# SNGULAR

## LLM Distillation. Fine Tuning

**Creating custom LLM model.**

SNGULAR

# Contents:

- What is fine tuning(training-testing)
- Types of fine tuning
  - **Full Fine-Tuning**
  - **LoRA**
  - **QLoRA**
- Dataset preparation
  - **Vectorization**
  - **K-Means(Pairwise distance)**
- Hyperparameters
- Grid Search
- Fine tuning Use case
- Google Cloud Vertex AI.
- Decoder Strategies (Generation Parameters) for LLMs.
- Deployment & integration Use Case.

SNGULAR

# What is fine tuning about?

## Base pretrained optimization function

Let:

- $\mathcal{D}_{\text{pretrain}}$: large dataset used for general pretraining.
- $\theta$: the model parameters (weights) of the LLM.
- The goal is to minimize the loss:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pretrain}}}[\mathcal{L}(f(x;\theta),y)]$$

where:

- $x$: input (e.g. tokenized text),
- $y$: target (e.g. next token in causal language modeling),
- $\mathcal{L}$: loss function (e.g. cross-entropy),
- $f(x;\theta)$: output logits of the model.

## Gradient update rule

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_{\theta}\mathcal{L}(f(x;\theta^{(t)}),y)$$

where:

- $\eta$: learning rate,
- $\nabla_{\theta}$: gradient with respect to the model parameters.

## Fine tuning optimization function

$$\theta_{\text{fine}} = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{fine}}}[\mathcal{L}(\text{DistilledLLM}(x;\theta),y)] \quad \text{with} \quad \theta \leftarrow \theta_S^*$$

# Learning Process

## Train

- This is the **data the model learns from**.
- It **includes both** the input features (**Symptoms**) and the correct output (**Diagnostic**).
- The model uses this data to find **patterns or relationships**.

## Test

- This is **new data the model hasn't seen before**.
- It's used to **check how well the model performs** on unseen examples.
- It helps evaluate whether the model **generalizes well**.

### Example

Training a model to recognize cats vs. dogs in pictures:

- **Training Set:** 80 photos (40 cats, 40 dogs) → Model learns features (like ears, tails).
- **Test Set:** 20 new photos → We check if the model can still correctly tell cats from dogs.

SNGULAR

# Full Fine Tuning

## Cross-Entropy Loss

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log P_\theta(y_t \mid x_{<t})$$

## Gradient Descent or Adam

Updates all parameters θ

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}(f(x;\theta), y)$$

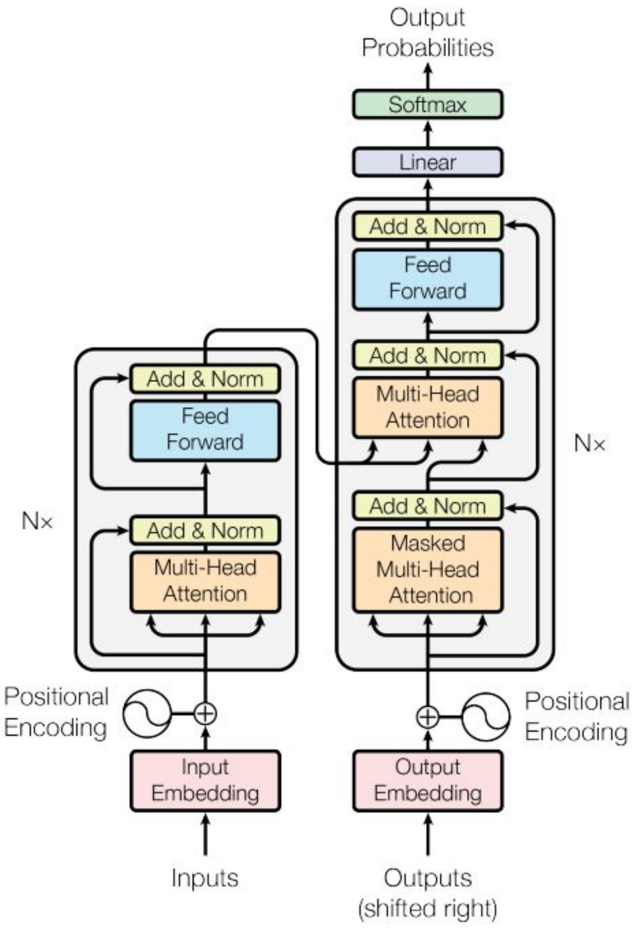
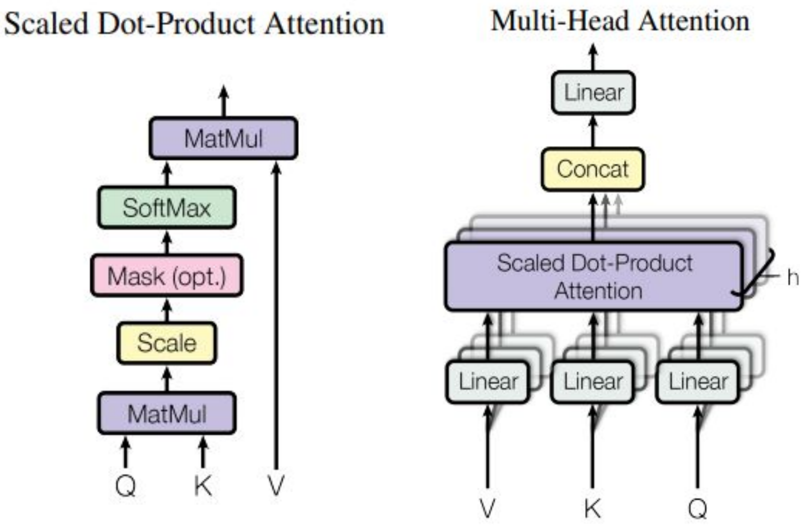



