IT Talent Training Course

Aug. 2025.

A.I. PROGRAMMING WITH PYTORCH

Instructor :
Daesung Kim

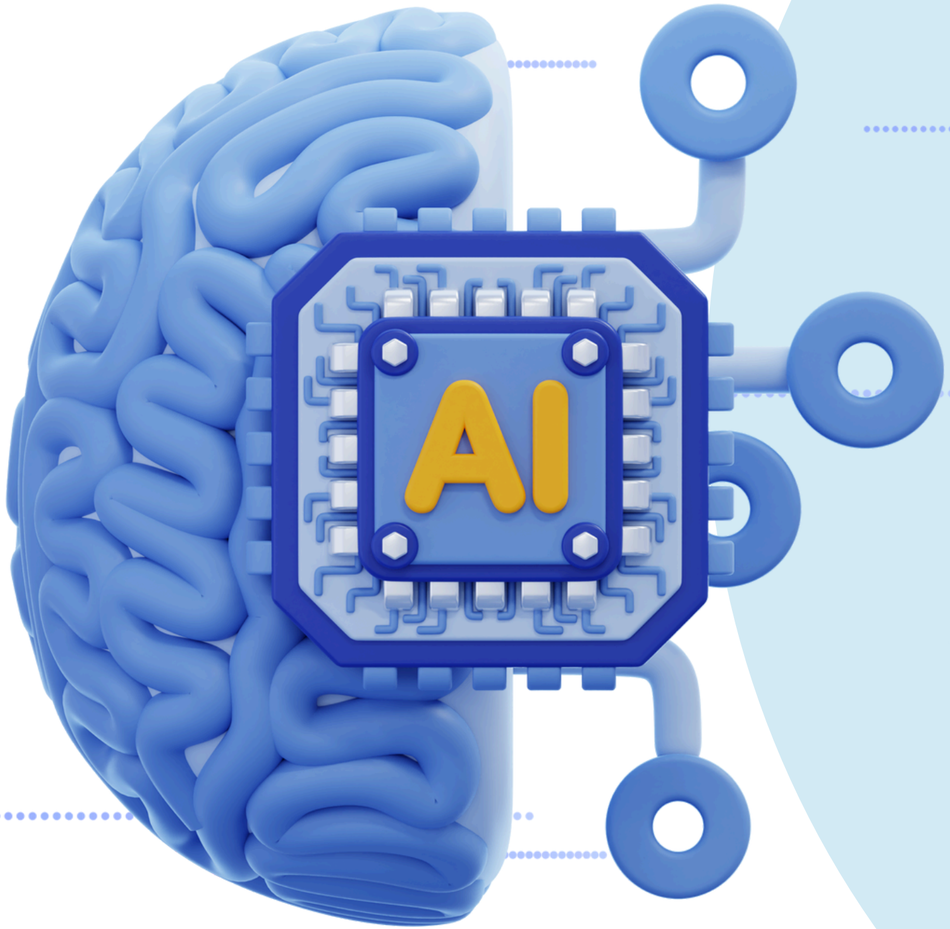
7th Day – Part 01

INDEX

COURSE OBJECTIVES

- You can explain the inherent limitations of sequential processing in RNN/LSTM models.
- You can understand the concept of 'Attention', which is the core idea of the Transformer model, and explain the roles of Query, Key, and Value.
- You can explain the calculation process step by step of how 'Self-Attention' learns the relationship between words in a sentence.
- You can understand the innovation of the Transformer architecture, which is the basis of modern natural language processing models, and explain its significance.

01
TRANSFORMER



LIMITATIONS OF RECURRENT NEURAL NETWORKS (RNNS) - 1

- Sequential Data
- Recurrent Neural Network (RNN)
- Long-Term Dependency Problem
- Vanishing/Exploding Gradient
- LSTM (Long Short-Term Memory)

Last time, we learned about **RNN and LSTM** for processing data where 'order' is important, such as text or time series data. RNN tried to understand the context through a recurrent structure that stores information from the previous time step in the 'hidden state' and passes it to the next time step. However, this structure **had several inherent limitations.**

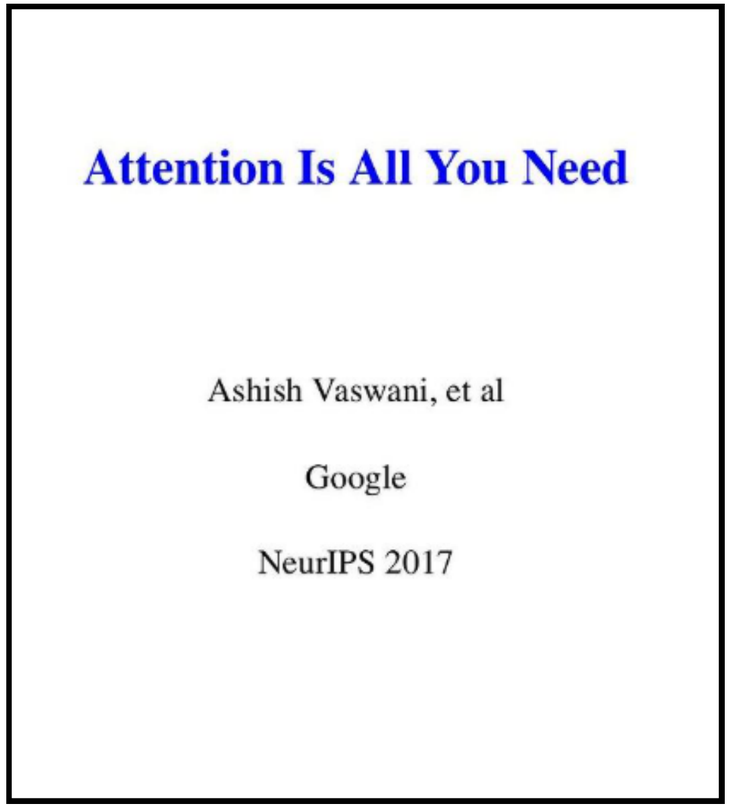
LIMITATIONS OF RECURRENT NEURAL NETWORKS (RNNS) - 2

- **Long-Term Dependency Problem:** As sentences get longer, there is a problem that important information (e.g., subject) at the beginning of the sentence becomes blurred as it goes on. LSTM significantly alleviates this problem by introducing 'cell state' and multiple 'gates', but it does not completely solve it.

