$

How many operations per read/write

## Theoretical arithmetic-intensity limit of the hardware

- **L40S:** $\approx$ 200 FLOP/byte (TF32), $\approx$ 400 FLOP/byte (BF16/FP16)
- **H100 SXM:** $\approx$ 300 FLOP/byte (TF32), $\approx$ 600 FLOP/byte (BF16/FP16)

- If computation intensity $>$ accelerator intensity, compute-limited (good)
- If computation intensity $<$ accelerator intensity, memory-limited (bad)

**Conclusion:** compute-limited in BF16 if $B > 400$.

# MLP Arithmetic Summary

## Prefill ($T = S$)

$$\text{FLOPs} \approx 6\,B\,S\,D\,F$$

$$\text{Memory} \approx 4\,B\,S\,D + 4\,B\,S\,F + 6DF$$

$$\text{Intensity} = \mathcal{O}(B\,S)$$

- Easy to make compute-limited
- Good regime for hardware utilization

# MLP Arithmetic Summary

## Prefill ($T = S$)

$$\text{FLOPs} \approx 6\,B\,S\,D\,F$$

$$\text{Memory} \approx 4\,B\,S\,D + 4\,B\,S\,F + 6DF$$

$$\text{Intensity} = \mathcal{O}(B\,S)$$

- Easy to make compute-limited
- Good regime for hardware utilization

## Generation ($T = 1$)

$$\text{FLOPs} \approx 6BDF$$

$$\text{Memory} \approx 4BD + 4BF + 6DF$$

$$\text{Intensity} = \mathcal{O}(B).$$

- One token: $T = 1$, for $B$ requests
- Hard to make $B$ large enough

# MLP Arithmetic Summary

## Prefill ($T = S$)

$$\text{FLOPs} \approx 6\,B\,S\,D\,F$$

$$\text{Memory} \approx 4\,B\,S\,D + 4\,B\,S\,F + 6DF$$

$$\text{Intensity} = \mathcal{O}(B\,S)$$

- Easy to make compute-limited
- Good regime for hardware utilization

## Generation ($T = 1$)

$$\text{FLOPs} \approx 6BDF$$

$$\text{Memory} \approx 4BD + 4BF + 6DF$$

$$\text{Intensity} = \mathcal{O}(B).$$

- One token: $T = 1$, for $B$ requests
- Hard to make $B$ large enough

## Takeaway

For $BT \ll D, F$,

$$\text{Intensity} = \mathcal{O}(BT).$$

So MLP layers benefit strongly from batching:

Prefill: increase $BT$, Generation: increase $B$.

# MHA Arithmetic (FlashAttn) Summary

$$\text{Intensity} = \frac{ST}{S+T}. \qquad \Rightarrow \qquad \text{Prefill: } \text{Intensity} = \frac{S}{2}, \qquad \text{Generation: } \text{Intensity} < 1$$

# MHA Arithmetic (FlashAttn) Summary

$$\text{Intensity} = \frac{ST}{S+T}. \quad \Rightarrow \quad \text{Prefill: } \text{Intensity} = \frac{S}{2}, \quad \text{Generation: } \text{Intensity} < 1$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 8BSD$$

$$\text{Intensity} = \frac{S}{2}$$

- Possible to make compute-limited
- Longer context directly helps

# MHA Arithmetic (FlashAttn) Summary

$$\text{Intensity} = \frac{ST}{S+T}. \quad \Rightarrow \quad \text{Prefill: } \text{Intensity} = \frac{S}{2}, \quad \text{Generation: } \text{Intensity} < 1$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 8BSD$$

$$\text{Intensity} = \frac{S}{2}$$

- Possible to make compute-limited
- Longer context directly helps

## Generation ($T = 1$)

$$\text{FLOPs} \approx 4BSD$$

$$\text{Memory} \approx 4BSD + 4BD$$

$$\text{Intensity} = \frac{S}{S+1} < 1$$

- One token attends to $S$ cached tokens
- Strongly memory-limited

# MHA Arithmetic (FlashAttn) Summary

$$\text{Intensity} = \frac{ST}{S+T}. \quad \Rightarrow \quad \text{Prefill: } \text{Intensity} = \frac{S}{2}, \quad \text{Generation: } \text{Intensity} < 1$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 8BSD$$

$$\text{Intensity} = \frac{S}{2}$$

- Possible to make compute-limited
- Longer context directly helps

## Generation ($T = 1$)

$$\text{FLOPs} \approx 4BSD$$

$$\text{Memory} \approx 4BSD + 4BD$$

$$\text{Intensity} = \frac{S}{S+1} < 1$$

- One token attends to $S$ cached tokens
- Strongly memory-limited

## Takeaway

Batching $B$ improves throughput, *but not* the intensity (each request reads its own KV cache.)

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Attention can have high arithmetic intensity (for long sequences)
- Batching increases throughput, **but also**, time-to-first-token (TTFT).

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Attention can have high arithmetic intensity (for long sequences)
- Batching increases throughput, **but also**, time-to-first-token (TTFT).

## Generation

Decode one new token at a time

$$T = 1, \qquad S = \text{cache length}$$

- Attention becomes memory-limited
- Must repeatedly read the KV cache
- Can increase **throughput** by batching
- But, **latency** is limited by intensity.

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Attention can have high arithmetic intensity (for long sequences)
- Batching increases throughput, **but also**, time-to-first-token (TTFT).

## Generation

Decode one new token at a time

$$T = 1, \qquad S = \text{cache length}$$

- Attention becomes memory-limited
- Must repeatedly read the KV cache
- Can increase **throughput** by batching
- But, **latency** is limited by intensity.

