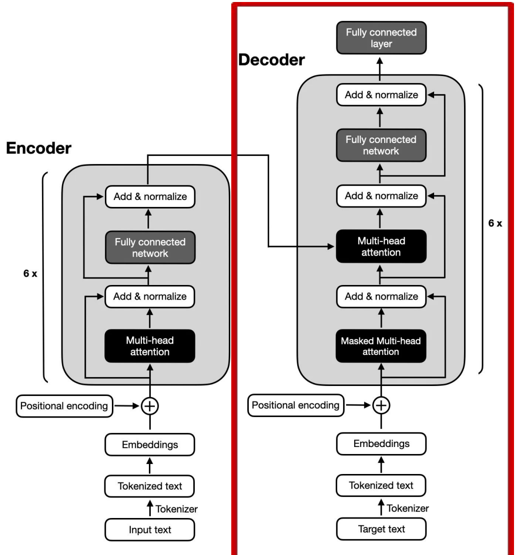
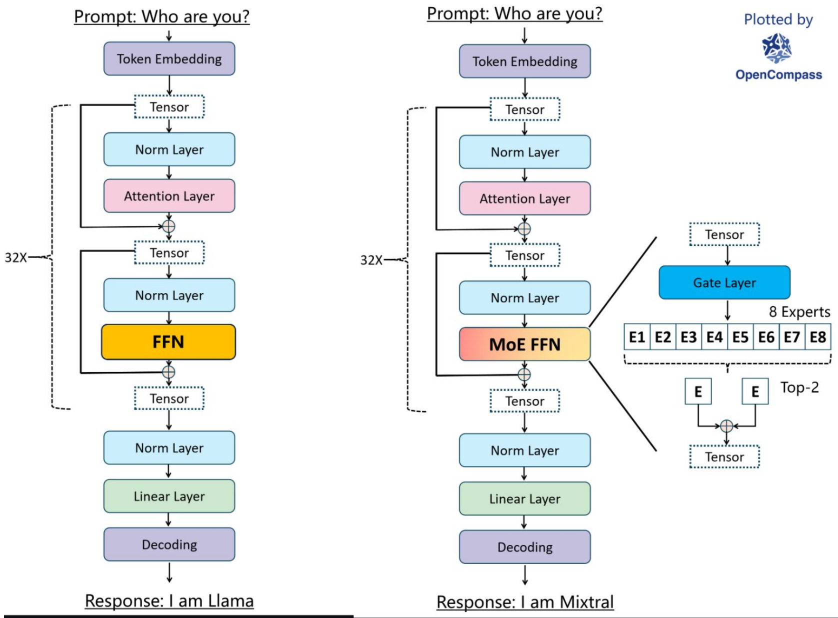
Building Llama3

TA: Jason, Tommy

LLM design variations

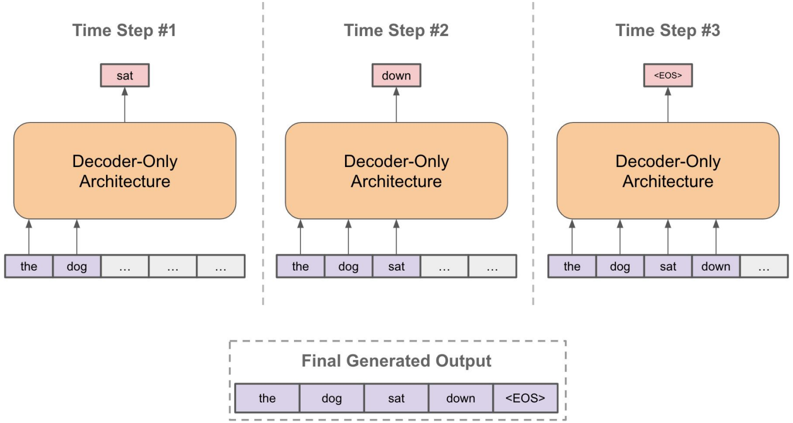
- Architecture:
  - encoder-decoder: Flan-T5, BART
  - **decoder-only**: GPT-series, Llama-series, PaLM
  - mixture of experts: Mixtral (build on decoder)

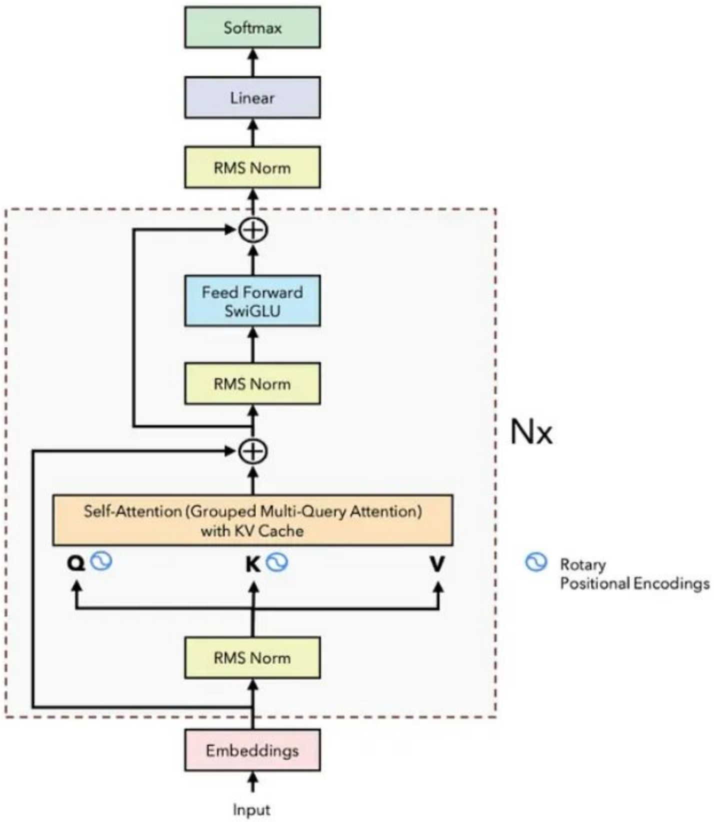




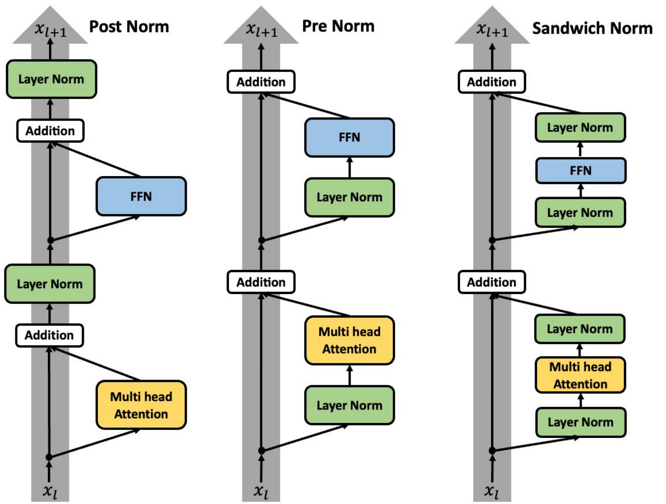
Llama3 architecture

- When autoregrssively generating the next token, decoder can only see preceding tokens.
- Differences between Llama3 and traditional decoder model
  - Rotary positional embedding
  - Grouped Query-attention
  - SwiGLU





design variants -normalization

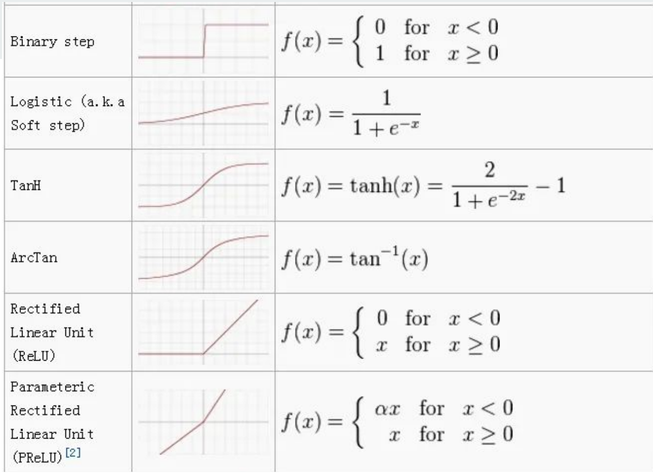


- goal: scale features to a controlled range to stabilize the training of NN
- methods:
  - LayerNorm: normalize across feature dimensions
  - RMSNorm: simplification of LN, faster
  - DeepNorm: learnable weight instead of simple residual, best for very deep NN (>100 layers)
- position:
  - pre
  - post
  - sandwich
- Llama3: pre+RMS Norm
  - Implementation: γ is learned in training, RMS(x) is calculated during forward

Normalization method	LayerNorm [256]	$\frac{\mathbf{x}-\mu}{\sigma} \cdot \gamma + \beta, \quad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu))^2}$
	RMSNorm [257]	$\frac{\mathbf{x}}{\mathrm{RMS}(\mathbf{x})} \cdot \gamma, \quad \mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$
	DeepNorm [258]	$\mathrm{LayerNorm}(\alpha \cdot \mathbf{x} + \mathrm{Sublayer}(\mathbf{x}))$

TABLE 5: Model cards of several selected LLMs with public configuration details. Here, PE denotes position embedding, #L denotes the number of layers, #H denotes the number of attention heads, d_{model} denotes the size of hidden states, and MCL denotes the maximum context length during training.