- **Difficulty of parallel processing:** This is the biggest structural limitation of RNN and LSTM. In order to calculate the hidden state at time t, the calculation at time t-1 must be completed. This is like dominoes, where the first block must fall before the next block can fall. Because of this sequential calculation method, it was difficult to fully utilize the advantage of GPUs that process numerous operations simultaneously and increase speed. No matter how much data there was, the model had to process one step at a time in sequence.

BACKGROUND OF THE TRANSFORMERS' APPEARANCE

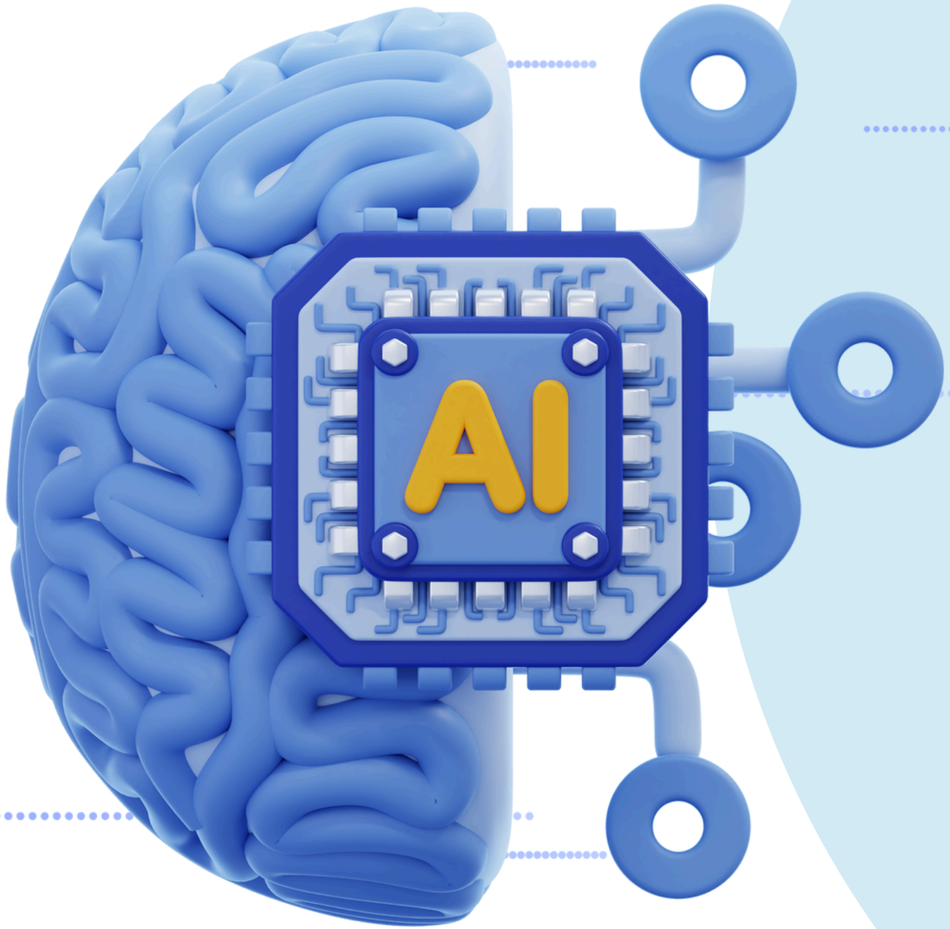
Against this backdrop, in 2017, Google researchers published a landmark paper titled “Attention Is All You Need,” which revolutionized the field of AI, particularly natural language processing (NLP).



This paper, as the name suggests, declares, "Attention is all you need!" and completely removes the 'recurrence' structure, which was a chronic problem of RNN/LSTM. Instead, it proposes a mechanism called 'self-attention' that can calculate the relationship of all words in a sentence at once and in parallel. This is optimized for GPU computation, which not only dramatically increases the learning speed, but also surpasses all existing models in terms of performance.

The transformer we will learn today is based on this 'attention', and is the core architecture that forms the basis of almost all large-scale language models (LLMs), including ChatGPT and Gemini.

02
ATTENTION MECHANISM



BASIC IDEA: "NOT ALL WORDS NEED EQUAL ATTENTION”

Let's imagine that we are translating the sentence "I did my homework at school yesterday." When translating the verb 'did' into English, which word in the sentence should we pay the most attention to? Perhaps words like 'I' and 'homework' are more important than 'yesterday' or 'at school'.

The attention mechanism is a way to calculate the correlation between a specific word and all the other words in the sentence when processing it, and **quantify which word to focus on more**. It assigns a different 'weight' to each word, so that more information is reflected in words with high correlation, and less information is reflected in words with low correlation.