## Main takeaway

Prefill : TTFT vs throughput tradeoff  Generation: Intensity limits latency, throughput with $B$.

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Compute over the whole prompt in parallel
- High arithmetic intensity for long sequences
- Batching improves throughput, but large prefill jobs can increase **TTFT**

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Compute over the whole prompt in parallel
- High arithmetic intensity for long sequences
- Batching improves throughput, but large prefill jobs can increase **TTFT**

## Generation / decode

Decode one new token at a time

$$T = 1, \qquad S = \text{cache length}$$

- Attention has low arithmetic intensity
- Each step repeatedly reads the KV cache
- Batching improves aggregate **tokens/s**
- Per-user latency is limited by sequential decoding and memory bandwidth

# Transformer Block: Prefill vs Generation

## Prefill

Input: many prompt tokens at once

$$T = S$$

- Compute over the whole prompt in parallel
- High arithmetic intensity for long sequences
- Batching improves throughput, but large prefill jobs can increase **TTFT**

## Generation / decode

Decode one new token at a time

$$T = 1, \qquad S = \text{cache length}$$

- Attention has low arithmetic intensity
- Each step repeatedly reads the KV cache
- Batching improves aggregate **tokens/s**
- Per-user latency is limited by sequential decoding and memory bandwidth

## Main takeaway

Prefill: compute-rich prompt processing  Decode: memory-bound token loop

Batching improves throughput, but serving must balance TTFT and per-token latency.

# Strategy

## Prefill

- Goal: reduce **TTFT**
- Avoid too large batches
- Prioritize latency over throughput

# Strategy

## Prefill

- Goal: reduce **TTFT**
- Avoid too large batches
- Prioritize latency over throughput

## Generation

- Goal: improve throughput **tokens / second**
- Merge requests (larger batches) if possible
- Prioritize throughput (latency is limited)

# Strategy

## Prefill

- Goal: reduce **TTFT**
- Avoid too large batches
- Prioritize latency over throughput

## Generation

- Goal: improve throughput **tokens / second**
- Merge requests (larger batches) if possible
- Prioritize throughput (latency is limited)

## Two regimes

Prefill: small batch (long sequence) $\Rightarrow$ fast first token

Generation: larger batch $\Rightarrow$ better overall throughput

# Strategy

## Prefill

- Goal: reduce **TTFT**
- Avoid too large batches
- Prioritize latency over throughput

## Generation

- Goal: improve throughput **tokens / second**
- Merge requests (larger batches) if possible
- Prioritize throughput (latency is limited)

## Two regimes

Prefill: small batch (long sequence) $\Rightarrow$ fast first token

Generation: larger batch $\Rightarrow$ better overall throughput

## ... but what to do about latency in generation?

# Grouped-query attention **(Ainslie et al., 2023)**

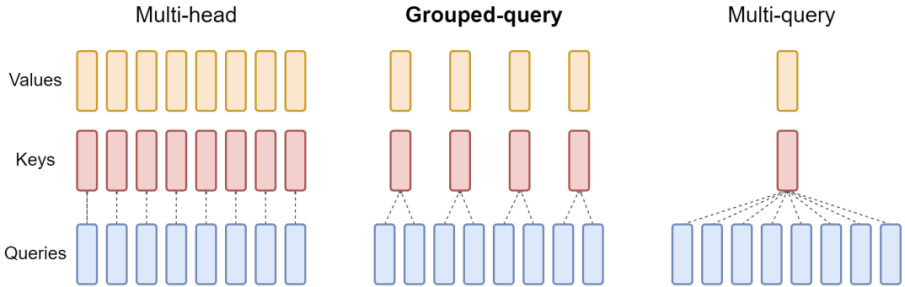



Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

# Grouped-query attention **(Ainslie et al., 2023)**

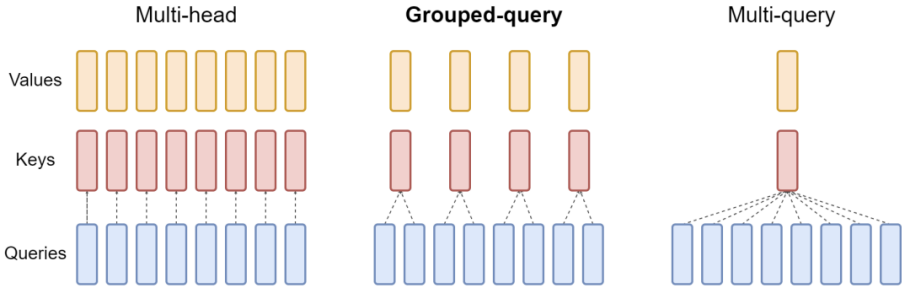



Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

**Idea:** Fewer key/value heads

- $H_Q$ query heads, but only $H_K$ key/value heads
- each interacting with $G = H_Q/H_K$ query heads

# Grouped-query attention **(Ainslie et al., 2023)**

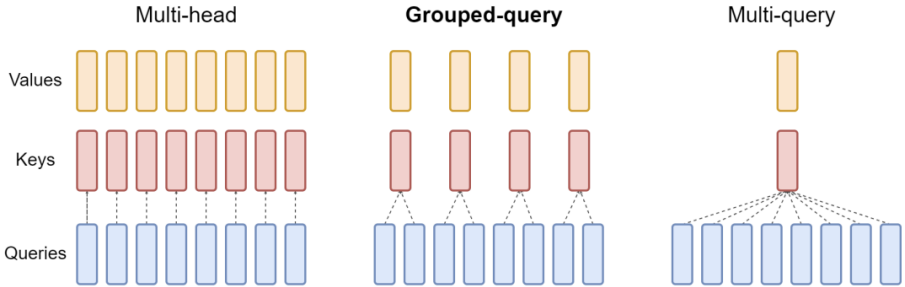



Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

**Idea:** Fewer key/value heads

- $H_Q$ query heads, but only $H_K$ key/value heads
- each interacting with $G = H_Q/H_K$ query heads