Model	Category	Size	Normalization	PE	Activation	Bias	#L	#H	d_{model}	MCL
GPT3 [55]	Causal decoder	175B	Pre LayerNorm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α [84]	Causal decoder	207B	Pre LayerNorm	Learned	GeLU	✓	64	128	16384	1024
OPT [90]	Causal decoder	175B	Pre LayerNorm	Learned	ReLU	✓	96	96	12288	2048
PaLM [56]	Causal decoder	540B	Pre LayerNorm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM [78]	Causal decoder	176B	Pre LayerNorm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG [113]	Causal decoder	530B	-	-	-	-	105	128	20480	2048
Gopher [64]	Causal decoder	280B	Pre RMSNorm	Relative	-	-	80	128	16384	2048
Chinchilla [34]	Causal decoder	70B	Pre RMSNorm	Relative	-	-	80	64	8192	-
Galactica [35]	Causal decoder	120B	Pre LayerNorm	Learned	GeLU	×	96	80	10240	2048
LaMDA [68]	Causal decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [107]	Causal decoder	178B	Pre LayerNorm	Learned	GeLU	✓	76	96	13824	2048
LLaMA [57]	Causal decoder	65B	Pre RMSNorm	RoPE	SwiGLU	×	80	64	8192	2048
LLaMA 2 [99]	Causal decoder	70B	Pre RMSNorm	RePE	SwiGLU	×	80	64	8192	4096
Falcon [141]	Causal decoder	40B	Pre LayerNorm	RoPE	GeLU	×	60	64	8192	2048
GLM-130B [93]	Prefix decoder	130B	Post DeepNorm	RoPE	GeGLU	✓	70	96	12288	2048
T5 [82]	Encoder-decoder	11B	Pre RMSNorm	Relative	ReLU	×	24	128	1024	512



Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

design variants -activation functions

- usually used in FFN, allow it to learn complex patterns with non-linear transforms
- methods:
  - Direct non-linear transform
  - GLU (gated linear units): combines non-linear transform and linear transform

$$\mathrm{GLU}(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$

variants: swish function, gaussian error function

- Llama3: SwiGLU
  - implementations: W, V are learned wights, **β** is pre-defined parameter

$$\mathrm{SwiGLU}(x, W, V, b, c, \beta) = \mathrm{Swish}_\beta(xW + b) \otimes (xV + c)$$

$$\mathrm{swish}(x) = x\,\mathrm{sigmoid}(\beta x)$$

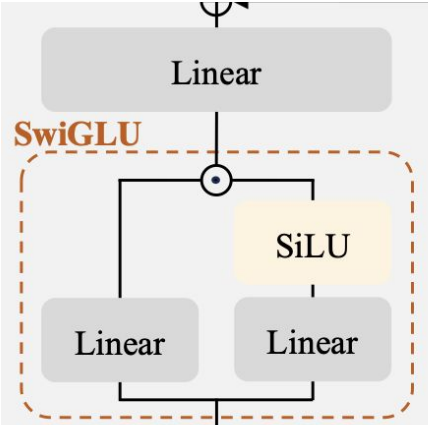


Table 3: SuperGLUE Language-Understanding Benchmark [Wang et al., 2019] (dev).

	Score Average	BoolQ Acc	CB F1	CB Acc	CoPA Acc	MultiRC F1	MultiRC EM	ReCoRD F1	ReCoRD EM	RTE Acc	WiC Acc	WSC Acc
FFN_{ReLU}	72.76	80.15	83.37	89.29	70.00	76.93	39.14	73.73	72.91	83.39	67.71	77.88
FFN_{GELU}	72.98	80.64	86.24	**91.07**	74.00	75.93	38.61	72.96	72.03	81.59	68.34	75.96
$\text{FFN}_{\text{Swish}}$	72.40	80.43	77.75	83.93	67.00	76.34	39.14	73.34	72.36	81.95	68.18	81.73
FFN_{GLU}	73.95	80.95	77.26	83.93	73.00	76.07	39.03	74.22	73.50	84.12	67.71	**87.50**
$\text{FFN}_{\text{GEGLU}}$	73.96	81.19	82.09	87.50	72.00	**77.43**	**41.03**	75.28	**74.60**	83.39	67.08	83.65
$\text{FFN}_{\text{Bilinear}}$	73.81	**81.53**	82.49	89.29	**76.00**	76.04	40.92	74.97	74.10	82.67	**69.28**	78.85
$\text{FFN}_{\text{SwiGLU}}$	**74.56**	81.19	82.39	89.29	73.00	75.56	38.72	**75.35**	74.55	**85.20**	67.24	86.54
$\text{FFN}_{\text{ReGLU}}$	73.66	80.89	**86.37**	**91.07**	67.00	75.32	40.50	75.07	74.18	84.48	67.40	79.81
[Raffel et al., 2019]	71.36	76.62	91.22	91.96	66.20	66.13	25.78	69.05	68.16	75.34	68.04	78.56
ibid. stddev.	0.416	0.365	3.237	2.560	2.741	0.716	1.011	0.370	0.379	1.228	0.850	2.029

TABLE 5: Model cards of several selected LLMs with public configuration details. Here, PE denotes position embedding, #L denotes the number of layers, #H denotes the number of attention heads, d_{model} denotes the size of hidden states, and MCL denotes the maximum context length during training.

Model	**Category**	**Size**	**Normalization**	**PE**	**Activation**	**Bias**	**#L**	**#H**	d_{model}	**MCL**
GPT3 [55]	Causal decoder	175B	Pre LayerNorm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α [84]	Causal decoder	207B	Pre LayerNorm	Learned	GeLU	✓	64	128	16384	1024
OPT [90]	Causal decoder	175B	Pre LayerNorm	Learned	ReLU	✓	96	96	12288	2048
PaLM [56]	Causal decoder	540B	Pre LayerNorm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM [78]	Causal decoder	176B	Pre LayerNorm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG [113]	Causal decoder	530B	-	-	-	-	105	128	20480	2048
Gopher [64]	Causal decoder	280B	Pre RMSNorm	Relative	-	-	80	128	16384	2048
Chinchilla [34]	Causal decoder	70B	Pre RMSNorm	Relative	-	-	80	64	8192	-
Galactica [35]	Causal decoder	120B	Pre LayerNorm	Learned	GeLU	×	96	80	10240	2048
LaMDA [68]	Causal decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [107]	Causal decoder	178B	Pre LayerNorm	Learned	GeLU	✓	76	96	13824	2048
LLaMA [57]	Causal decoder	65B	Pre RMSNorm	RoPE	SwiGLU	×	80	64	8192	2048
LLaMA 2 [99]	Causal decoder	70B	Pre RMSNorm	RePE	SwiGLU	×	80	64	8192	4096
Falcon [141]	Causal decoder	40B	Pre LayerNorm	RoPE	GeLU	×	60	64	8192	2048
GLM-130B [93]	Prefix decoder	130B	Post DeepNorm	RoPE	GeGLU	✓	70	96	12288	2048
T5 [82]	Encoder-decoder	11B	Pre RMSNorm	Relative	ReLU	×	24	128	1024	512