QUERY, KEY, VALUE - 1

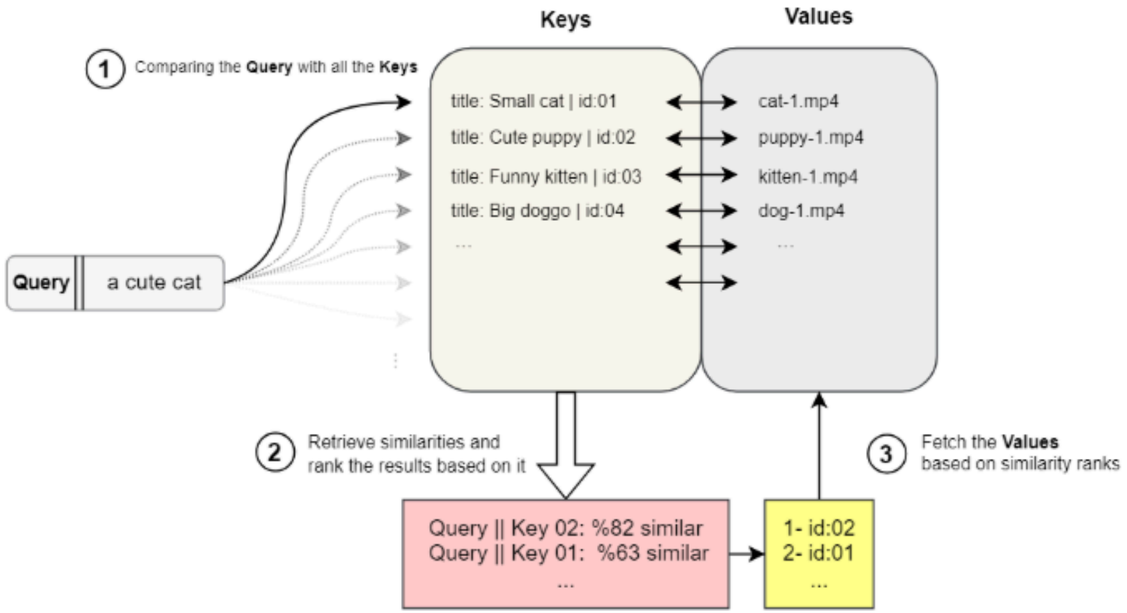
Attention divides this 'finding words to focus on' process into three roles: Query, Key, and Value. It's similar to the process of finding a book in a library.

- **Query (Q):** 'The information I'm looking for now'. You can think of it as the word you're currently processing. (Example: I want to figure out the meaning of the word 'did'.)
- **Key (K):** 'The index or keyword of the book'. It's a unique feature or identifier that all other words in the sentence have. The Query is compared to these Keys and finds out how related they are. (Example: 'I', 'yesterday', 'at school', 'homework', other words in the sentence.)
- **Value (V):** 'The actual content of the book'. It's the actual information value linked to the Key. When the Query calculates the weight through the similarity with the Key, the weight is multiplied by the Value and reflected in the final result.

QUERY, KEY, VALUE - 2

How Attention Works:

- The current word (Query) is calculated as a 'relation score (similarity)' with all the words (Key) in the sentence.
- Based on this score, a 'focus (weight)' for each word is determined.
- The actual information (Value) of each word is multiplied by this weight and added together. This becomes a 'new expression that understands the context' for the current word.



SELF-ATTENTION - 1

https://jalammar.github.io/illustrated-transformer/

- **Matrix Multiplication**
- **Dot Product**
- **Softmax Function**

The core of the transformer is this 'self-attention'. The reason the word 'self' is attached is because the sentence applies attention to 'itself (other words in the sentence)' rather than external information to understand the context.

- For example, in the sentence "The animal didn't cross the street because it was too tired.", does 'it' refer to 'animal' or 'street'? People can easily tell that 'it' refers to 'animal' through the context. Self-attention allows the machine to do exactly this. When processing 'it', it calculates the relationship with other words in the sentence such as 'animal' and 'street' and determines that the word most closely related to 'it' is 'animal'.

SELF-ATTENTION - 2

Step 1: Create Query, Key, Value Vectors

- First, three types of vectors are created from the embedding vector of each word (a word expressed as a number), namely, Query, Key, and Value vectors. These are created by multiplying each word's embedding vector by different weight matrices (W^Q, , W^K, , W^V). These weight matrices are values that the model learns on its own during the learning process.
    - $Query = Embedding \times W^Q$
    - $Key = Embedding \times W^K$
    - $Value = Embedding \times W^V$

Step 2: Calculating Attention Score

- The current word's Query vector is then dot-producted with each of the Key vectors of all words in the sentence (including itself). This score indicates how related the two words are. The dot product of the Query vector of 'it' with the Key vector of 'animal' will be much larger than the dot product of the Key vector of 'street'.

$$Score(q_i, k_j) = q_i \cdot k_j$$

SELF-ATTENTION - 3

Step 3: Scaling

- The inner product value can become unstable as the dimension of the vector increases because its value increases too much. To prevent this, it is divided by the square root of the dimension of the Key vector($\sqrt{d_k}$). This is a kind of normalization process.

$$ScaledScore = \frac{Score}{\sqrt{d_k}}$$

Step 4: Applying the Softmax function

- We apply the softmax function to the scaled scores. Softmax transforms all scores into values between 0 and 1, so that their sum is 1. The transformed values become the 'attention weights' for each word. In other words, it is a probability distribution that represents how much 'it' should focus on each word.

