SNGULAR

LLM Deployment And Service Integration

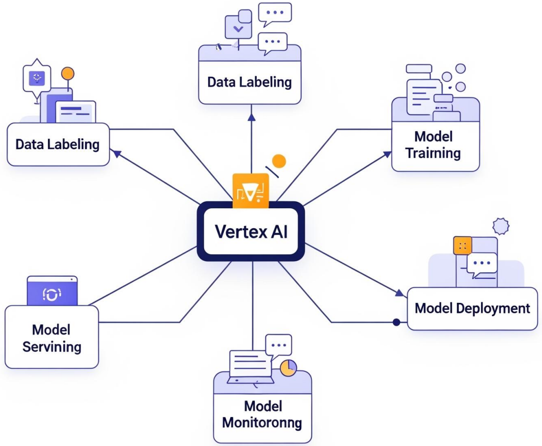
Google Cloud Vertex AI Deployment.

SNGULAR

Contents:

- Google Cloud Vertex AI's and its role in deployment.
- Endpoints and Models.
- Costs of Using Models and Endpoints.
- Service integration with Python SDK Google Cloud Vertex AI.
- Decoder Strategies (Generation Parameters) for LLMs.
- Use Case.

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).

Vertex AI Deployment

- Managed service for deploying LLMs.
- Scalable, robust, and secure infrastructure.
- Simplifies model hosting and serving.
- Integrates with Google Cloud services and Hugging Face for model upload.
- Model Registry.
- Endpoint Creation.
- Model Deployment to Endpoint.
- Prediction Options (REST API calls and batch prediction).
- Monitoring and Logging (Google Cloud Monitoring and Logging).

Endpoint, Model Deployment

- First step is to define and create an Endpoint for model serving.
- Deploy the trained model to the created Endpoint.
- Endpoint handles predictions and serves the model.
- Endpoints can be created using **Google Cloud Console**, **Vertex AI Python SDK** or **gcloud CLI**.
- Creating an endpoint **does not automatically deploy a model**.

Model & Endpoint Costs

- No cost for empty endpoint.
- Billing starts after deployment.
- Charged for allocated VM resources (e.g., CPUs, RAM, GPUs) whether or not predictions are being made:
    - **Per-hour pricing** for each replica.
    - Based on the **machine type** (e.g., `g2-standard-16, NVIDIA_L4,` etc.)
- Online prediction charges

https://cloud.google.com/compute/all-pricing?hl=en
https://cloud.google.com/vertex-ai/pricing?utm_source=chatgpt.com#text-data

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
    }
)
```

Generation Parameters
Temperature (randomness of predictions)

Let:

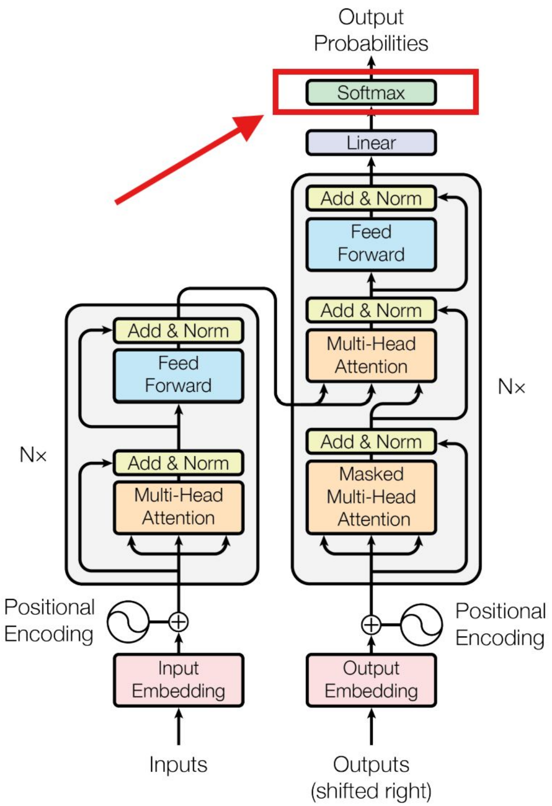
- z_i be the logit (unnormalized score) for token i
- V be the vocabulary size

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).

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

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

Use Case

01 Upload Hugging Face model to Google Vertex AI Model Registry

02 Create endpoint and deploy model into Vertex AI

03 Test model using https request

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*.

- 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*.

SNGULAR

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!!!