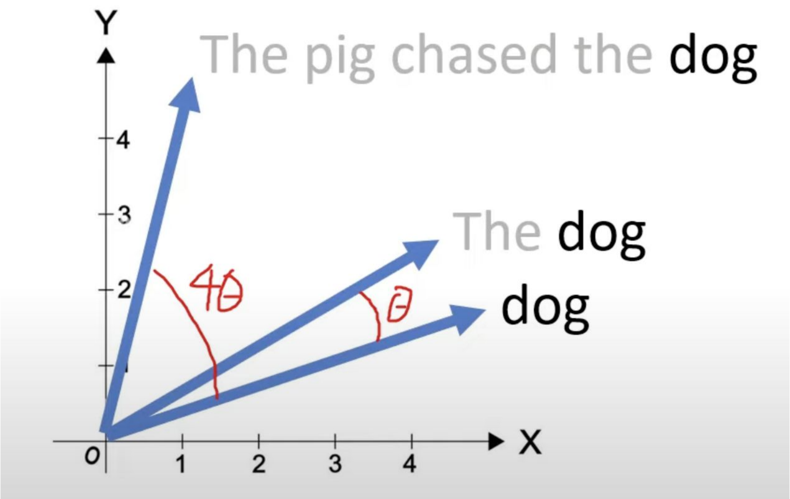
design variants -positional embedding

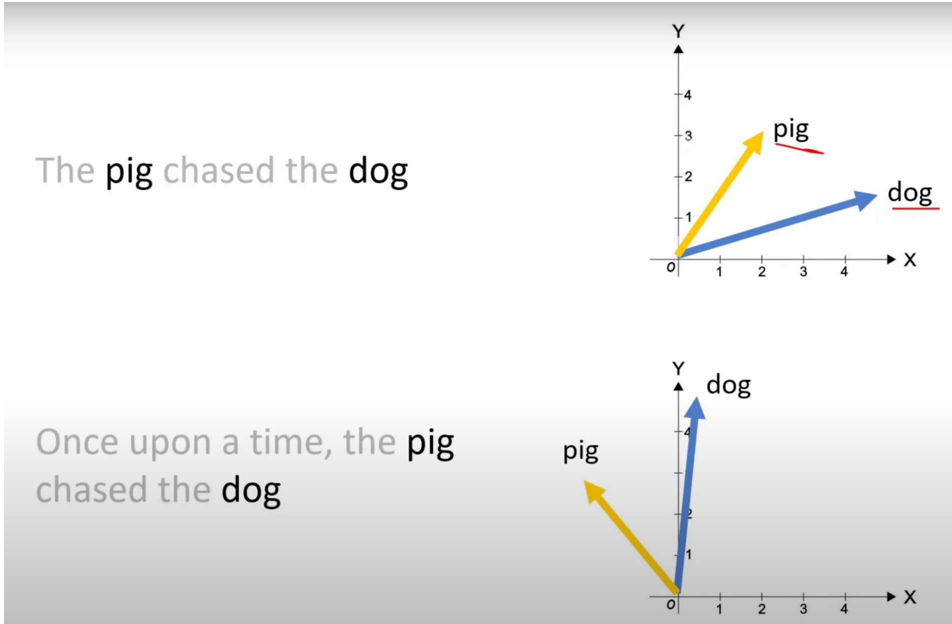
- goal: injects position information for modling language sequence
- methods:
  - ~~absolute position embedding~~: can't generalize to unseen positions, no relative information
  - ~~relative position embedding~~: extra delay, can't use KV cache
  - rotary position embedding: rotate the embedding according to its position and depth(dimension)
    - relative information is indicated by angles
    - can use KV cache
  - Alibi position encoding: better extrapolation ability (generalize to unseen long sequence)
- Llama3: rotary positioin embedding

RoPE detailed

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

- Rotate an embedding based on its position





$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

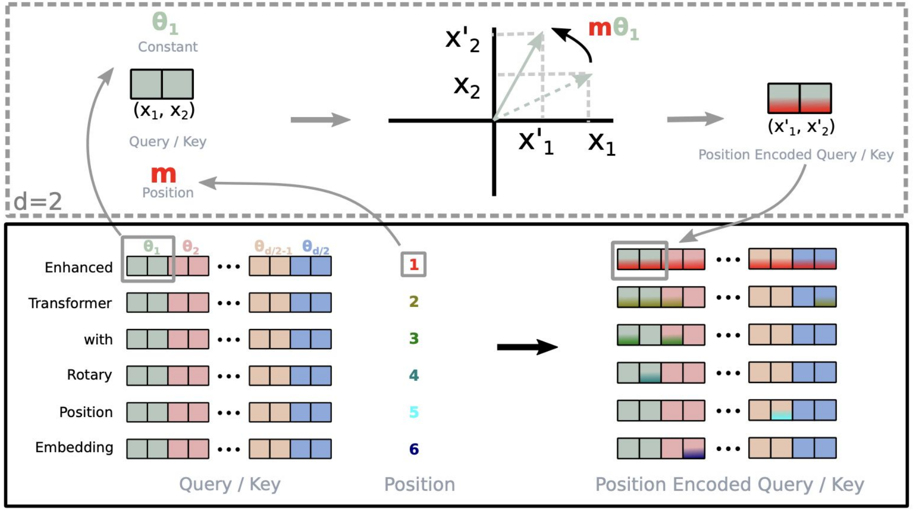
2D rotation

RoPE detailed

- The rotation angle depends on the dimension depth and the position
- Steps:
  1. project embedding X by Query/Value matrix (Wq, Wk)
  2. split Xq, Xk to 2-d embedding
  3. rotate 2-d embedding by mΘi (b: rope theta)
  rotate slower in deeper dimension

$$\Theta = \{\theta_i = b^{-2(i-1)/d} | i \in \{1, 2, \ldots, d/2\}\}.$$

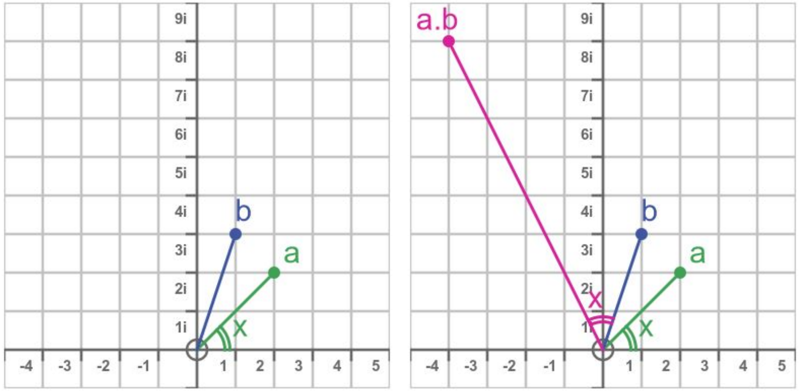
Multi-dimension rotation



Rotation implementation

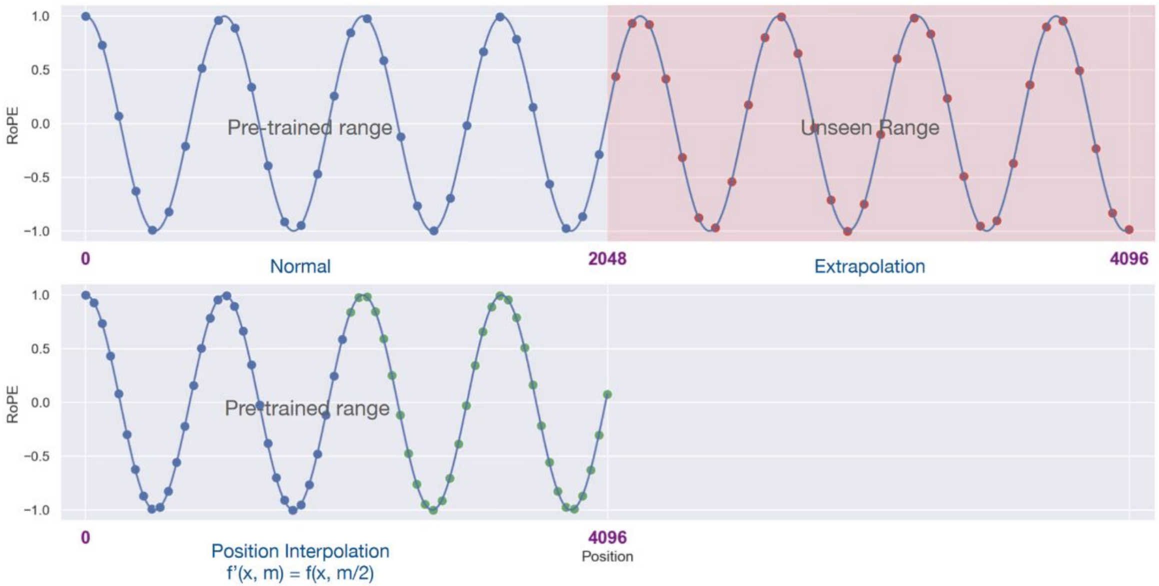
1. Matrix multiplication
2. Complex number multiplication:
   turn 2-d embedding and rotation matrix into complex numbers, multiplication of complex numbers equals to rotation operation.