Figure 1: The Transformer - model architecture.

# LoRA Fine Tuning

$$W' = W + \Delta W \quad \text{where } \Delta W = BA,\ A \in \mathbb{R}^{r\times k},\ B \in \mathbb{R}^{d\times r}$$

$$h = Wx + BAx$$



# QLoRA Fine Tuning

Instead of using:

$$h = Wx$$

QLoRA uses:

$$h = \underbrace{\tilde{W}x}_{\text{quantized frozen weights}} + \underbrace{\alpha \cdot BAx}_{\text{trainable LoRA adapters}}$$

$$\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; \tilde{\theta}, \phi),\ y_i)$$

- $\tilde{\theta}$: quantized model weights (frozen)
- $\phi$: LoRA adapters (trainable)
- Only $\phi$ gets updated during training

- $\tilde{W}$ is stored in **4-bit quantized form** (typically using `NF4` quantization).
- The adapter $BA$ is **full-precision (float32 or bfloat16)**.

# Dataset Preparation

*Vectorization*

## 1- Input Sentence

"Hello"

## 2- Character Vocabulary

Assume a vocabulary:

$$\mathcal{V} = \{H, e, l, o\} \quad \Rightarrow \quad \text{Indices: } H = 0,\ e = 1,\ l = 2,\ o = 3$$

## 3- One-Hot Encoding

Each character is converted into a **one-hot vector** of size 4 (the vocabulary size):

| Character | One-hot vector |
|---|---|
| H | [1, 0, 0, 0] |
| e | [0, 1, 0, 0] |
| l | [0, 0, 1, 0] |
| o | [0, 0, 0, 1] |

So the word "Hello" becomes:

$$X = \begin{bmatrix} [1, 0, 0, 0] \\ [0, 1, 0, 0] \\ [0, 0, 1, 0] \\ [0, 0, 1, 0] \\ [0, 0, 0, 1] \end{bmatrix} \quad \in \mathbb{R}^{5 \times 4}$$

SNGULAR

# Dataset Preparation

*Vectorization*

## 4- Embedding Matrix

Let's define a **learned embedding matrix** $E \in \mathbb{R}^{4\times3}$ to map each character to a 3D vector:

$$E = \begin{bmatrix} \mathrm{H} & \rightarrow & [0.1,\ 0.3,\ 0.5] \\ \mathrm{e} & \rightarrow & [0.2,\ 0.1,\ 0.4] \\ \mathrm{l} & \rightarrow & [0.4,\ 0.4,\ 0.4] \\ \mathrm{o} & \rightarrow & [0.7,\ 0.9,\ 0.2] \end{bmatrix}$$

$$X \cdot E = Z \in \mathbb{R}^{5\times3}$$

$$Z = \begin{bmatrix} [0.1,\ 0.3,\ 0.5] \\ [0.2,\ 0.1,\ 0.4] \\ [0.4,\ 0.4,\ 0.4] \\ [0.4,\ 0.4,\ 0.4] \\ [0.7,\ 0.9,\ 0.2] \end{bmatrix} \in \mathbb{R}^{5\times3}$$

