Clinical Note Classification

Fine-tuning Large Language Models (LLMs) with LoRA

Description:

This project aims to fine-tuned Large Language Models **(LLMs)** to classify clinical notes based on patient information documented in unstructured medical records. Clinical notes, written by healthcare professionals, often contain valuable insights regarding patient symptoms, history, and treatment plans. By fine-tuning a pre-trained LLM using domain-specific medical data , the model will learn to understand medical notes , detect key clinical patterns, and accurately assign medical categories. that Contribute in decision support and enhancing the efficiency of healthcare professionals

Dataset kaggle

Medical Transcriptions

Context

Medical data is extremely hard to find due to HIPAA privacy regulations. This dataset offers a solution by providing medical transcription samples.

Data Columns

description medical specialty sample name transcription keywords

Dataset kaggle

Medical Transcriptions

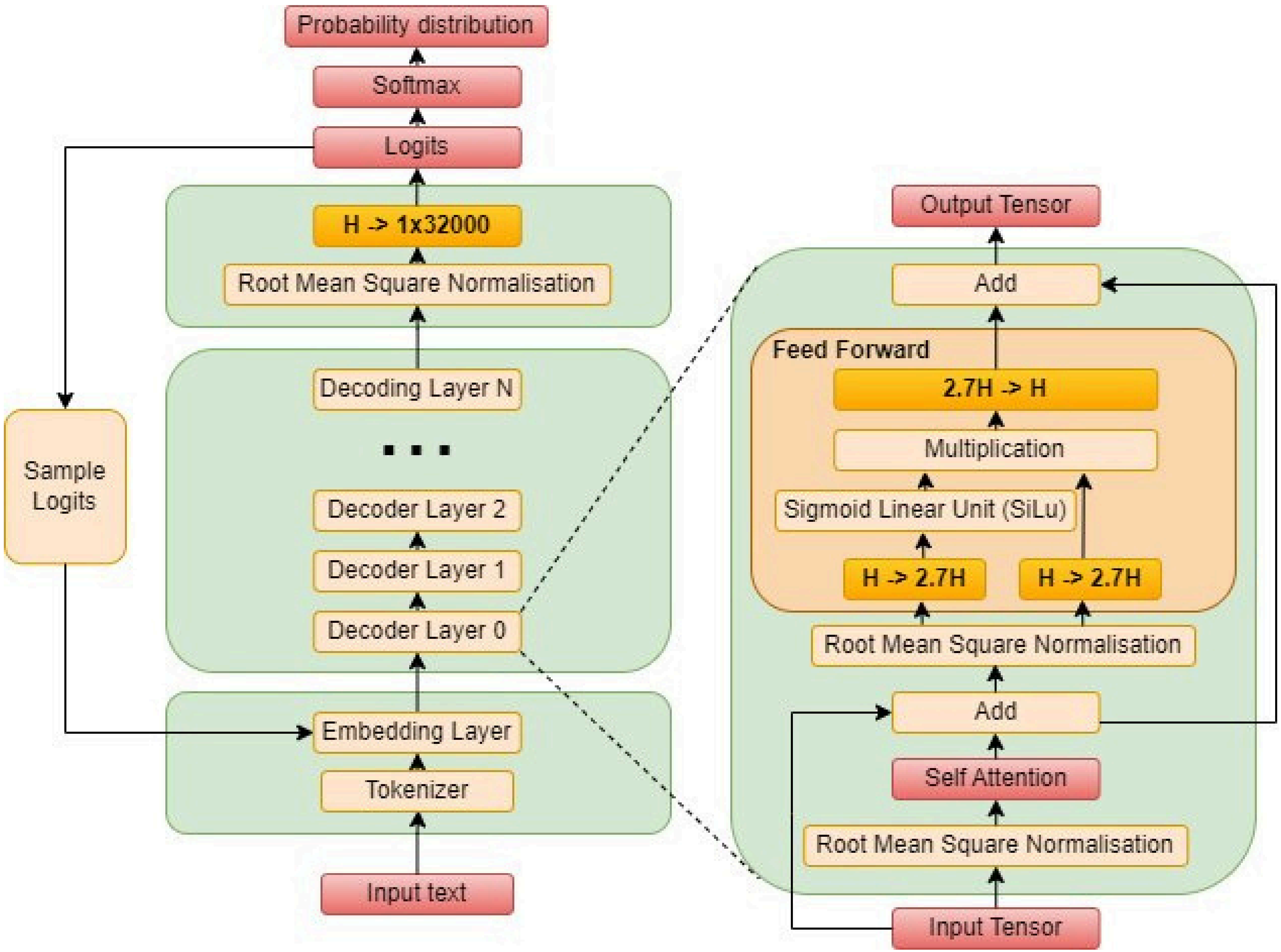
description: A short summary or overview of the medical case
medical specialty: The area of medicine related to the transcription.
Sample name: unique or descriptive title for the transcription sample.
transcription: The full text of the transcribed medical record or note.
Keywords : Important terms or phrases extracted from the transcription