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$



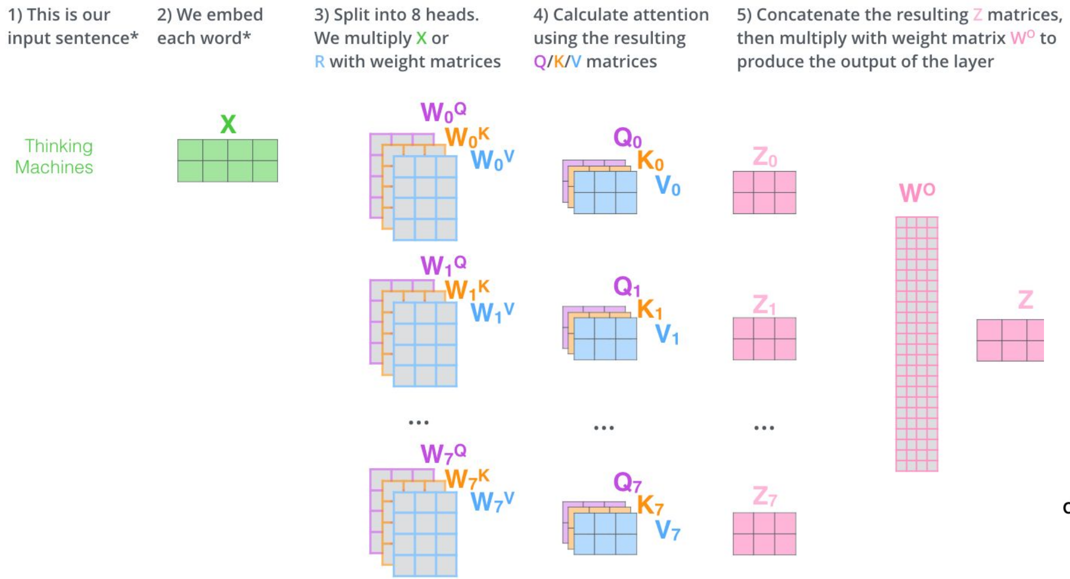
Extrapolation of RoPE

Reference: https://gradient.ai/blog/scaling-rotational-embeddings-for-long-context-language-models



Multi-head attention (brief recap)

$$Attention(Q, K, V) = softmax(\frac{\boxed{QK^T}}{\sqrt{d_k}})V$$



Attention weight between the tokens corresponding to “Life” and “short”

	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

In the row for corresponding to “Life”, mask out all words that come after “Life”

	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

Mask implementation

<TODO> explain -inf and softmax yields 0 prob

Grouped-query attention

Efficiency and Accuracy tradoff
Several query share the same key/value
For Llama3, 32 heads are divided into 8 groups (4 heads per group)

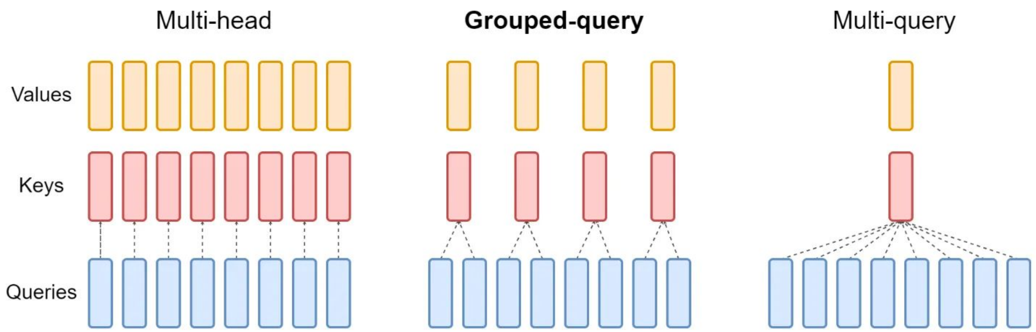
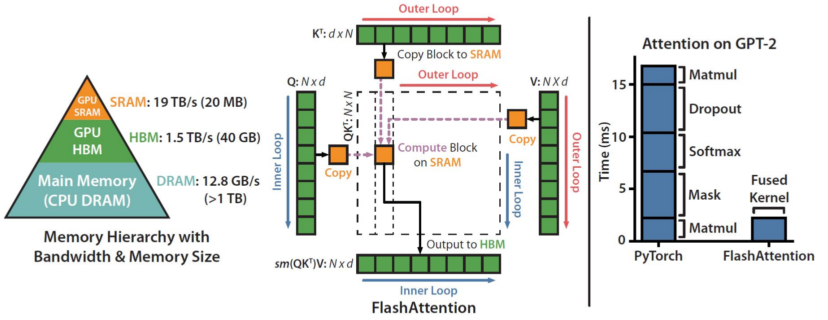
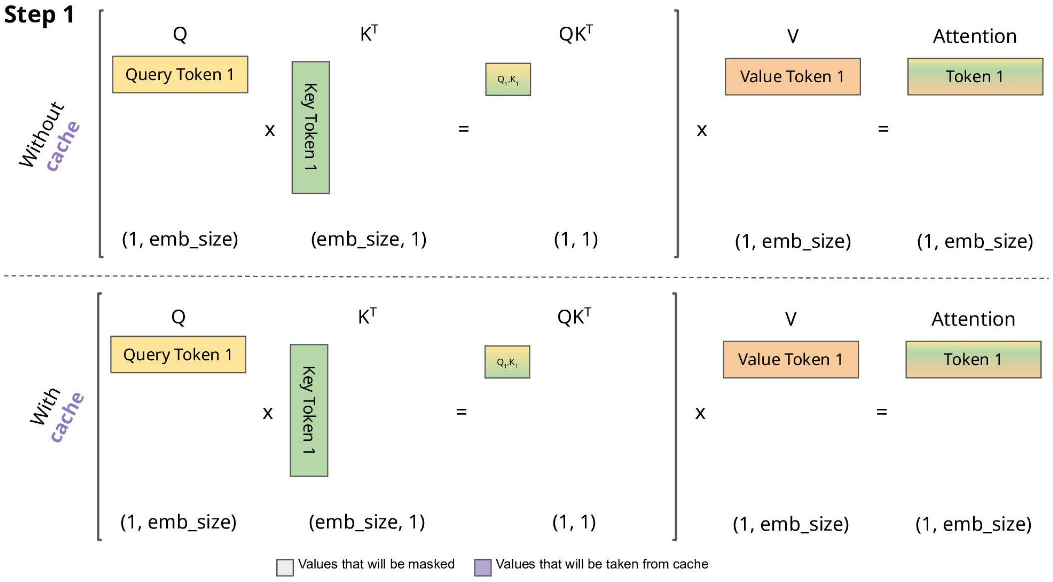


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between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

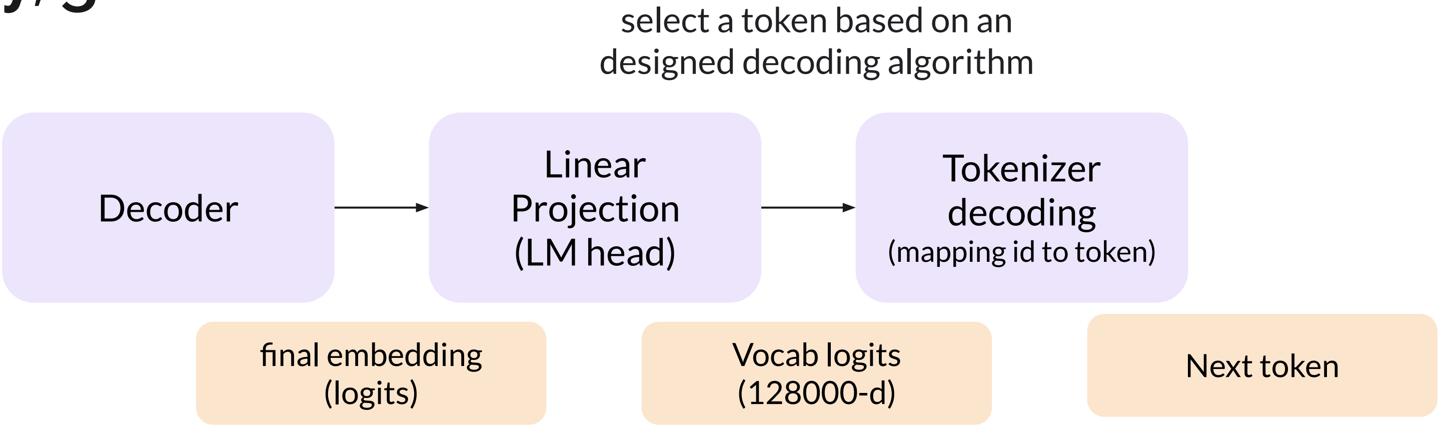
Other attention optimizations

- KV cache:
  cache past keys and values of previous tokens
- Flash attention:
  Hardware optimization - fragment attention computation into blocks to speed up memory IO.
- Paged attention:
  memory-efficient kv caching