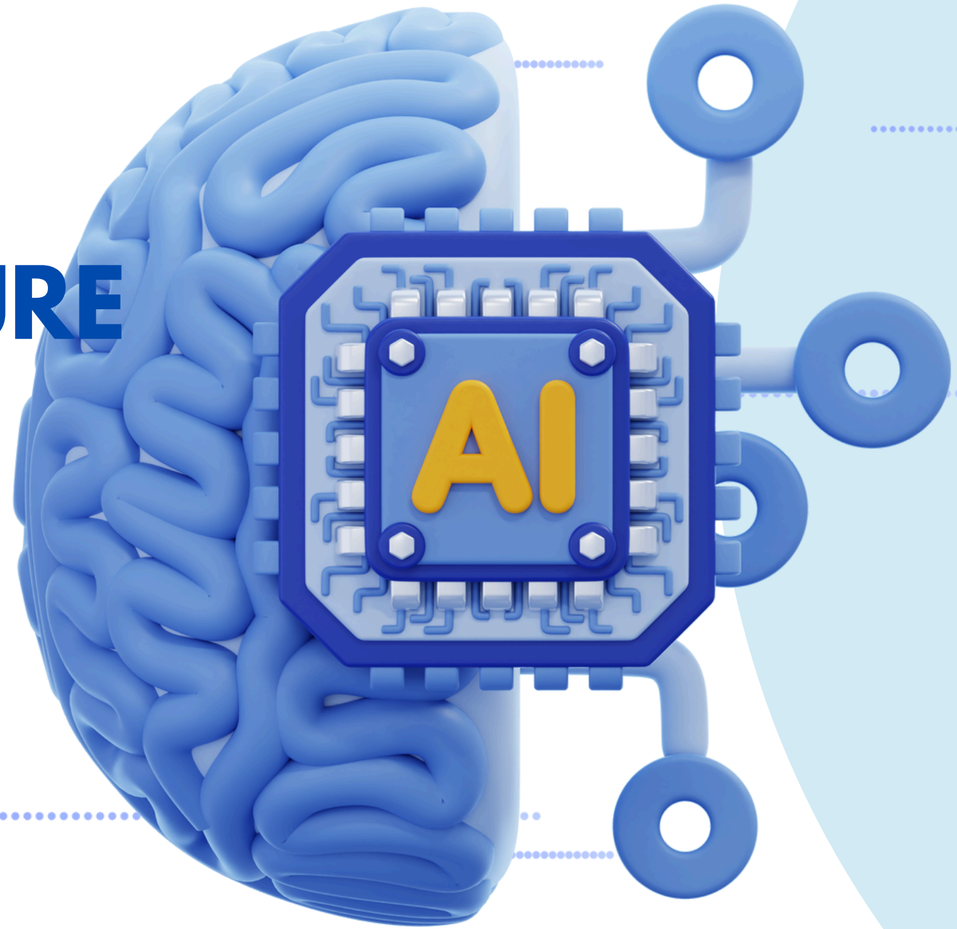
$$AttentionWeights = softmax(\frac{QK^T}{\sqrt{d_k}})$$

Step 5: Final result (weighted sum of value vectors)

- Finally, the attention weights calculated in step 4 are multiplied by the Value vector of each word and then added together. If the weight for a specific word is high, the Value vector of that word is reflected more in the final result.

$$Z = \sum(AttentionWeight \times Value)$$

03
TRANSFORMER STRUCTURE



MULTI-HEAD ATTENTION

Using only one attention (head) is like looking at a sentence from only one perspective. For example, in the sentence "The animal didn't cross the street because it was too tired", 'it' is closely related to 'animal' from the perspective of 'tired subject', but it may also be slightly related to 'street' from the perspective of 'object that did not cross'.

Multi-head attention is performing multiple attentions in parallel. That is, the W^Q, W^K, W^V weight matrices created in step 1 are created into multiple sets (e.g., 8 sets), and each generates a Query, Key, and Value from a different perspective and performs self-attention independently. The multiple result vectors (Z_0, Z_1, ..., Z_7) obtained in this way are combined again to create the final result. Through this, the model can understand the sentence in a richer and more diverse way.

POSITIONAL ENCODING

Since the transformer does not have a cyclic structure, it does not know the order of the words. "I love you" and "you love me" have the same words, but their order is different, and their meanings are completely different. Self-attention alone cannot distinguish this difference.

To solve this, the transformer adds a special vector called 'position encoding' to the embedding vector of each word. This position encoding vector is created using the sine and cosine functions, and contains unique position information for each word. This allows the model to understand the relative and absolute positions of the words and understand their order.

ENCODER AND DECODER ARCHITECTURE

Transformers are largely divided into two parts: the Encoder and the Decoder.

- Encoder: It receives an input sentence, and creates a vector representation that compresses the contextual meaning of the sentence through self-attention and other layers. (Example: Understanding the meaning of a Korean sentence)
- Decoder: Based on the meaning compressed by the encoder and previously generated words, it predicts the next word and generates an output sentence. (Example: Generating an English sentence based on the understood meaning)

The language models we use today, such as GPT (Generative Pre-trained Transformer) and Gemini, are mainly created by modifying the decoder part of this structure and stacking it in multiple layers.

SUMMARY

Summary

- RNN/LSTM had difficulty in parallelization due to its sequential processing method and had limitations in long-term dependency issues.
- Transformer solved this problem by eliminating the circular structure and introducing the 'attention' mechanism.
- Self-attention calculates the relationship between words in a sentence at once using Query, Key, and Value vectors, and identifies the context through this.
- This structure is optimized for GPU parallel operations, dramatically increasing the learning speed.
- Transformer understands sentences from various perspectives with multi-head attention and learns word order information with position encoding.
- The innovation of Transformer became the foundation of modern AI, especially large-scale language models (LLMs).

Glossary of terms:

- Transformer: A deep learning model architecture based on the attention mechanism instead of the recurrent structure.
- Attention: A mechanism that calculates weights that indicate which part to focus on while referring to the entire input sequence when predicting the output value of a specific time step.
- Self-Attention: A type of attention mechanism that is used to identify the dependency between words in a sequence to better represent the sequence itself.
- Query, Key, Value (Q, K, V): Three vectors for calculating attention weights. Query is the current word, Key is the target to compare the relationship, and Value is the actual information value.
- Multi-Head Attention: A method of learning relationships from different perspectives by performing multiple attentions in parallel.

IT Talent Training Course

Aug. 2025.