SNGULAR

# Dataset Preparation

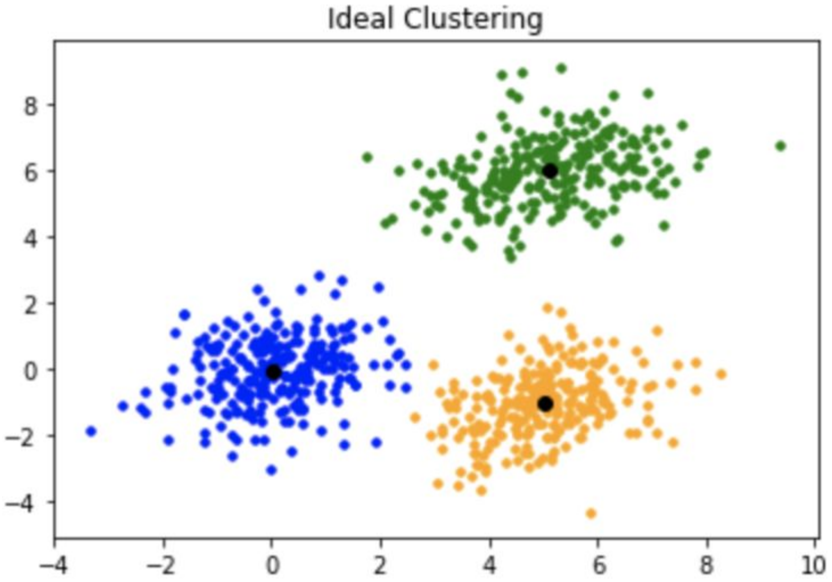
## *K-Means: Statistically Representative Sample*

$$\min_{\{C_1,\ldots,C_K\}} \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Where:

- $C_k$: the set of points assigned to cluster $k$
- $\mu_k$: the **centroid** (mean) of cluster $C_k$:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$



SNGULAR

# Hyperparameters

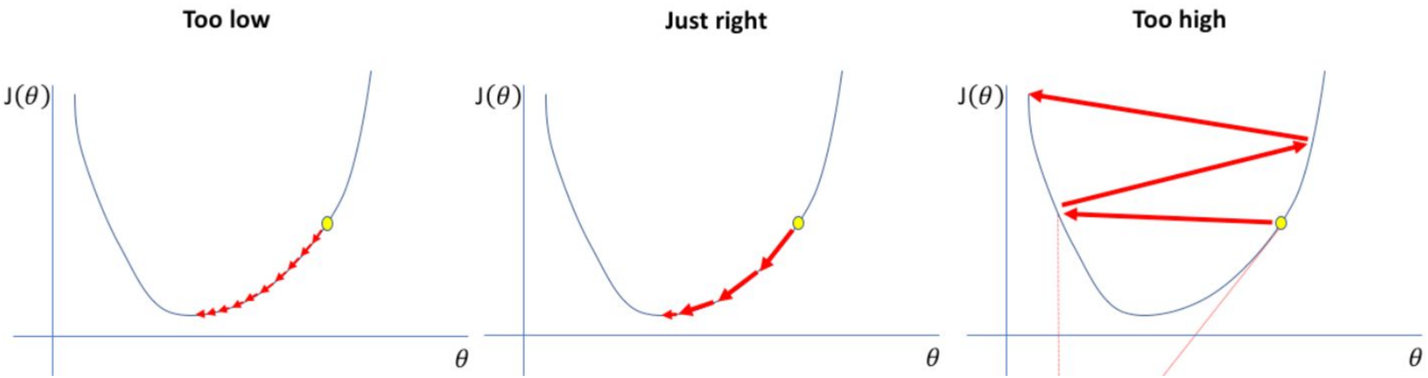
## *Learning Rate*

For a model with parameters $\theta$, and a loss function $\mathcal{L}(\theta)$, the gradient update is:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(t)})$$

Where:

- $\theta^{(t)}$: parameters at iteration $t$
- $\nabla_\theta \mathcal{L}$: gradient of the loss w.r.t. parameters
- $\eta$: **learning rate**