Used Features : **transcription (text) , medical_specialty (label)**
Data size : **4999 row**

Model (LLaMa)

TinyLLaMA/TinyLLaMA-1.1B-Chat-v1.0

the same architecture and tokenizer as Llama 2. This means TinyLlama can be plugged and played in many open-source projects built upon Llama. Besides, TinyLlama is compact with only 1.1B parameters. This compactness allows it to cater to a multitude of applications demanding a restricted computation and memory footprint


Probability distribution
Softmax
Logits
H -> 1x32000
Root Mean Square Normalisation
Decoding Layer N
. . .
Decoder Layer 2
Decoder Layer 1
Decoder Layer 0
Embedding Layer
Tokenizer
Input text
Sample Logits
Output Tensor
Add
Feed Forward
2.7H -> H
Multiplication
Sigmoid Linear Unit (SiLu)
H -> 2.7H
H -> 2.7H
Root Mean Square Normalisation
Add
Self Attention
Root Mean Square Normalisation
Input Tensor

LLaMA Decoder Block – Main Components

1. Self-Attention Layer
   - Purpose: Enables the model to focus on relevant tokens across the sequence.
   - Core Components:
     - Q, K, V projections: Linear layers for Query, Key, and Value vectors.
     - RoPE: Rotary positional embeddings for position information.
     - Output projection: Projects attention output back to hidden size H.

2. MLP (Feed-Forward Network)
   - Purpose: Transforms hidden states non-linearly to improve expressiveness.
   - Structure:
     - Up-projection: H → 2.7H
     - Activation: SiLU (smooth, non-linear)
     - Down-projection: 2.7H → H
   - Residual connection: MLP output is
   - added to the input of the MLP.

3. RMSNorm (Root Mean Square Normalization)
   - Purpose: Normalizes hidden states to stabilize
   - training (alternative to LayerNorm).

4. Residual Connections
   - Add the input of each block (Attention, MLP) to its output → helps gradient flow.

Base Model information

Component	Description
Model Type	LLaMA-based causal transformer
Params	~1.1 billion
Task	Sequence Classification
PEFT Type	LoRA (Low-Rank Adaptation)
Trainable Components	q_proj, v_proj (LoRA) + classifier head
Layers	22 Transformer Decoder Layers
Hidden Dim	2048
Token Embedding Size	32,000
Positional Encoding	Rotary Embeddings
Activation	SiLU
Norm	RMSNorm
Classification Output	Linear(2048 → num_labels)