A.I. PROGRAMMING WITH PYTORCH

Instructor :
Daesung Kim

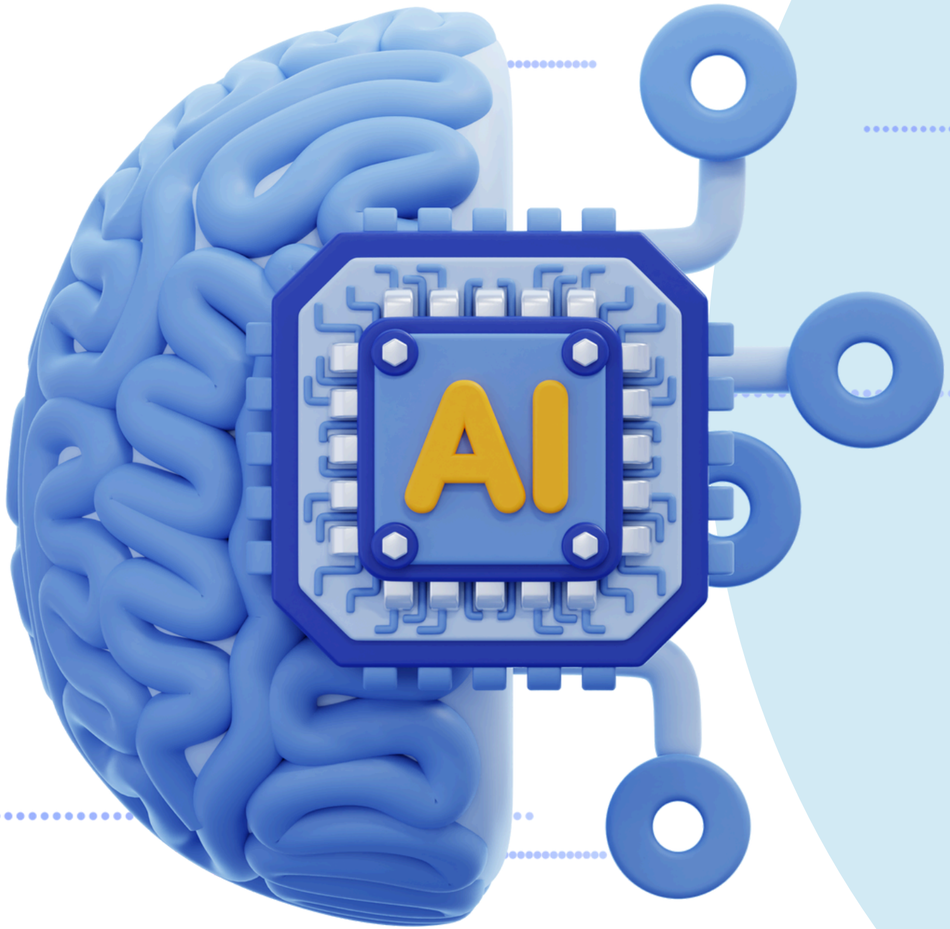
7th Day – Part 02

INDEX

COURSE OBJECTIVES

- Understand the paradigm of leveraging pre-trained large-scale language models (LLMs) in the form of APIs.
- Obtain a Gemini API key and set up your development environment in Google AI Studio.
- Generate responses to text prompts using the Gemini API.
- Implement advanced AI functions using the multimodal function that inputs text and images simultaneously.
- Expand your horizons in AI application development by experiencing the implementation of powerful AI functions with just a few lines of code.

01
API Utilization



OVERVIEW

- **API (Application Programming Interface):** A set of rules that allow other programs to use the functions of a specific program.
- **Transformer:** This is the architecture that forms the basis of modern LLM, which we learned in the morning. Gemini is also based on the transformer structure.
- **Python Basics:** Basic Python grammar knowledge such as variables, function calls, and library installation (pip) is required.

So far, we have learned how to use PyTorch to directly prepare data, design the structure of the model, and train it for a long time to solve a specific problem. This is a very important process for understanding the fundamental principles of deep learning.

However, training a model from scratch every time to solve every AI problem can be very inefficient in terms of time and cost. In particular, the latest large-scale language models (LLMs) have hundreds of billions of parameters, and training them requires hundreds or thousands of high-performance GPUs and a huge amount of data.

WHY USE AN API?

The paradigm of 'utilizing pre-trained large models as APIs' emerged to solve these problems. This is like driving a well-made car instead of building a car engine ourselves.

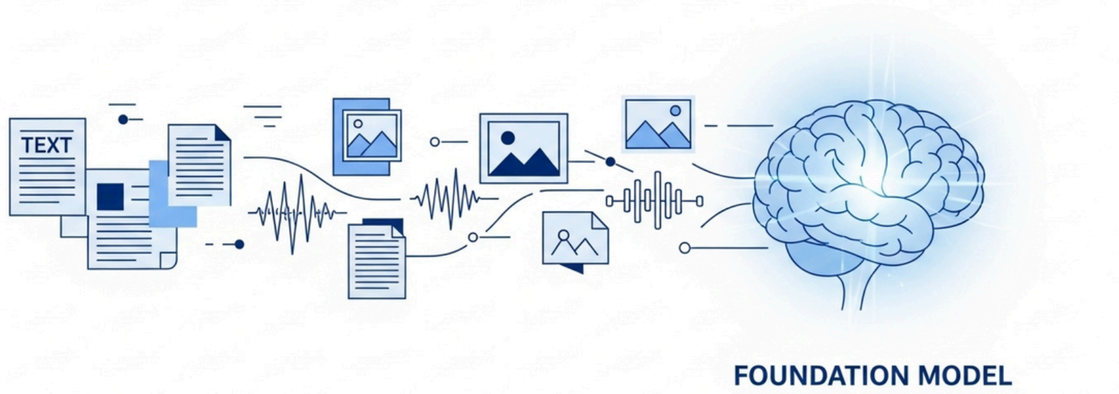
- **Efficiency:** You can save a huge amount of time and computing resources required for model training.
- **Accessibility:** You can utilize cutting-edge AI technology with just a few lines of API call code without having to understand complex deep learning theory or model structure.
- **Performance:** Since the model is trained by a large company like Google with the best data and technology, it guarantees much better performance than a model created by an individual.
- **Rapid prototyping:** When you have an idea, you can quickly create and verify a prototype by immediately linking the API without going through the model training process.