**Why?**

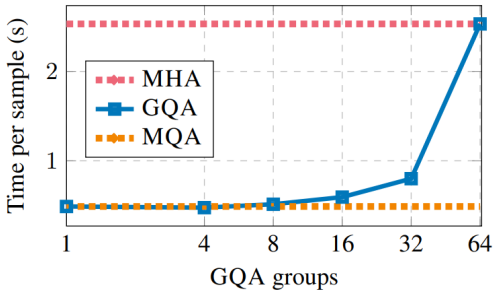
Smaller KV cache and less memory traffic during generation.

# Why GQA Helps Inference

- **MHA:** $H_Q = H_K$
  - largest KV cache
  - best flexibility
- **MQA:** $H_K = 1$
  - smallest KV cache
  - strongest decoding speedup
- **GQA:** $1 < H_K < H_Q$
  - compromise between quality and efficiency

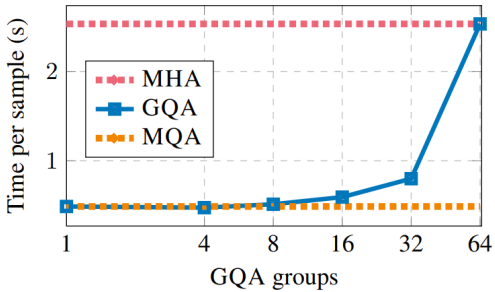
# Why GQA Helps Inference

- **MHA:** $H_Q = H_K$
  - largest KV cache
  - best flexibility
- **MQA:** $H_K = 1$
  - smallest KV cache
  - strongest decoding speedup
- **GQA:** $1 < H_K < H_Q$
  - compromise between quality and efficiency



# Why GQA Helps Inference

- **MHA:** $H_Q = H_K$
  - largest KV cache
  - best flexibility
- **MQA:** $H_K = 1$
  - smallest KV cache
  - strongest decoding speedup
- **GQA:** $1 < H_K < H_Q$
  - compromise between quality and efficiency




## Takeaway

GQA keeps many query heads, but few keys/values heads ⇒ reduce the memory cost + operations.

# GQA Arithmetic Summary

$$\text{Intensity}: \quad \frac{ST}{T+S} \Rightarrow \frac{ST}{T+S/G}$$

$$\text{Memory}: \quad 4BSD \Rightarrow 4BS\frac{D}{G}.$$

# GQA Arithmetic Summary

$$\text{Intensity}: \quad \frac{ST}{T+S} \Rightarrow \frac{ST}{T+S/G}$$

$$\text{Memory}: \quad 4BSD \Rightarrow 4BS\frac{D}{G}.$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 4BSD + 4BS(D/G)$$

$$\text{Intensity} = \frac{S}{1+1/G} = \frac{GS}{G+1}$$

- Slightly better than MHA

# GQA Arithmetic Summary

$$\text{Intensity}: \quad \frac{ST}{T+S} \Rightarrow \frac{ST}{T+S/G}$$

$$\text{Memory}: \quad 4BSD \Rightarrow 4BS\frac{D}{G}.$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 4BSD + 4BS(D/G)$$

$$\text{Intensity} = \frac{S}{1+1/G} = \frac{GS}{G+1}$$

- Slightly better than MHA

## Generation ($T = 1$)

$$\text{FLOPs} \approx 4BSD$$

$$\text{Memory} \approx 4BD + 4BS(D/G)$$

$$\text{Intensity} = \frac{S}{1+S/G} \approx G \quad (S \gg G)$$

- Higher intensity than MHA
- Smaller KV-cache reads by factor $G$

# GQA Arithmetic Summary

$$\text{Intensity}: \quad \frac{ST}{T+S} \Rightarrow \frac{ST}{T+S/G}$$

$$\text{Memory}: \quad 4BSD \Rightarrow 4BS\frac{D}{G}.$$

## Prefill ($T = S$)

$$\text{FLOPs} \approx 4BS^2D$$

$$\text{Memory} \approx 4BSD + 4BS(D/G)$$

$$\text{Intensity} = \frac{S}{1+1/G} = \frac{GS}{G+1}$$

- Slightly better than MHA

## Generation ($T = 1$)

$$\text{FLOPs} \approx 4BSD$$

$$\text{Memory} \approx 4BD + 4BS(D/G)$$

$$\text{Intensity} = \frac{S}{1+S/G} \approx G \quad (S \gg G)$$

- Higher intensity than MHA
- Smaller KV-cache reads by factor $G$

## Takeaway

Prefill: still good, Generation: latency and memory lower than MHA .. and same FLOPS!

# Takaways: Inference

- **Prefill** is parallel and often efficient; **decode** is sequential and much harder.

# Takaways: Inference

- **Prefill** is parallel and often efficient; **decode** is sequential and much harder.
- The **KV cache** makes autoregressive generation possible, but creates a memory-bandwidth bottleneck.

# Takaways: Inference

- **Prefill** is parallel and often efficient; **decode** is sequential and much harder.
- The **KV cache** makes autoregressive generation possible, but creates a memory-bandwidth bottleneck.
- During decode, **attention is usually memory-limited**, so batching mainly improves **throughput**, not **latency**.