SNGULAR

# Hyperparameters

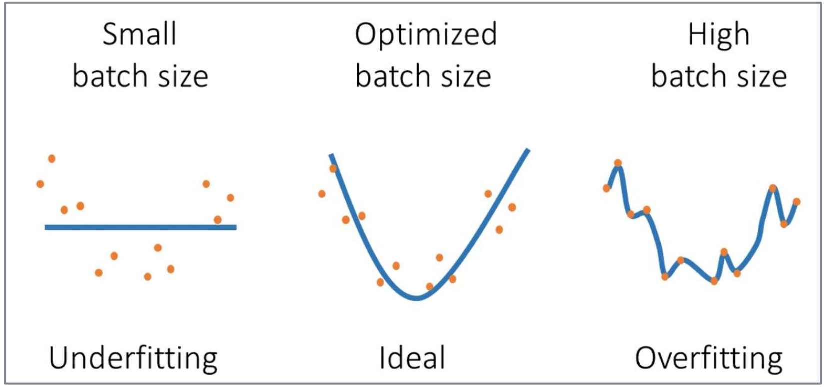
*Batch Size*

**Stochastic Gradient Descent** is approximated to the true gradient:

$$\nabla_\theta \mathcal{L}_{\text{true}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mathcal{L}(\theta; x_i, y_i)$$

With the **mini-batch gradient**:

$$\nabla_\theta \mathcal{L}_{\text{batch}}(\theta) = \frac{1}{B}\sum_{(x_j, y_j)\in\mathcal{B}} \nabla_\theta \mathcal{L}(\theta; x_j, y_j)$$



SNGULAR

# Hyperparameters

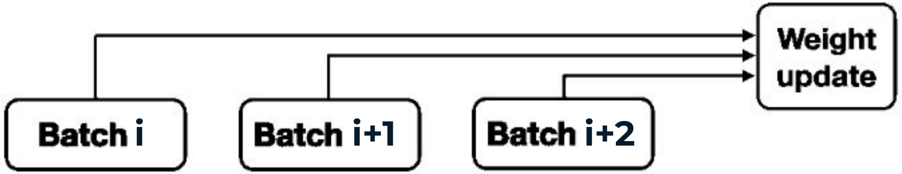
## *Gradient Accumulation*

- $b$: micro-batch size (e.g., 4)
- $g$: **gradient accumulation steps**

$$B = b \cdot g$$

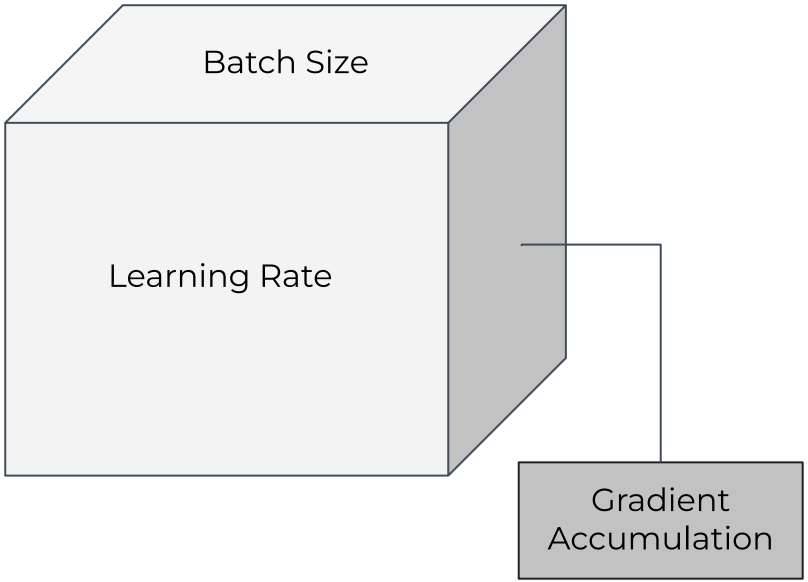
$$\nabla_\theta \mathcal{L}_B = \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \mathcal{L}(\theta; x_i)$$

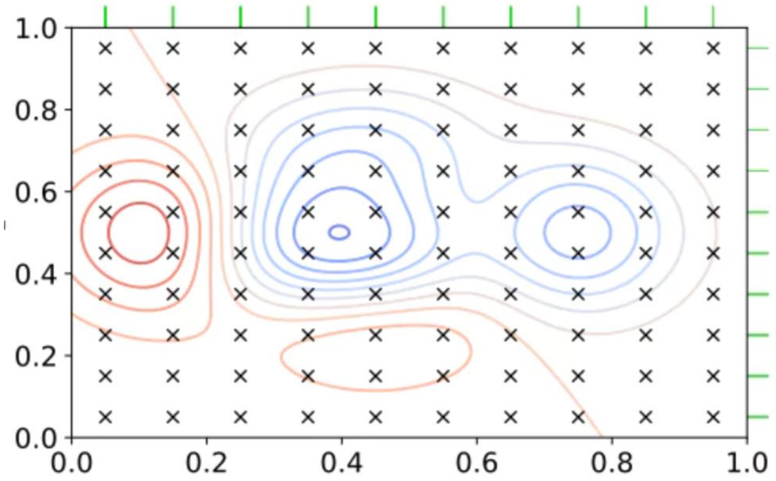
$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_B$$