FOUNDATION MODEL AND GEMINI

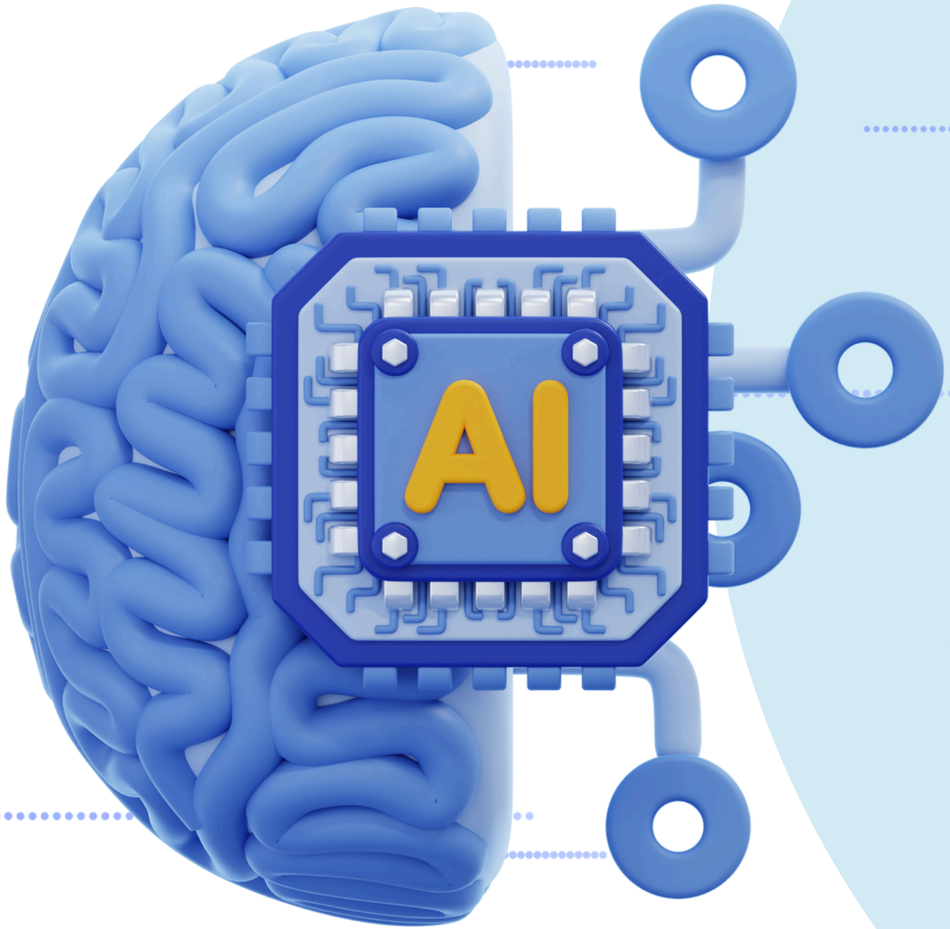
Foundation models are large-scale AI models that are pre-trained with very broad and general data, and have the potential to solve a variety of problems without being limited to specific tasks.

Gemini is the next-generation foundation model developed by Google, and its biggest feature is its multimodal ability to simultaneously understand and process various types of data such as text, images, audio, and video.

In this afternoon's hands-on lab, you will directly handle this powerful Gemini model through the API, broadening your perspective on problem solving using AI, and experiencing firsthand how easy and fast it is to create powerful applications.



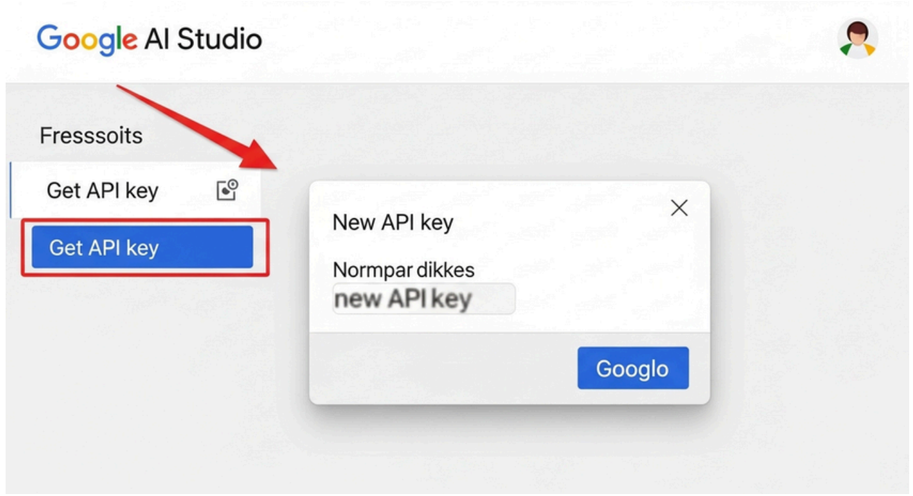
02
Preparing to use the Gemini API



ACCESS GOOGLE AI STUDIO AND ISSUE API KEY

To use the Gemini API, you need a unique 'API key' for each person. This key is a kind of password that allows our code to access Google's Gemini model.

1. Go to Google AI Studio in a web browser.
2. Log in with your Google account.
3. Click the "Get API key" button on the left menu of the screen.
4. Click the "Create API key in new project" button to create a new API key.
5. The long string of characters generated is your API key. You should keep this key safe from anyone else. Copy this key now and keep it in a notepad or similar location for a while.