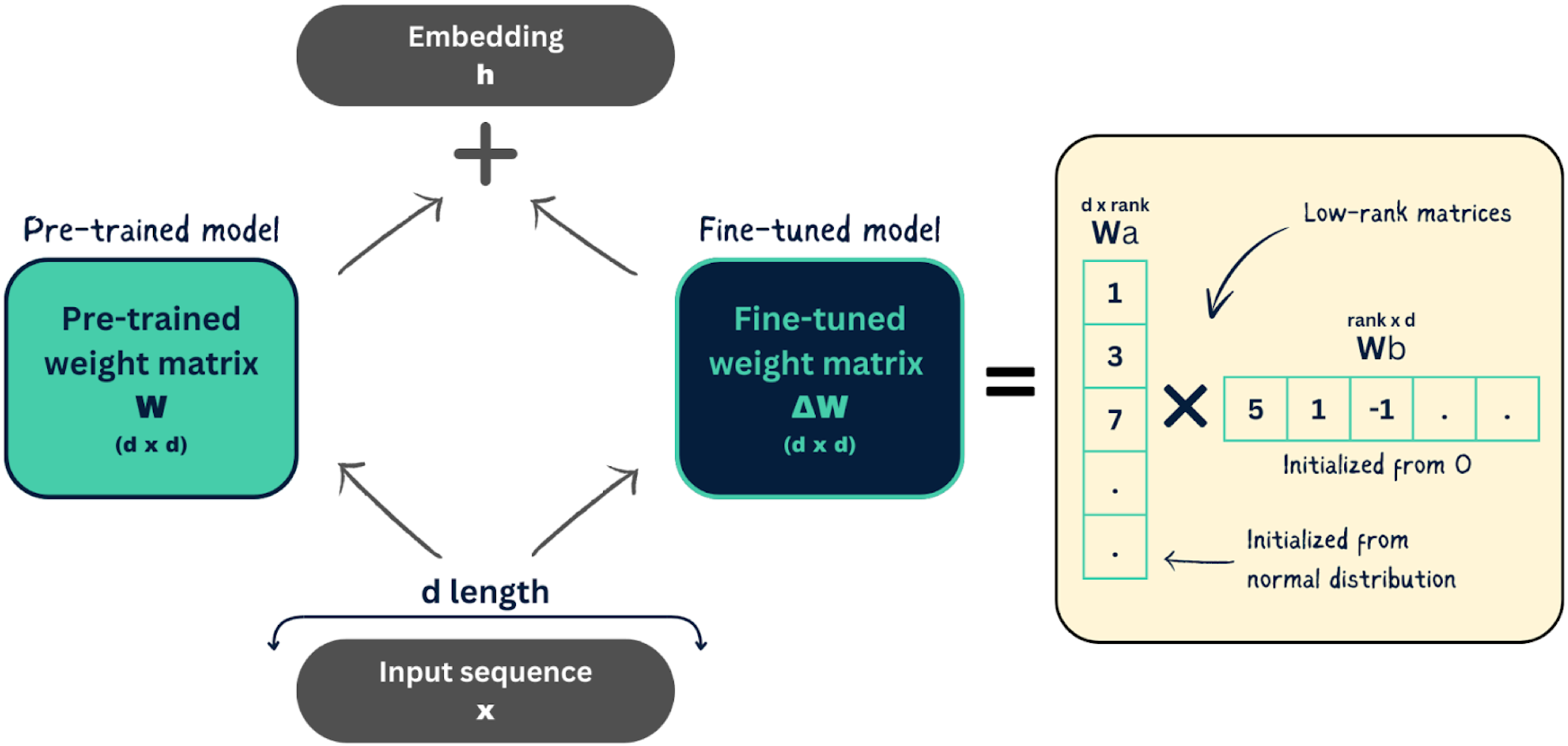
LoRA (Low-Rank Adaptation)