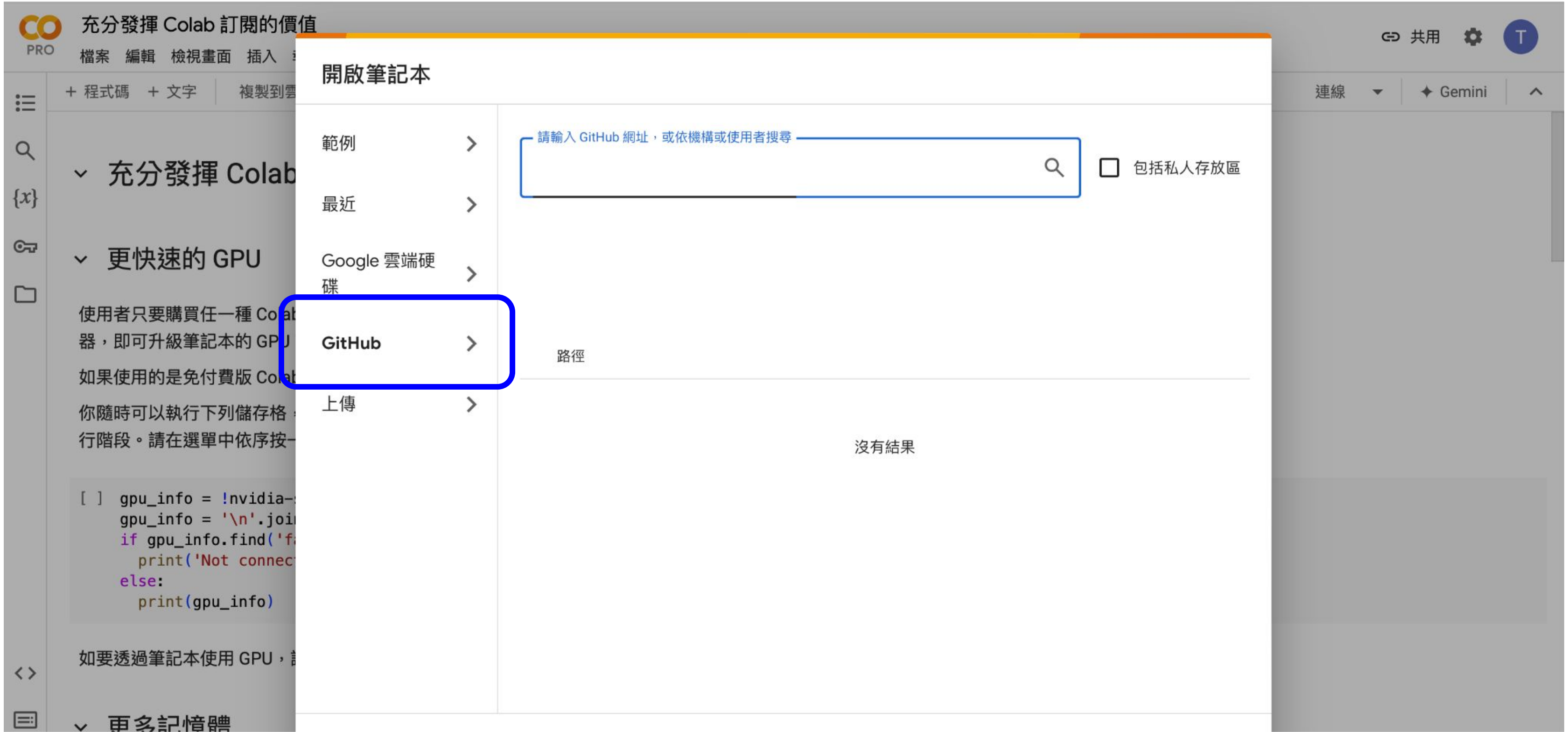
Finally, generate the next token



Lab time!

Github: https://github.com/Huan80805/MediaTek2024

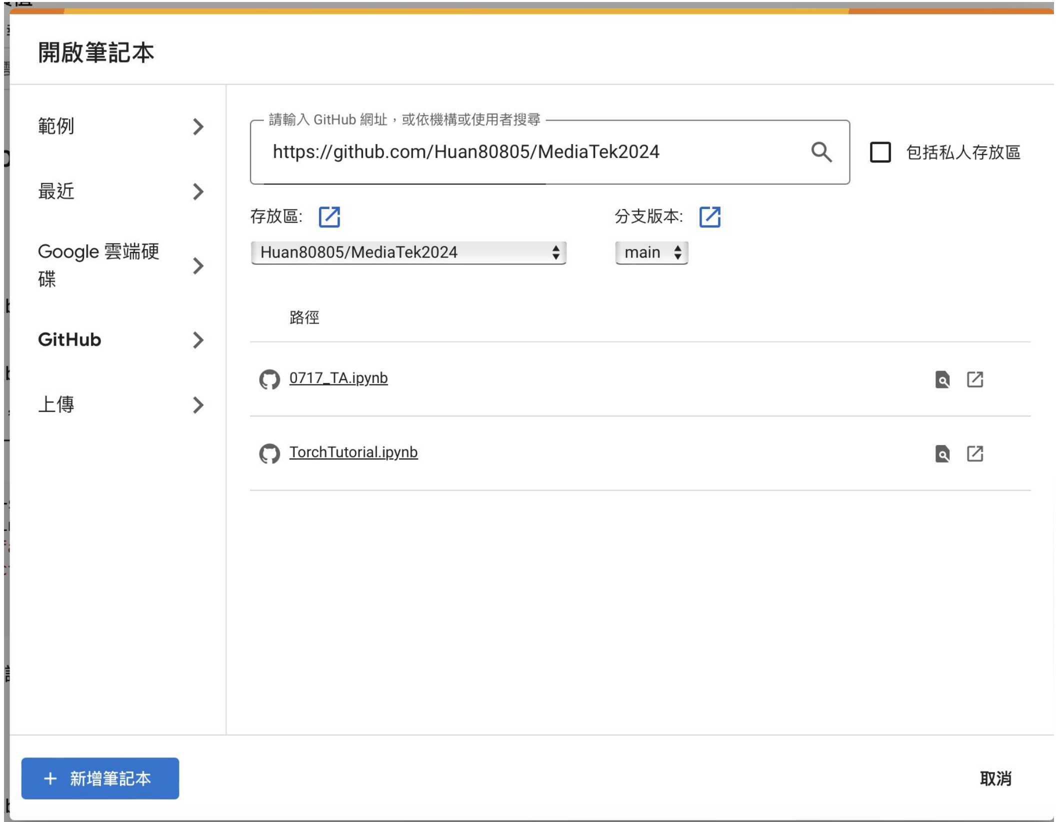
1. Search: colab
2. Open colab
3. Login your google account
4. Choose Github