INSTALLING THE PYTHON SDK (SOFTWARE DEVELOPMENT KIT)

To conveniently use the Gemini API in Google Colab notebooks, you need to install the Python library (SDK) provided by Google.

```
# 'pip' is a command to install Python libraries.
# The '-U' option means to upgrade to the latest version if it is already installed.
# 'google-generativeai' is the name of the Python SDK for the Gemini API.
!pip install -U google-generativeai
```

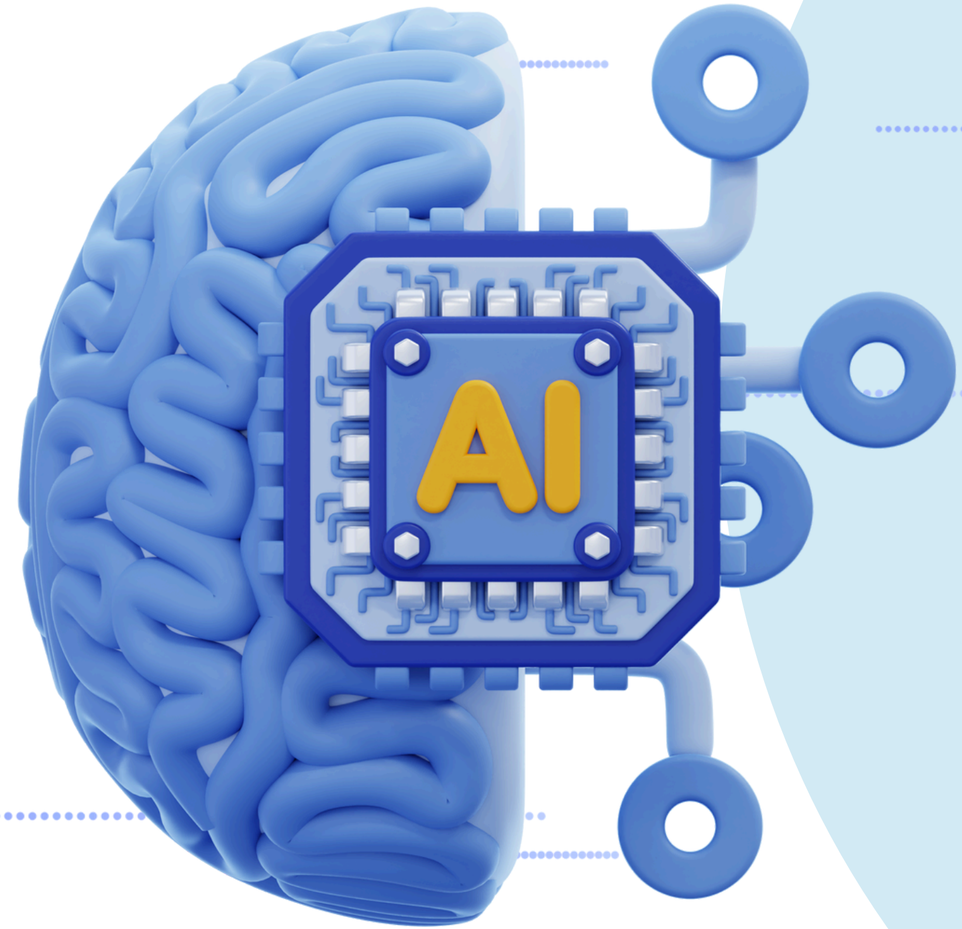
SETTING UP AN API KEY IN YOUR COLAB ENVIRONMENT

API keys are sensitive information, so it's not a good idea to hardcode them directly into your code. Google Colab provides a way to manage your API keys securely using the 'Secrets' feature.

1. Click the **key (🔑) icon (Secret Manager)** in the left menu of your Colab notebook.
2. Click + Add a new secret.
3. In the Name field, type GOOGLE_API_KEY.
4. In the Value field, paste your API key that you copied earlier from Google AI Studio.
5. Enable (check) 'Notebook Access'.
6. You can now safely retrieve and use this secret in your code.

```
import google.generativeai as genai
from google.colab import userdata

# Securely retrieve the API key stored in the Colab Secret Manager.
# The userdata.get() function does this.
api_key = userdata.get('GOOGLE_API_KEY')

# Initialize the google-generativeai library using the retrieved API key.
genai.configure(api_key=api_key)
```

03
Generate text



USE THE GEMINI-PRO MODEL, WHICH ONLY PROCESSES TEXT - 1

1. Initialize Gemini Pro model

First, we write code to specify and initialize the model we want to use.

```
# Specify the Gemini model to use. 'gemini-pro' is a model optimized for text input/output.
model = genai.GenerativeModel('gemini-pro')
```

2. Send the first prompt and confirm the response

Now let's ask the model its first question (prompt).

```
# Define the question to be passed to the model, i.e. the prompt.
prompt = "Tell me 3 positive impacts that artificial intelligence (AI) will have on society in the future."

# Call the API by passing the prompt to the model.generate_content() function.
# When this function is executed, our code communicates with the Gemini model on Google's servers.
response = model.generate_content(prompt)

# Extract only the text portion from the response object and print it.
print(response.text)
```

USE THE GEMINI-PRO MODEL, WHICH ONLY PROCESSES TEXT - 2

3. Prompt Engineering Fundamentals: How to Elicit Better Answers

Rather than simply asking a model a question, you can get much more satisfying answers by giving it a persona or specifying the format of the desired output. This is called prompt engineering.

Example 1: Assigning roles

```
prompt = """
You are a professional Laos travel guide.
For a 20-something backpacker visiting Laos for the first time, please recommend a 3-night, 4-day Luang Prabang travel itinerary.
Please include the places to visit and a brief description for each day.
"""


response = model.generate_content(prompt)
print(response.text)
```