# Takaways: Inference

- **Prefill** is parallel and often efficient; **decode** is sequential and much harder.
- The **KV cache** makes autoregressive generation possible, but creates a memory-bandwidth bottleneck.
- During decode, **attention is usually memory-limited**, so batching mainly improves **throughput**, not **latency**.
- Modern inference methods mostly try to:
  - reduce KV-cache size/bandwidth (GQA, MLA, CLA, local attention),
  - or recover parallelism (speculative decoding).

# Takaways: Inference

- **Prefill** is parallel and often efficient; **decode** is sequential and much harder.
- The **KV cache** makes autoregressive generation possible, but creates a memory-bandwidth bottleneck.
- During decode, **attention is usually memory-limited**, so batching mainly improves **throughput**, not **latency**.
- Modern inference methods mostly try to:
  - reduce KV-cache size/bandwidth (GQA, MLA, CLA, local attention),
  - or recover parallelism (speculative decoding).
- **Final takeaway:** LLM inference is not only a "systems" problem, but also an architecture modeling problem!

# Distribution of language ⇒ task-oriented model

## Pretraining

- Distribution of language
- Syntax, style, world knowledge
- but only to *predict the next token*

# Distribution of language ⇒ task-oriented model

## Pretraining

- Distribution of language
- Syntax, style, world knowledge
- but only to *predict the next token*

## Post-training

- Teaches behaviour
- **SFT:** teach the format
  - answer questions
  - follow instructions
  - imitate demonstrations
- **RL:**
  - human preferences
  - reward models / verifiers
  - RLHF, DPO, RLVR

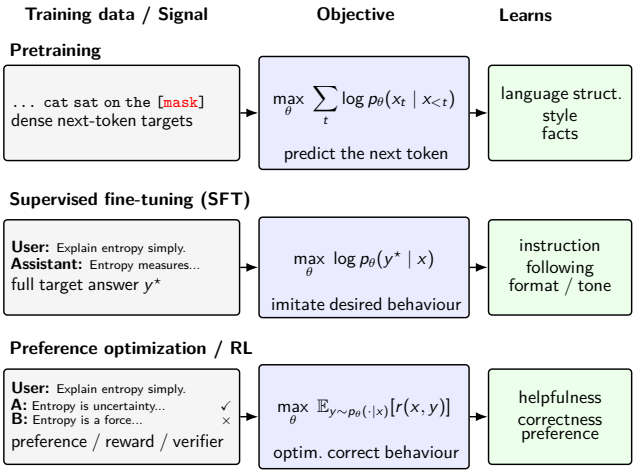
# Distribution of language ⇒ task-oriented model

## Pretraining

- Distribution of language
- Syntax, style, world knowledge
- but only to *predict the next token*

## Post-training

- Teaches behaviour
- **SFT:** teach the format
  - answer questions
  - follow instructions
  - imitate demonstrations
- **RL:**
  - human preferences
  - reward models / verifiers
  - RLHF, DPO, RLVR



# Part 3c) Post-training: SFT, IT, RL

# Instruction tuning: objective and optimization

## Data and model

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

with instructions $x^{(i)}$ and desired responses $y^{(i)}$.
The autoregressive model defines

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}).$$

# Instruction tuning: objective and optimization

## Data and model

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

with instructions $x^{(i)}$ and desired responses $y^{(i)}$.
The autoregressive model defines

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}).$$

## Objective

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \pi_\theta(y_i \mid x_i).$$

- Negative log-likelihood on the **assistant response**.
- **Next-token prediction** conditioned on instruction.

# Instruction tuning: objective and optimization

## Data and model

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

with instructions $x^{(i)}$ and desired responses $y^{(i)}$.
The autoregressive model defines

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}).$$

## Objective

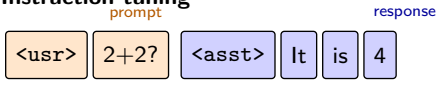
$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \pi_\theta(y_i \mid x_i).$$

- Negative log-likelihood on the **assistant response**.
- **Next-token prediction** conditioned on instruction.

**Pretraining**


loss on all next-token predictions

**Instruction tuning**




prompt tokens: context only
assistant tokens: optimized

Feed the concatenated sequence $[x_i; y_i]$ into the causal LM, but mask out prompt tokens in the loss.

# PEFT: adapt a large model with a small trainable update

## Low-Rank Adaptation (LoRA)

Freeze the pretrained matrix $W_0$ and learn only a low-rank update:

$$W = W_0 + \Delta W, \qquad \Delta W = BA, \qquad r \ll d.$$

So instead of training all of $W$, we train only the small factors $A$ and $B$.

$$h = W_0 x + B(Ax).$$

# PEFT: adapt a large model with a small trainable update

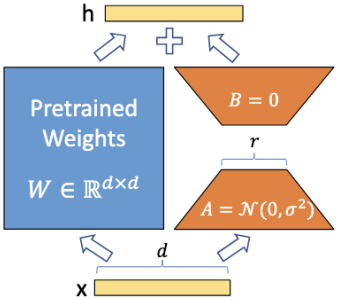
## Low-Rank Adaptation (LoRA)

Freeze the pretrained matrix $W_0$ and learn only a low-rank update:

$$W = W_0 + \Delta W, \qquad \Delta W = BA, \qquad r \ll d.$$

So instead of training all of $W$, we train only the small factors $A$ and $B$.