# Hyperparameters

*Grid Search*



SNGULAR

# Use Case

**01** Find Best Hyperparameters

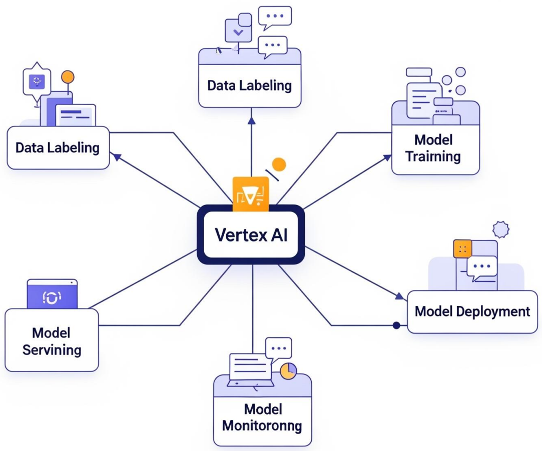
**02** Fine Tune Qwen Distilled Model

**03** Push to Hugging Face

## Deployment & Integration

# LLM Deployment And Service Integration

SNGULAR




# What is Vertex AI?

- Managed Machine Learning Platform
- Supports Custom and AutoML Models.
- Vertex AI accelerates MLOps workflows efficiently.
- It integrates seamlessly with other Google Cloud services.
- Jupyter notebooks and Training Infrastructure.
- Scalable and Cost-Effective(Autopilot).

# Python SDK Google Cloud Vertex AI



- The SDK is part of the `google-cloud-aiplatform` package.
- Simplifies working with models, datasets, endpoints, pipelines, and jobs.
- Initialization:

```python
from google.cloud import aiplatform

aiplatform.init(
    project = "your-project-id",
    location = "us-central1"
)
```

- Create endpoint reference:

```python
ENDPOINT = aiplatform.Endpoint(
    endpoint_name=(
        f"projects/your-project-id"
        f"/locations/us-central1"
        f"/endpoints/your-endpoint-id"
    )
)
```

- Make prediction:

```python
prediction = ENDPOINT.predict(
    instances=["".join(prompt)],
    parameters={
        "temperature": 1.0,
        "max_new_tokens": 512,
        "top_k": 50,
        "top_p": 1.0,
        "repetition_penalty": 1.0
    }
)
```

Cloud SDK

CLI for GCP

SNGULAR

# Generation Parameters
## *Temperature (randomness of predictions)*

Let:

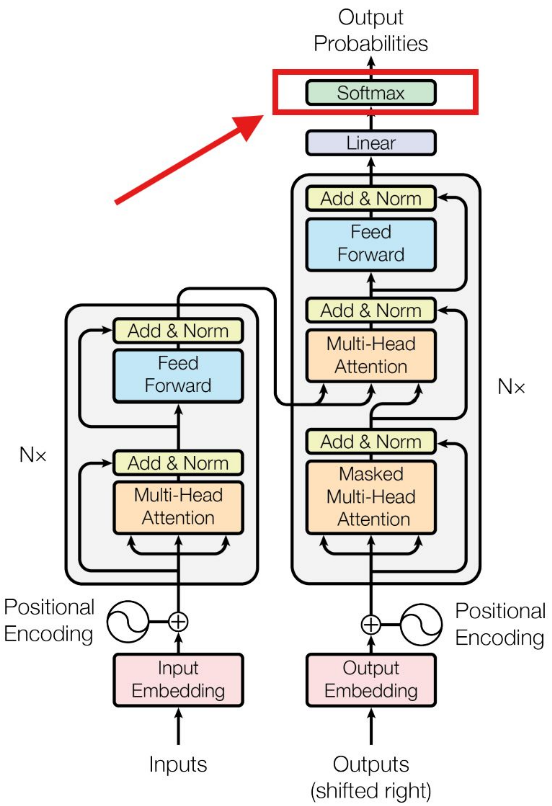
- $z_i$ be the logit (unnormalized score) for token $i$
- $V$ be the vocabulary size

The **standard softmax** computes the probability $p_i$ of token $i$ as:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

To apply temperature $T > 0$, the logits are **divided by** $T$:

$$p_i^{(T)} = \frac{e^{z_i/T}}{\sum_{j=1}^{V} e^{z_j/T}}$$




- $T = 1$: Regular softmax (default).
- $T < 1$: Makes the distribution **sharper** (more confident).
- $T > 1$: Makes the distribution **flatter** (more random).