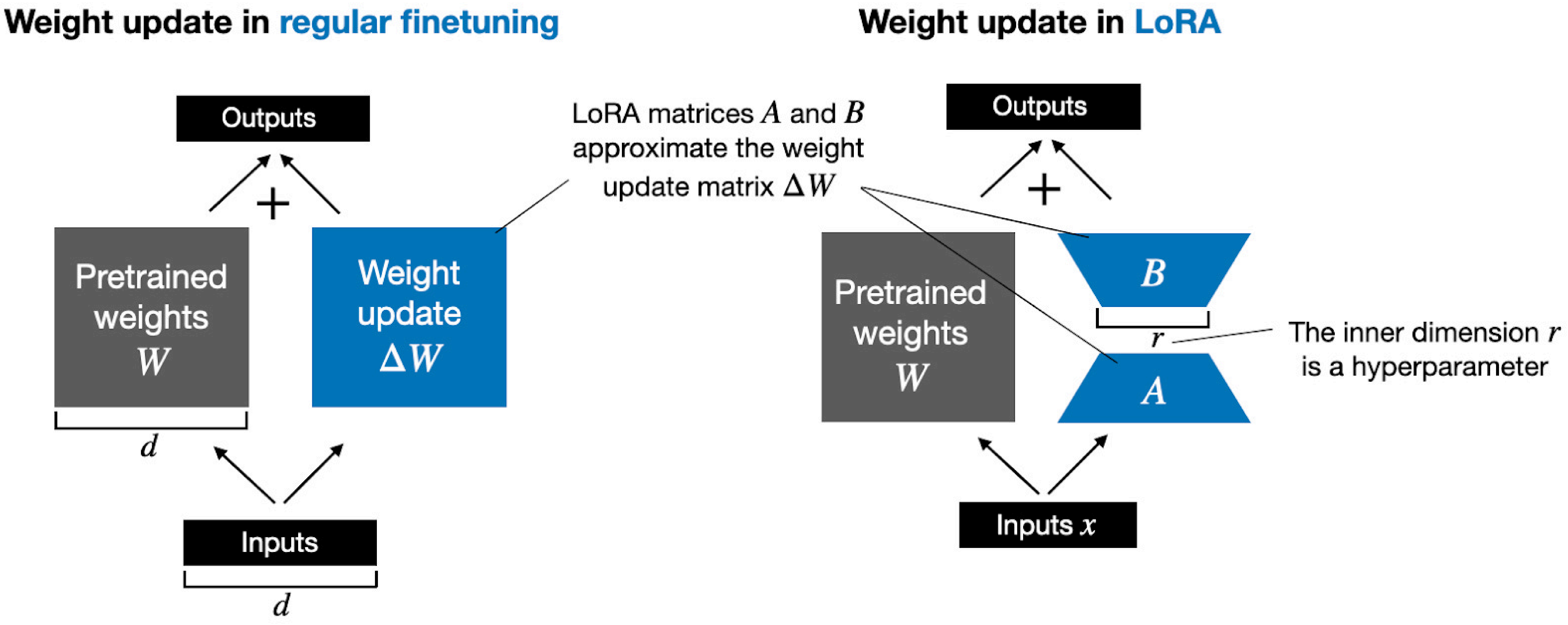
LoRA is a method that lets you fine-tune only a small part of a large model by adding a few extra trainable parameters, instead of updating all of the model's weights.

It's designed for:

- Efficiency (less memory and compute)
- Scalability (works on very large models like LLaMA or GPT)
- Modularity (easy to plug into existing models)

Low Rank Adaptation (LoRA) Overview





Weight update in regular finetuning
Weight update in LoRA
Outputs
Pretrained weights W
Weight update ΔW
d
Inputs
d
LoRA matrices A and B approximate the weight update matrix ΔW
Outputs
Pretrained weights W
B
r
A
The inner dimension r is a hyperparameter
Inputs x

As illustrated above, the decomposition of ΔW means that we represent the large matrix ΔW with two smaller LoRA matrices, A and B. If A has the same number of rows as ΔW and B has the same number of columns as ΔW, we can write the decomposition as ΔW = AB. (AB is the matrix multiplication result between matrices A and B.)

How much memory does this save? It depends on the rank r, which is a hyperparameter. For example, if ΔW has 10,000 rows and 20,000 columns, it stores **200,000,000 parameters**. If we choose A and B with r=8, then A has 10,000 rows and 8 columns, and B has 8 rows and 20,000 columns, that's 10,000×8 + 8×20,000 = 240,000 parameters, which is about 830× less than 200,000,000.

the computation complexity of multiplying A and B is similar to that of multiplying ΔW directly, as you're still computing a 10,000×20,000 matrix in the end.

Note: The saving happens in terms of memory usage, not in computational cost for multiplication

20,000 Column

$$\underbrace{\begin{bmatrix} 1 \\ 3 \\ 7 \\ -4 \\ 2 \end{bmatrix}}_{\text{1 Column}} \times \underbrace{\begin{bmatrix} 5 & 1 & -1 & 3 & 4 \end{bmatrix}}_{\text{1 Row}} = \underbrace{\begin{bmatrix} 5 & 1 & -1 & 3 & 4 \\ 15 & 3 & -3 & 9 & 12 \\ 35 & 7 & -7 & 21 & 28 \\ -20 & -4 & 4 & -12 & -16 \\ 10 & 2 & -2 & 6 & 8 \end{bmatrix}}_{10{,}000 \times 20{,}000}$$

10,000 Row

1 Row

1 Column

10,000x20,000

The saving happens in terms of memory usage, not in computational cost for multiplication

Key Parameters:

r (rank): — determines the size of the low-rank matrices, balancing efficiency and model capacity.

lora alpha: — scales the low-rank update to control its impact on the model.

target modules: Focus on (query and value projections) to optimize key parts of the transformer.

lora_dropout: — applies dropout to the low-rank matrices to prevent overfitting.

bias: — no bias term is used in the low-rank approximation.