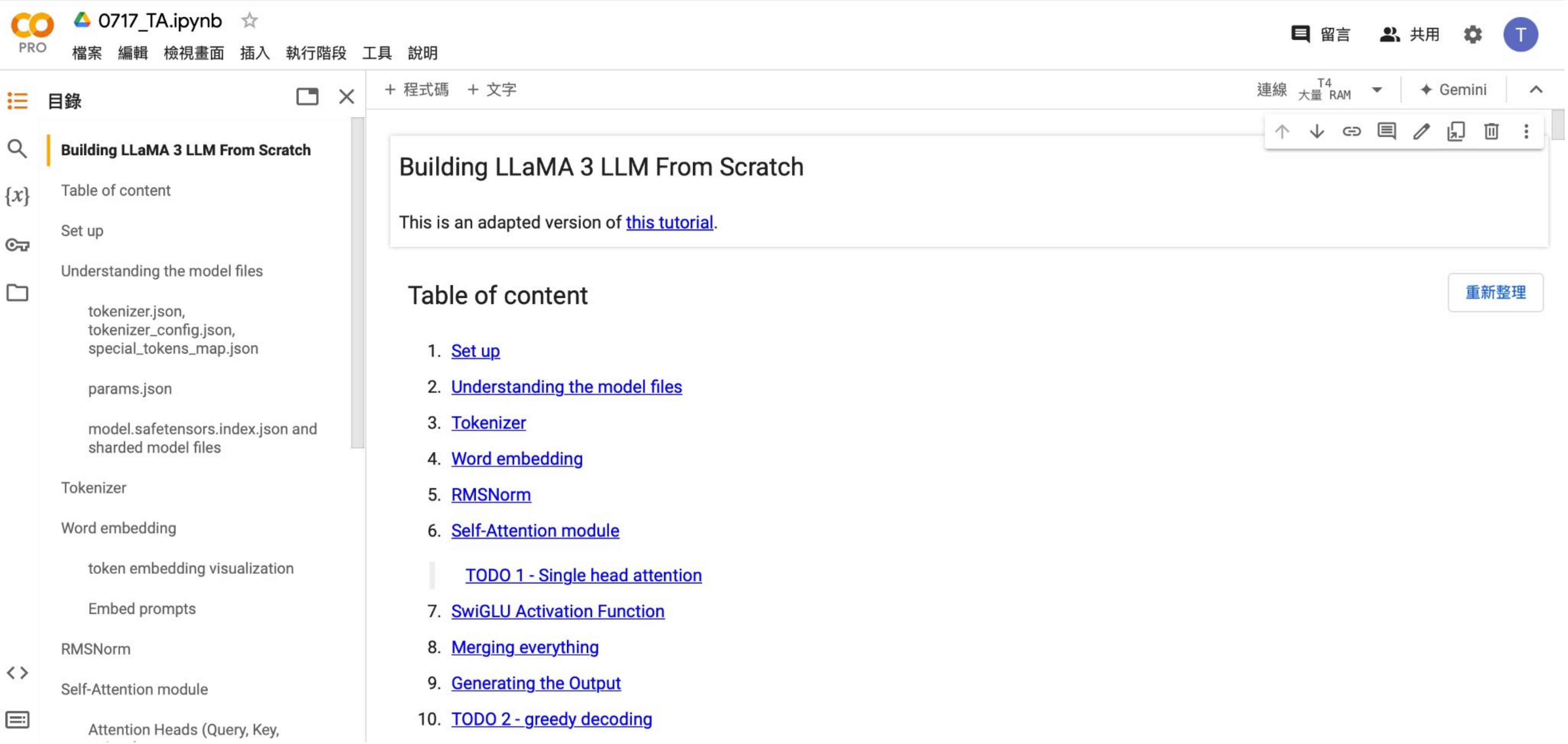
Search the github url: https://github.com/Huan80805/MediaTek2024

不要點選“包括私人存放區”

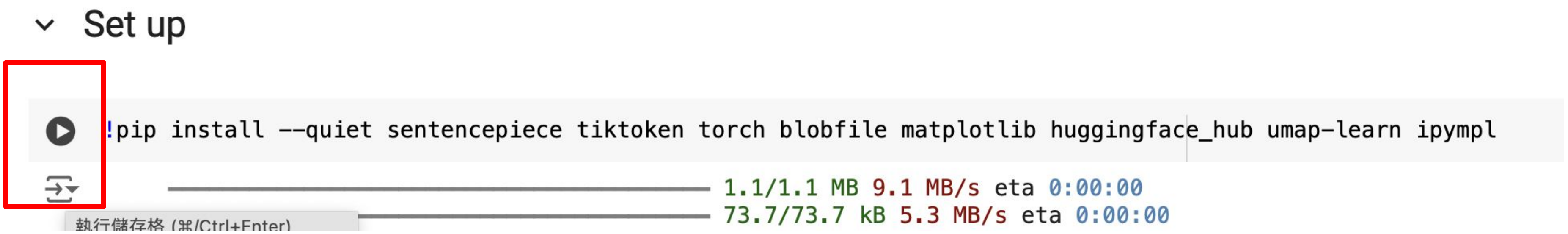
choose “0717_student.ipynb”



If successful, you should see this



登入後確認執行按鈕是否出現，按play選擇仍要執行



Setup 1: pip install 安裝相關套件

Set up

```
!pip install --quiet sentencepiece tiktoken torch blobfile matplotlib huggingface_hub umap-learn ipympl
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 9.1 MB/s eta 0:00:00
```

Setup 2: 在notebook上登入huggingface

```
from huggingface_hub import snapshot_download, notebook_login
notebook_login() # paste your huggingface token
```

Copy a token from your Hugging Face tokens page and paste it below.

Immediately click login after copying your token or it might be stored in plain text in this notebook file.

Token: ••••••••••••••••••••••••••••••••••••

☐ Add token as git credential?

Login

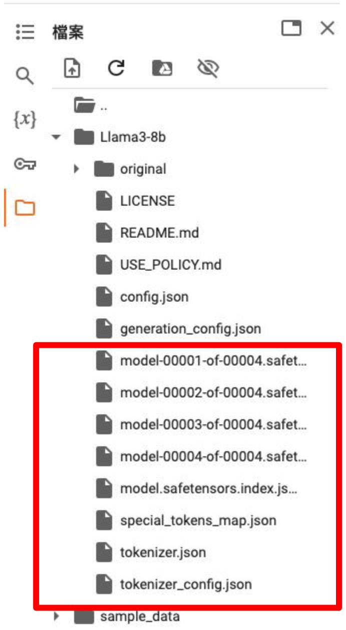
Pro Tip: If you don't already have one, you can create a dedicated 'notebooks' token with 'write' access, that you can then easily reuse for all notebooks.

Setup 3: 下載模型

若申請Llama3 access失敗...
請改用非官方的權重: “NousResearch/Meta-Llama-3-8B”
此備案不需要access token, 在下載模型時將
”meta-llama/Meta-Llama-3-8B”替換成”NousResearch/Meta-Llama-3-8B”即可

```
snapshot_download(repo_id="meta-llama/Meta-Llama-3-8B", local_dir='./Llama3-8b', ignore_patterns=['*.pth'])
```

下載成功後確認資料夾內有Llama3權重



TODO 1 - implement self-attention

1-1: QK matrix multiplication

```
torch.Size([10, 10])
tensor([[ 11.4479,  -4.1105,  -3.8678,  -8.4197, -13.4337, -19.9065, -13.0498,
         -12.5076,  -9.7084, -15.0782],
        [ 25.7426,  -3.4930, -27.2487, -32.8046, -47.2528, -55.9145, -38.3004,
         -32.3618, -20.9183, -42.5390],
        [ 21.9852, -47.8893,  -5.4096, -10.5724, -39.2348, -73.6150, -71.2243,
         -76.7680, -47.8613, -47.5007],
        [  7.7314, -80.2166, -34.3020, -13.2043, -23.2753, -48.1211, -66.5193,
         -88.0660, -57.5897, -58.0146],
        [  8.5806, -48.5645, -31.9186, -19.7255, -10.6553, -15.2242, -26.0849,
         -40.0397, -39.4801, -54.3271],
        [  4.0151, -52.8908, -55.1465, -47.4607, -37.9957, -19.9962, -19.3024,
         -23.7862, -41.0774, -73.0196],
        [  9.0454, -36.0905, -33.6601, -34.3695, -34.7710, -31.2113,  -6.9521,
          -7.8172,  -8.0342, -41.5321],
        [  9.9245, -21.5995, -27.5412, -27.4226, -34.1755, -30.7363, -14.1517,
           0.3500,   5.0962, -18.9832],
        [ -0.3087, -45.6346, -24.5328, -20.4884, -33.3145, -31.9410, -23.9552,
         -16.9752,  10.6239, -15.9402],
        [ -3.6735, -73.1409, -43.8472, -32.9144, -47.6157, -48.3189, -49.4618,
         -47.7429,   2.0054,  -5.7737]], grad_fn=<MmBackward0>)
```

1-2: create attention mask

```
tensor([[0., -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., -inf, -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., 0., -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., 0., 0., -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., 0., 0., 0., -inf, -inf, -inf],
        [0., 0., 0., 0., 0., 0., 0., 0., -inf, -inf],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., -inf],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

TODO 1 - implement self-attention

1-3: softmax to get probability