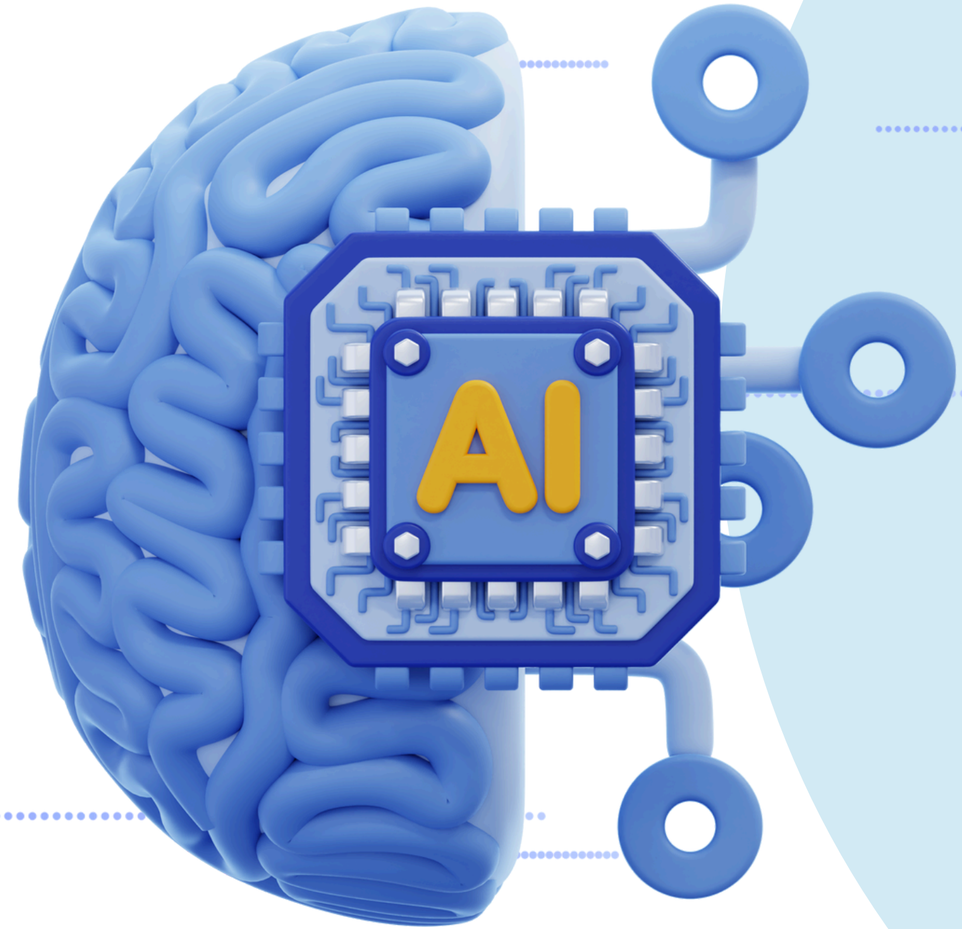
USE THE GEMINI-PRO MODEL, WHICH ONLY PROCESSES TEXT - 3

Example 2: Specifying the output format

```
prompt = """
Explain the difference between PyTorch Tensor and NumPy ndarray.
Organize your answer in the following format as a markdown table.

| Classification | PyTorch Tensor | NumPy ndarray |
|---|---|---|
| Main purpose | | |
| GPU usage | | |
| Automatic differentiation | | |
"""


response = model.generate_content(prompt)

# It looks better if you use the IPython library for markdown rendering.
from IPython.display import Markdown
Markdown(response.text)
```

As you can see, the quality of AI's answers can vary greatly depending on how you design the prompts.

04
Utilizing multimodal functions



MULTIMODAL PRACTICE - 1

1. What is multimodal?

Multimodal means 'multiple modes (forms)'. In AI, multimodality refers to the ability to understand and process two or more different forms of data at the same time, such as text, images, and voice. Gemini is a model that is very powerful in this multimodal function.

In this exercise, we will use the gemini-pro-vision model that uses text and images as input.

2. Prepare and upload images

First, you need to upload the image you want to analyze to your Colab environment. Download a picture of Laotian food (e.g. Lap, Khao Piak) from the internet and upload it by dragging and dropping it into the file explorer area of your Colab notebook.

MULTIMODAL PRACTICE- 2

3. Sending a prompt with an image

Now let's send the image and text together to the model.

```
import PIL.Image # Use PIL (Pillow) library for image processing.

# Change the model to be used to 'gemini-pro-vision'.
vision_model = genai.GenerativeModel('gemini-pro-vision')

# Open the image file uploaded to Colab. The file name should be changed to the actual uploaded file name.
# Example: 'laap.jpg'
img = PIL.Image.open('image_file_name_uploaded_here.jpg')

# Pass text and images together in a list format to the prompt.
prompt_with_image = [
    "What is the name of the food in this image?",
    "And tell me the 3 main ingredients of this food.",
    img
]

response = vision_model.generate_content(prompt_with_image)
print(response.text)
```

MULTIMODAL PRACTICE - 3

When you run the code, you can see that the Gemini model 'sees' the image and accurately describes the name and ingredients of the food. This is a much more advanced image understanding function with much simpler code than the CNN model we built on Day 5.

This experience greatly expands our thinking about how to use AI. You don't always have to start from scratch to solve complex problems. It can be much smarter and faster to stand on the shoulders of well-built giants, that is, to leverage the powerful Foundation Model API.

SUMMARY

Summary

- Paradigm Shift: We understood that we are moving from an era where we train all models ourselves to an era where we utilize well-made foundation models as APIs.
- Gemini API Key Issue: We learned how to issue a personal API key through Google AI Studio and set it up securely through Colab Secret Manager.
- Text Generation: We practiced generating desired text with simple prompts using the gemini-pro model and the generate_content() function.
- Prompt Engineering: We confirmed that specifying the role and output format to the model specifically can lead to better quality answers.
- Multimodal AI: We experienced powerful multimodal capabilities using the gemini-pro-vision model to simultaneously input text and images and receive answers to questions about the images.

Glossary of terms:

- API (Application Programming Interface): An application programming interface. A collection of predefined functions and rules that allow external use of a specific function (e.g., Gemini model).
- SDK (Software Development Kit): A software development kit. A collection of related tools and libraries that allow for easy use of a specific service (e.g., Gemini API) in a specific programming language (e.g., Python).
- Foundation Model: A large-scale AI model that is pre-trained with a large amount of data and serves as a foundation for performing various types of downstream tasks.
- Multimodal: The ability to process two or more different types of data together, such as text, images, and voice.
- Prompt Engineering: A technique or process for optimizing and designing inputs (prompts) to obtain the best desired results from an AI model.