$$h = W_0 x + B(Ax).$$




## Why this is useful

- few trainable parameters (lower memory / optimizer cost
- one frozen base model + small adapters
- reduces forgetting since $W_0$ stays fixed

# PEFT: adapt a large model with a small trainable update

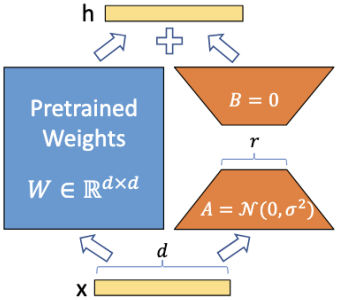
## Low-Rank Adaptation (LoRA)

Freeze the pretrained matrix $W_0$ and learn only a low-rank update:

$$W = W_0 + \Delta W, \qquad \Delta W = BA, \qquad r \ll d.$$

So instead of training all of $W$, we train only the small factors $A$ and $B$.

$$h = W_0 x + B(Ax).$$




### Why this is useful

- few trainable parameters (lower memory / optimizer cost
- one frozen base model + small adapters
- reduces forgetting since $W_0$ stays fixed

## Other PEFT

- **Adapters:** bottleneck modules
- **QLORA:** Quantized LoRA

# FLAN: Instruction Tuning ⇒ Zero-shot Learning **(Wei et al., 2021)**

## What FLAN showed

- IT improves **zero-shot** generalization
- gains transfer to **unseen tasks**
- FLAN 137B beats **zero-shot GPT-3 175B** on many benchmarks

# FLAN: Instruction Tuning ⇒ Zero-shot Learning **(Wei et al., 2021)**

## What FLAN showed

- IT improves **zero-shot** generalization
- gains transfer to **unseen tasks**
- FLAN 137B beats **zero-shot GPT-3 175B** on many benchmarks

### Takeaways

- SFT can create **instruct behavior**
- diverse supervised data gives **cross-task generalization**

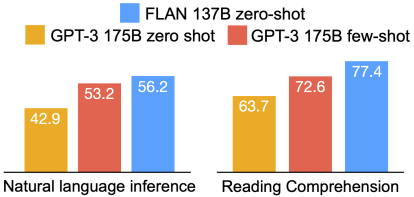
# FLAN: Instruction Tuning ⇒ Zero-shot Learning **(Wei et al., 2021)**

## What FLAN showed

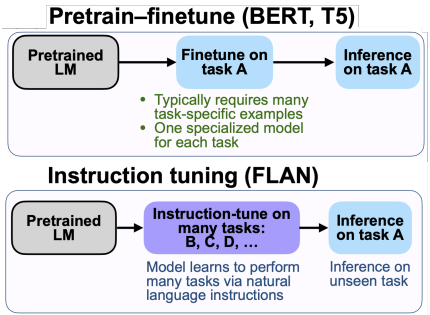
- IT improves **zero-shot** generalization
- gains transfer to **unseen tasks**
- FLAN 137B beats **zero-shot GPT-3 175B** on many benchmarks

## Takeaways

- SFT can create **instruct behavior**
- diverse supervised data gives **cross-task generalization**




Performance on unseen tasks



# Knowledge extraction, alignment, and hallucinations

- When the model doesn't have the knowledge, SFT will just *imitate the style*

# Knowledge extraction, alignment, and hallucinations

- When the model doesn't have the knowledge,
  SFT will just *imitate the style*
- Cross-entropy loss:
  a confident hallucination $\gg$ no answer at all

# Knowledge extraction, alignment, and hallucinations

- When the model doesn't have the knowledge, SFT will just *imitate the style*
- Cross-entropy loss:
  a confident hallucination $\gg$ no answer at all

## Counterintuitive phenomenon

- *Rich and correct* instruction data will still encourage hallucinations
- A weaker student may learn the *format* of expert answers, not the underlying ability
- In distillation, a much stronger teacher can make this worse

# Knowledge extraction, alignment, and hallucinations

- When the model doesn't have the knowledge, SFT will just *imitate the style*
- Cross-entropy loss:
  a confident hallucination $\gg$ no answer at all

## Counterintuitive phenomenon

- *Rich and correct* instruction data will still encourage hallucinations
- A weaker student may learn the *format* of expert answers, not the underlying ability
- In distillation, a much stronger teacher can make this worse

## Takeaway

- SFT is good at **extracting present knowledge**
- It is less reliable for **adding new capabilities**

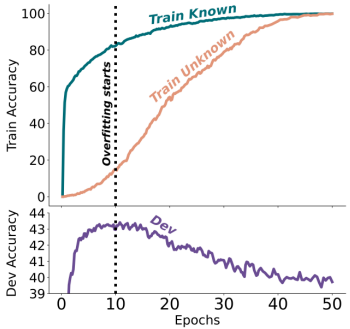
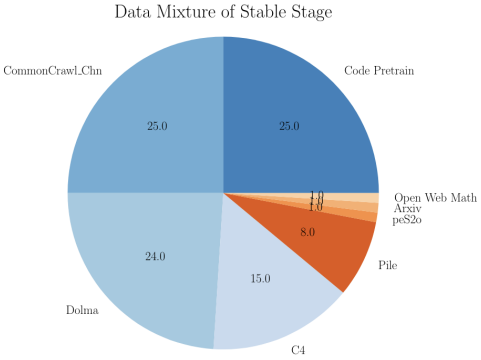
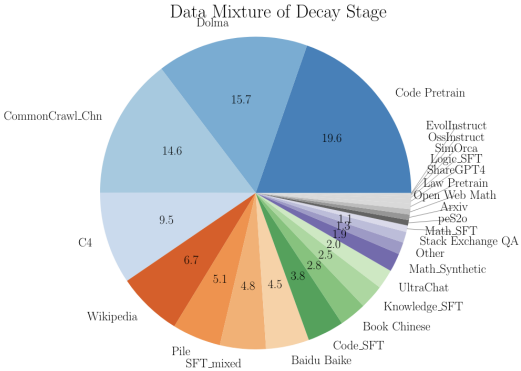



Figure 1: Train and development accuracies as a function of the fine-tuning duration, when fine-tuning on 50% `Known` and 50% `Unknown` examples. `Unknown` examples are fitted substantially slower than `Known`. The best development performance is obtained when the LLM fits the majority of the `Known` training examples but only few of the `Unknown` ones. From this point, fitting `Unknown` examples reduces the performance.