```
tensor([[1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 2.0073e-13, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 4.4990e-31, 1.2648e-12, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 6.3366e-39, 5.5565e-19, 8.0763e-10, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 1.5187e-25, 2.5750e-18, 5.0804e-13, 4.4238e-09, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 1.9306e-25, 2.0296e-26, 4.4047e-23, 5.6921e-19, 3.7289e-11,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 2.4987e-20, 2.8460e-19, 1.3976e-19, 9.3597e-20, 3.2933e-18,
         1.1269e-07, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 2.0428e-14, 5.3560e-17, 6.0282e-17, 7.0304e-20, 2.1955e-18,
         3.5016e-11, 6.9618e-05, 0.0000e+00, 0.0000e+00],
        [1.7881e-05, 3.6997e-25, 5.3776e-16, 3.0864e-14, 8.2586e-20, 3.2695e-19,
         9.5757e-16, 1.0303e-12, 1.0000e+00, 0.0000e+00],
        [3.4027e-03, 2.3111e-33, 1.2176e-20, 6.8001e-16, 2.8124e-22, 1.3897e-22,
         4.4254e-23, 2.4650e-22, 9.9609e-01, 4.1771e-04]],
       dtype=torch.bfloat16, grad_fn=<ToCopyBackward0>)
```

1-4: compute attention output