SNGULAR

# Generation Parameters
*Top K (pick top-k most likely tokens)*

Let:

- $z_i$: logit for token $i$
- $V$: vocabulary of size $N$
- $p_i$: softmax probability for token $i$

Compute probabilities:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

- Identify the **set** $S_k \subseteq V$ of the **k tokens with the highest probabilities.**

$$S_k = \{i \in V \mid p_i \text{ is among the top } k \text{ values in } \{p_j\}_{j=1}^{N}\}$$

## Example (k = 3)

| Token | $p_i$ |
|---|---|
| A | 0.50 |
| B | 0.25 |
| C | 0.15 |
| D | 0.07 |
| E | 0.03 |

SNGULAR

# Generation Parameters
*Top P (pick group of tokens with total probability ≥ p)*

Let:

- $z_i$: logit for token $i \in V$, the vocabulary
- $p_i$: softmax probability

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Create a sorted list of tokens $T = \{t_1, t_2, \ldots, t_N\}$, such that:

$$p_{t_1} \geq p_{t_2} \geq \cdots \geq p_{t_N}$$

Define $S_p$ as the smallest set of tokens such that:

**cumulative probability** $$\sum_{t_i \in S_p} p_{t_i} \geq p$$

## Example (top-p = 0.9)

| Token | $p_i$ |
|---|---|
| A | 0.40 |
| B | 0.30 |
| C | 0.15 |
| D | 0.10 |
| E | 0.05 |

Cumulative sum:

- A: 0.40
- A + B: 0.70
- A + B + C: 0.85
- A + B + C + D: **0.95** → first time ≥ 0.9

SNGULAR

# Generation Parameters

## *Repetition Penalty (penalizes frequent tokens to avoid repetition)*

Let:

- $\mathcal{V}$: vocabulary
- $L \in \mathbb{R}^{|\mathcal{V}|}$: the vector of logits output by the model at time step $t$
- $R > 1$: the **repetition penalty** factor (e.g., 1.1, 1.2, ...)

Let $G = \{g_1, g_2, \ldots, g_{t-1}\}$ be the set of **previously generated tokens**.

For each token $i \in \mathcal{V}$:

$$L'_i = \begin{cases} \frac{L_i}{R} & \text{if } i \in G \text{ and } L_i > 0 \\ L_i \cdot R & \text{if } i \in G \text{ and } L_i < 0 \\ L_i & \text{otherwise} \end{cases}$$

$$p_i = \frac{e^{L'_i}}{\sum_{j \in \mathcal{V}} e^{L'_j}}$$

### **Example ($R = 1.2$)**

- Vocabulary: `["cat", "dog", "mouse"]`
- Original logits: `L = [3.0, 1.0, 0.5]`
- Generated so far: `["cat"]`

Apply the penalty:

- `"cat"`: $3.0/1.2 = 2.5$
- `"dog"` and `"mouse"` stay the same

New logits: `[2.5, 1.0, 0.5]` → lower chance of picking "cat" again

SNGULAR

# Bibliography


- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, *1*(2), 3.
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, *36*, 10088-10115.
- Hamerly, G., & Elkan, C. (2003). Learning the k in k-means. *Advances in neural information processing systems*, *16*.
- Zhu, Y., Li, J., Li, G., Zhao, Y., Jin, Z., & Mei, H. (2024, March). Hot or cold? adaptive temperature sampling for code generation with large language models. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 38, No. 1, pp. 437-445).
- Yerram, V., You, C., Bhojanapalli, S., Kumar, S., Jain, P., & Netrapalli, P. (2024). HiRE: High Recall Approximate Top-$ k $ Estimation for Efficient LLM Inference. *arXiv preprint arXiv:2402.09360*.
- Chu, K., Chen, Y. P., & Nakayama, H. (2024). A better llm evaluator for text generation: The impact of prompt output sequencing and optimization. *arXiv preprint arXiv:2406.09972*.

## Next Talk

- Retrieval-augmented Generation (RAG).
- Grounding and Context.
- Vectorized Database - Distance Measurements.
- Real-world use case:
  - RAG implementation to enhance community well-being.

# Questions

Thanks for your attention!!!