Evaluation Methods

Accuracy Over 5 Classes:

Before Fine-tuning	After Fine-tuning
Base Model	**Base Model with (LoRA) Adapter**
15.6489%	**73.5632%**

+ loRA

Adapter

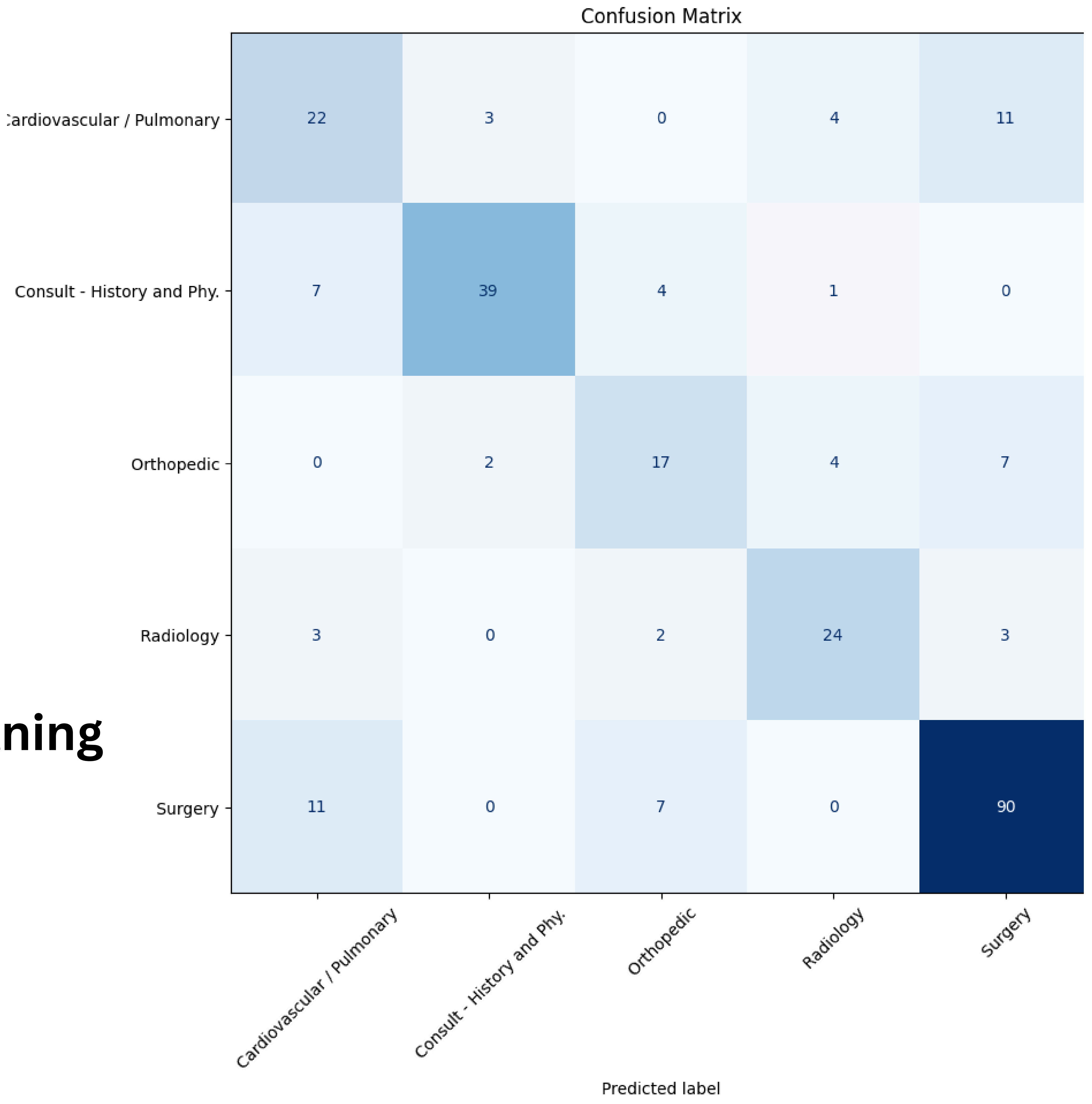
Confusion Matrix

Note:

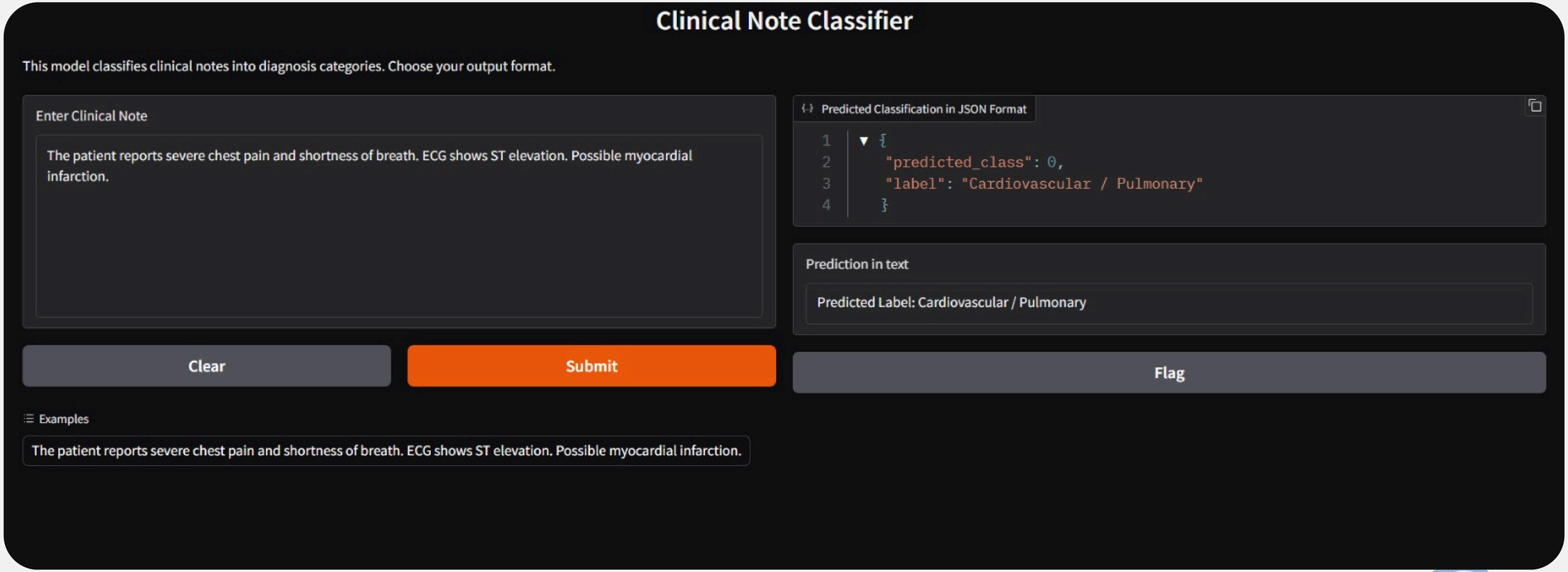
Before Fine-tuning Model randomly predict output

so Confustion Matrix Doesn’t Add Meaning

After Fine-tuning



GUI


Clinical Note Classifier
This model classifies clinical notes into diagnosis categories. Choose your output format.
Enter Clinical Note
The patient reports severe chest pain and shortness of breath. ECG shows ST elevation. Possible myocardial infarction.
Predicted Classification in JSON Format
{
"predicted_class": 0,
"label": "Cardiovascular / Pulmonary"
}
Prediction in text
Predicted Label: Cardiovascular / Pulmonary
Clear
Submit
Flag
Examples
The patient reports severe chest pain and shortness of breath. ECG shows ST elevation. Possible myocardial infarction.

Thank you very much!