# Midtraining / Two-phase training

- train on a **mixture** of web-scale text and instruction data
- **injects instruct behaviour** without leaving the pretraining regime entirely
- preserves **general knowledge and language modelling ability**
- reduces **catastrophic forgetting:** the model loosing previously learned capabilities.



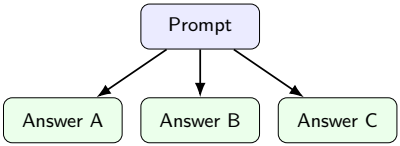



# When instruction tuning is not sufficient

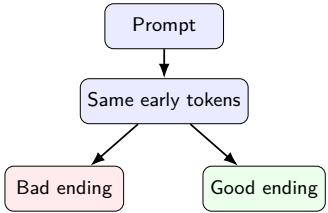
## Instruction tuning is limited

- **There is no single "correct" path**
- **Quality is judged by outcome**
- **The important signal appears only after generation**
- **Different mistakes have different severity**
- **The model may exploit superficial patterns**
- **Preferences are often relative, not absolute**

*In short: instruction tuning learns from fixed targets, but many post-training signals are comparative or outcome-based.*

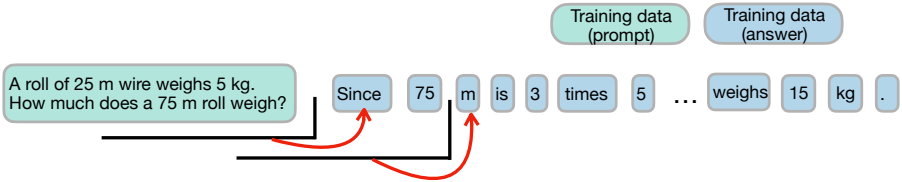



**Many plausible responses:** instruction tuning reinforces one recorded target, even when several answers are acceptable.



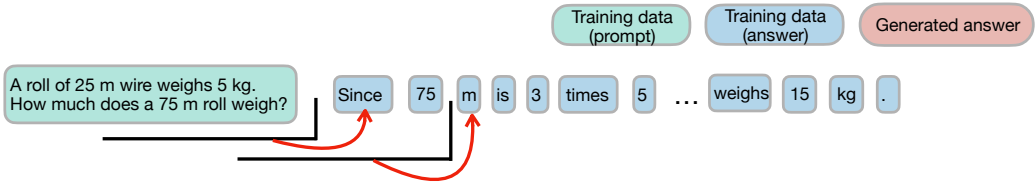

**Outcome visible late:** two responses may look similar at first, but only the full completion reveals whether it is correct, safe, or useful.

# SFT $\Rightarrow$ RL

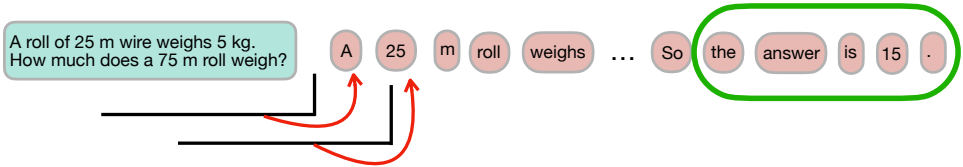



Compute loss of $p_\theta(\text{"m"} \mid \text{"Since"}, \text{"75"}, \text{prompt}) \Rightarrow$ gradient update on $\theta$ to increase the prob. of observed training distribution

# SFT ⇒ RL




Compute loss of $p_\theta(\text{"m"} \mid \text{"Since"}, \text{"75"}, \text{prompt}) \Rightarrow$ gradient update on $\theta$ to increase the prob. of observed training distribution




Generate from "m" $\sim \pi_\theta(\cdot \mid \text{"A"}, \text{"25"}, \text{prompt}) \Rightarrow$ check correctness $\Rightarrow$ push probability of path up

# SFT vs RL

## Supervised Fine-Tuning (SFT)

$$\max_{\theta} \mathbb{E}_{(x,y^\star)\sim\mathcal{D}}\big[\log \pi_\theta(y^\star \mid x)\big]$$

- **Target:** *"do this"*
- Full desired output $y^\star$ is given
- Strong, structured training signal

$$x \longrightarrow y^\star$$

Increase probability of the target response.

# SFT vs RL

## Supervised Fine-Tuning (SFT)

$$\max_{\theta} \mathbb{E}_{(x,y^\star)\sim\mathcal{D}}\big[\log \pi_\theta(y^\star \mid x)\big]$$

- **Target:** *"do this"*
- Full desired output $y^\star$ is given
- Strong, structured training signal

$$x \longrightarrow y^\star$$

Increase probability of the target response.