```
torch.Size([10, 128])
tensor([[ 0.0154,  0.0008,  0.0334,  ..., -0.0248, -0.0028,  0.0649],
        [ 0.0121, -0.0035,  0.0278,  ..., -0.0243, -0.0012,  0.0554],
        [ 0.0150,  0.0031,  0.0327,  ..., -0.0228,  0.0018,  0.0618],
        ...,
        [ 0.0084,  0.0073,  0.0205,  ..., -0.0197, -0.0087,  0.0286],
        [ 0.0057,  0.0102,  0.0378,  ..., -0.0033, -0.0090,  0.0439],
        [ 0.0195, -0.0034,  0.0110,  ..., -0.0151,  0.0181,  0.0493]],
       dtype=torch.bfloat16, grad_fn=<MmBackward0>)
```

Decoding algorithms

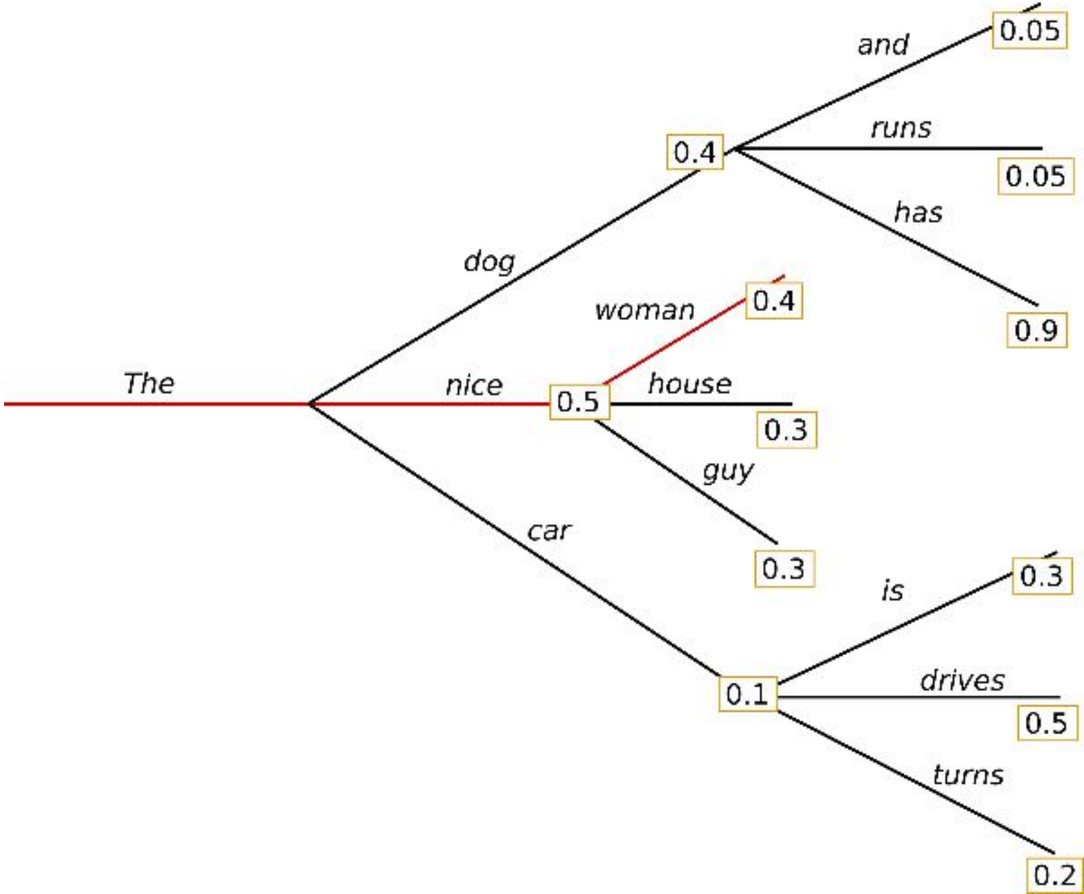
LLM can predict the next token, but what we want is a whole sequence
What decoding does is choose the suitable next token in each step and autoregressively do so, until reaching EOS

Common decoding strategies:
1. greedy decoding
2. beam search
3. sampling
   top-p sampling
   top-k sampling

Greedy decoding

Pick the **most probable token** in each step

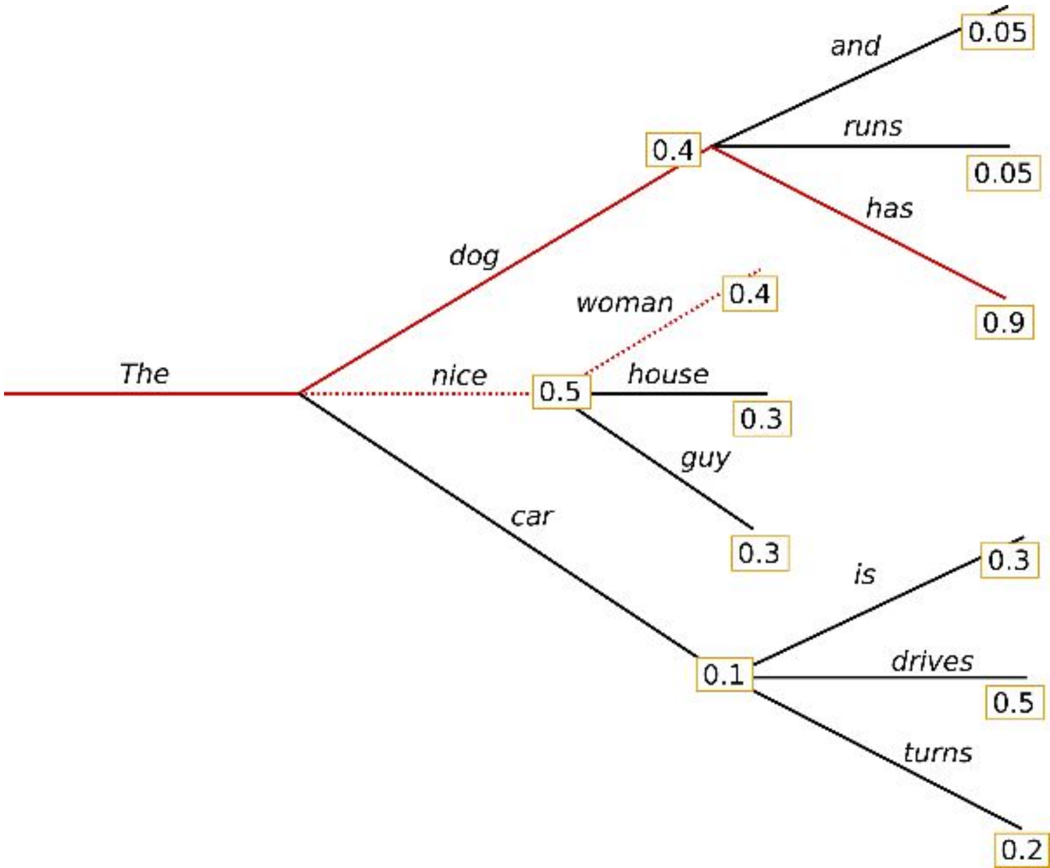
feature :
1. often repeat itself
e.g. I am who I am who I am …
2. some times miss the most probable “sequence”
e.g. The dog has v.s. The nice woman



Beam search

Keep top-k candidates (k is called beam size), choose the most probable sequence in the final step

feature：
1. Still repeat itself, but better when compared with greedy decoding.
2. Sometimes ”repetition penalty” is used to improve quality



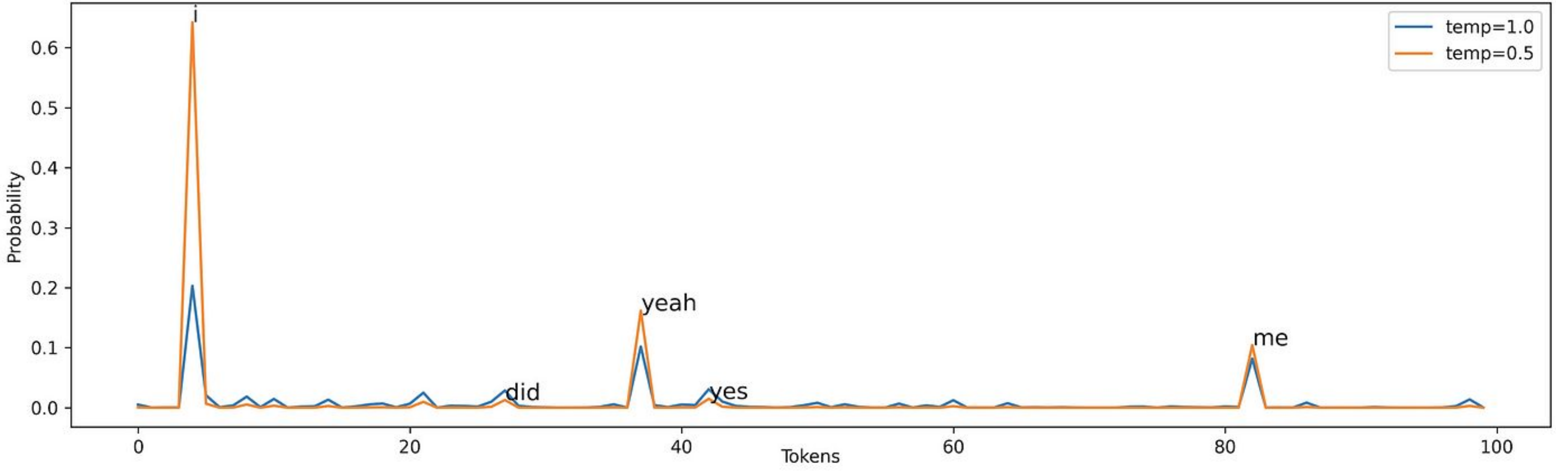
Sampling

$$P(x_i|x_{1:i-1}) = \frac{\exp(u_i/t)}{\sum_j \exp(u_j/t)}$$

Based on current token probability, pick the next toiken **randomly**

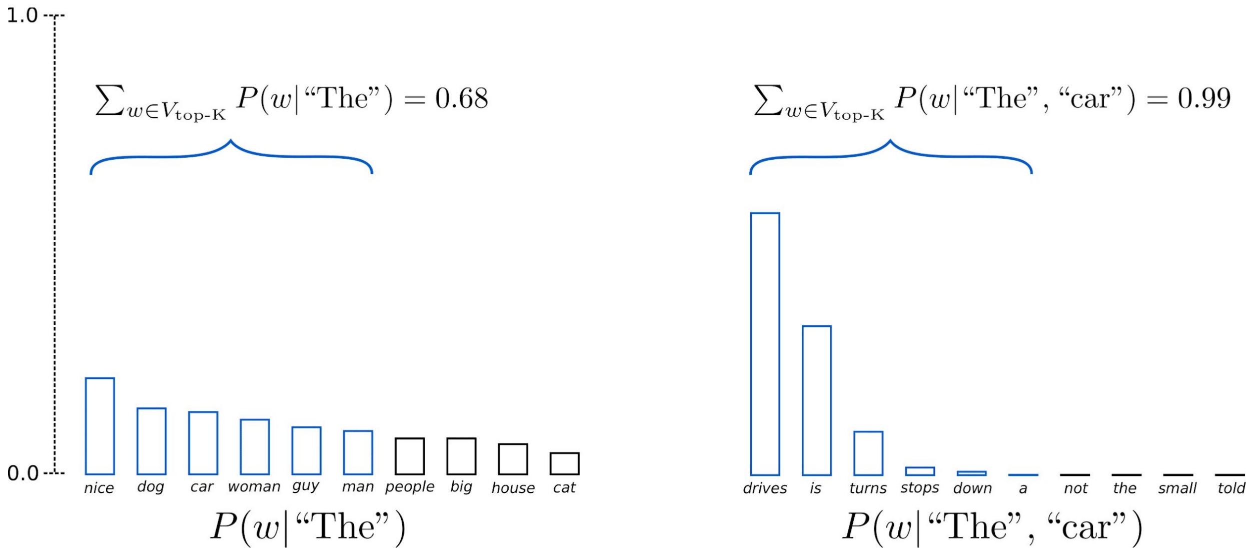
features:

1. increase the generation diversity, particularly suitable for open-ended generation
2. every token can be selected → may accidentally select weird token
3. “temperature” is used to adjust probability (lower temperature → prefer tokens with higher probability)



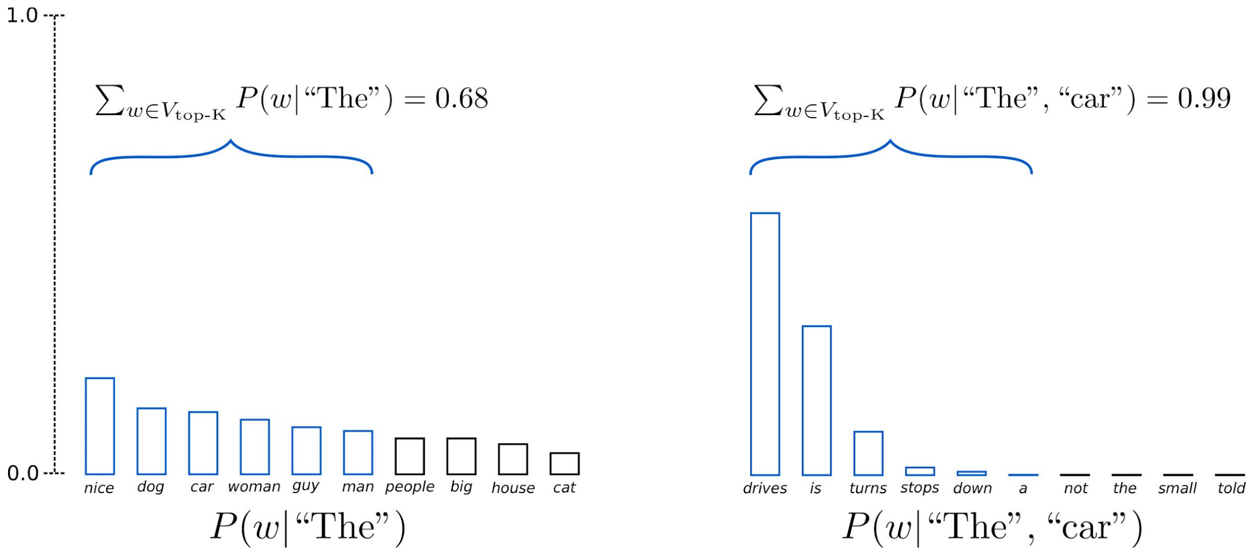
Sampling - top-k sampling

Randomly pick the next token, but only from top-k tokens
When k is large, the text is more diverse, the quality is not ensured.



Sampling - top-p sampling (nucleus sampling)

Randomly pick the next token, but only from the most probable tokens whose cumulative probability sum more than p
Judge tokens based on their probability instead of the count



TODO 2- greedy decoding

```
[74] print(tokenizer.decode(tokens))
```

```
<|begin_of_text|>Life was like a box of chocolates. You never know what you are
```

補充資料

- Transformer walkthrough with Colab: https://github.com/markriedl/transformer-walkthrough/tree/main
- Decoding strategies implementation with Colab: https://colab.research.google.com/drive/19CJIOS5II29g-B3dziNn93Enez1yiHk2
- Survey of LLMs (~2023.12): https://arxiv.org/abs/2303.18223