## Reinforcement Learning (RL)

$$\max_{\theta} \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\big[r(x,y)\big]$$

- **Reward:** *"what you did worked well"*
- Model must first sample an output $y$
- Feedback is weaker: a reward score

$$x \longrightarrow y \longrightarrow r(x,y)$$

Increase probability of higher-reward responses.

# SFT vs RL

## Supervised Fine-Tuning (SFT)

$$\max_{\theta} \mathbb{E}_{(x,y^\star)\sim\mathcal{D}}\big[\log \pi_\theta(y^\star \mid x)\big]$$

- **Target:** *"do this"*
- Full desired output $y^\star$ is given
- Strong, structured training signal

$$x \longrightarrow y^\star$$

Increase probability of the target response.

## Reinforcement Learning (RL)

$$\max_{\theta} \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\big[r(x,y)\big]$$

- **Reward:** *"what you did worked well"*
- Model must first sample an output $y$
- Feedback is weaker: a reward score

$$x \longrightarrow y \longrightarrow r(x,y)$$

Increase probability of higher-reward responses.

## Challenges of using rewards

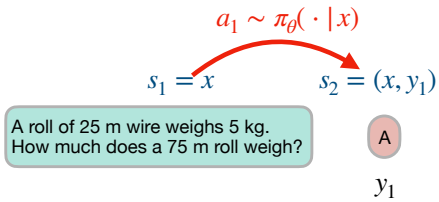
- **Scalar vs structured:** a label specifies the desired output; a reward is one number.
- **Credit assignment:** the reward arrives after the whole response, so it is unclear which part of reasoning are responsible for it.

# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction

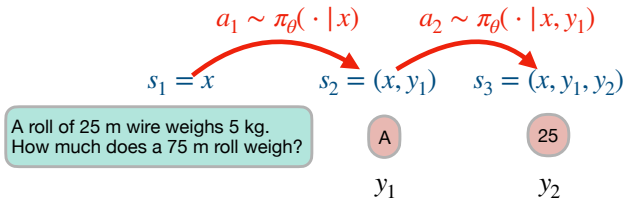


# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction

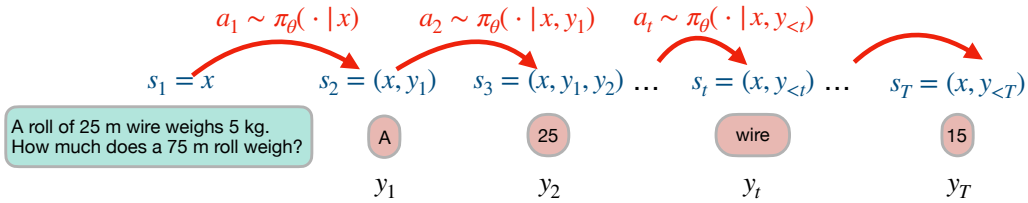


# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction



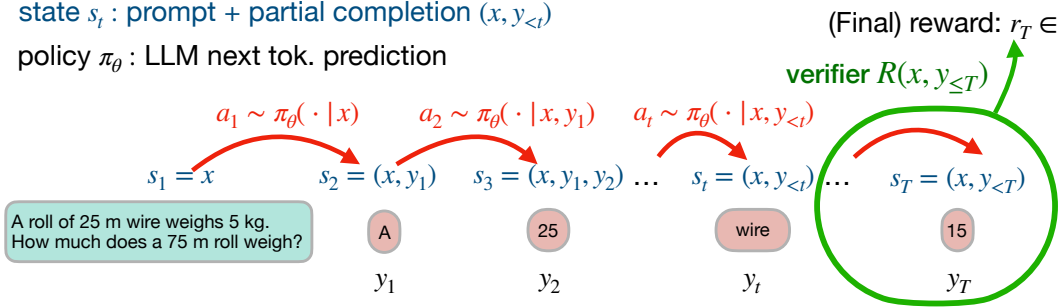

rollout $y_{\leq T}$ : fully AR sampled answer

# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction



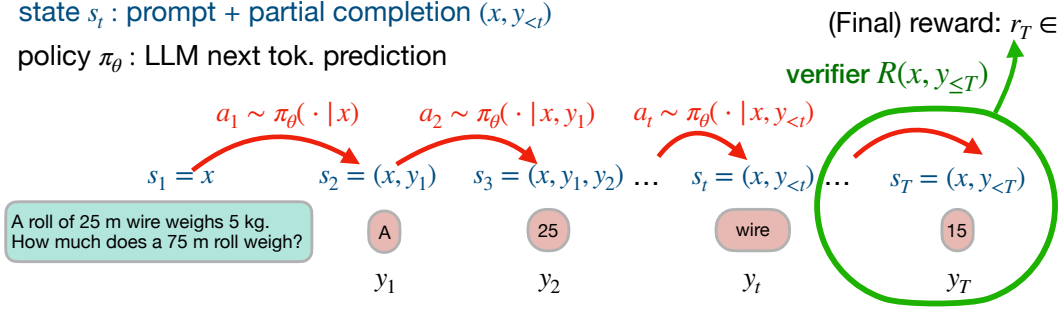

rollout $y_{\leq T}$ : fully AR sampled answer

# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction




rollout $y_{\le T}$ : fully AR sampled answer

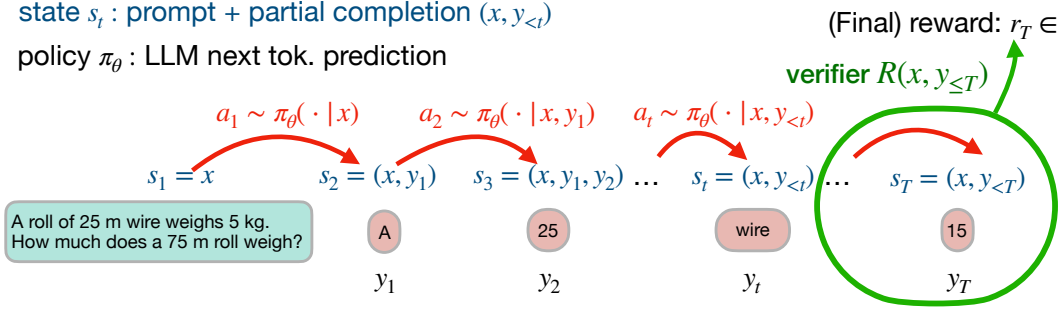
total future reward: $G_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, where $\gamma \in (0,1)$ is fixed discounting

# RL formulation for an LLM

action $a_t$ : sampled token $y_t \sim p_\theta(\cdot \mid x, y_{<t})$

state $s_t$ : prompt + partial completion $(x, y_{<t})$

policy $\pi_\theta$ : LLM next tok. prediction




rollout $y_{\leq T}$ : fully AR sampled answer

total future reward: $G_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, where $\gamma \in (0,1)$ is fixed discounting

Value function: $V^\pi(s_t) = \mathbb{E}_{a \sim \pi_\theta(\cdot \mid s_t)} \left[ G_t \mid s_t \right]$ $\Longleftarrow$ Expected future reward under $\pi_\theta$ from $s_t$

# REINFORCE: Gradient Update on the Policy (LLM)

**Objective:**

$$\max_{\theta} J(\theta) := \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\left[R(x,y)\right]$$

# REINFORCE: Gradient Update on the Policy (LLM)

**Objective:**

$$\max_{\theta} J(\theta) := \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\left[R(x,y)\right]$$

**Idea:** Use gradient update:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta J(\theta)$$

# REINFORCE: Gradient Update on the Policy (LLM)

**Objective:**

$$\max_{\theta} J(\theta) := \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\left[R(x,y)\right]$$

**Idea:** Use gradient update:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta J(\theta)$$

**How to compute $\nabla_\theta J(\theta)$?**

$$\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y}[R(x,y)] = \mathbb{E}_{x,y}[R(x,y)\,\nabla_\theta \log \pi_\theta(y\,|\,x)]$$

# REINFORCE: Gradient Update on the Policy (LLM)

**Objective:**

$$\max_{\theta} J(\theta) := \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\left[R(x,y)\right]$$

**Idea:** Use gradient update:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta J(\theta)$$

**How to compute $\nabla_\theta J(\theta)$?**

$\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y}[R(x,y)] = \mathbb{E}_{x,y}[R(x,y)\,\nabla_\theta \log \pi_\theta(y\,|\,x)]$$

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}) \qquad = \mathbb{E}_{x,y}\left[R(x,y)\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right].$$

# REINFORCE: Gradient Update on the Policy (LLM)

**Objective:**

$$\max_\theta J(\theta) := \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\left[R(x,y)\right]$$

**Idea:** Use gradient update:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta J(\theta)$$

**How to compute $\nabla_\theta J(\theta)$?**

$\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y}[R(x,y)] = \mathbb{E}_{x,y}[R(x,y)\,\nabla_\theta \log \pi_\theta(y\,|\,x)]$$

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}) \qquad = \mathbb{E}_{x,y}\left[R(x,y)\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})\right].$$

**REINFORCE:**

- sample prompt $x \sim D$
- answer $y \sim \pi_\theta(\,\cdot\,|\,x)$
- score reward $R(x,y) \in \mathbb{R}^d$

$$\theta^{(t+1)} = \theta^{(t)} + \alpha\, R(x,y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})$$

# Summary

# Summary

- **We covered:**
  - Evolution of language models into LLMs (motivations, challenges, ...)
  - GPT2-like architecture (tokenizer, transformer, MHA)
  - Further architecture variations (RoPE, SwiGLU, pre-Norm)
  - Inference (prefill vs. generation, memory intensity, GQA)
  - Post-training (SFT, LoRA, RLVR)

# Summary

- **We covered:**
  - Evolution of language models into LLMs (motivations, challenges, ...)
  - GPT2-like architecture (tokenizer, transformer, MHA)
  - Further architecture variations (RoPE, SwiGLU, pre-Norm)
  - Inference (prefill vs. generation, memory intensity, GQA)
  - Post-training (SFT, LoRA, RLVR)
- **Many important topics not covered:**
  - Mixture of Experts
  - Scaling laws
  - Theory + Training dynamics
  - Data-mixtures
  - Technical: Inference vLLM, PagedAttention, FlashAttention.

# Summary

- **We covered:**
  - Evolution of language models into LLMs (motivations, challenges, . . . )
  - GPT2-like architecture (tokenizer, transformer, MHA)
  - Further architecture variations (RoPE, SwiGLU, pre-Norm)
  - Inference (prefill vs. generation, memory intensity, GQA)
  - Post-training (SFT, LoRA, RLVR)
- **Many important topics not covered:**
  - Mixture of Experts
  - Scaling laws
  - Theory + Training dynamics
  - Data-mixtures
  - Technical: Inference vLLM, PagedAttention, FlashAttention.
- **Recommended Sources**:
  - CS336 Language Modeling from Scratch
  - Karpathy's nanoGPT + microGPT, link
  - CS224N NLP with Deep Learning
  - Blog "A History of Large Language Models" by Gregory Gundersen