

# INTRODUCTION TO FINE-TUNING

Yogesh Haribhau Kulkarni



# Outline

## ① INTRODUCTION TO FINE-TUNING

## ② IMPLEMENTATIONS

## ③ REFERENCES

# Fine-tuning LLMs (Large Language Models)

# Introduction to Fine-Tuning

- ▶ LLMs like GPT have vast language knowledge but lack specialization.
- ▶ Fine-tuning allows learning from domain-specific data for accuracy.
- ▶ **Definition:** Process of training pre-existing models on smaller, domain-specific datasets to enhance task or domain performance.
- ▶ **Example:** Healthcare organization fine-tunes on patient reports, adapting to medical terminologies.
- ▶ **Note:** Base Models are typically raw ie generic, may not give good results, but instruct-models are already tuned a bit, so they give better results. Fine-tuning either of them is ok, IMO. Instruct models come with their own prompt templates (read documentation).



# See the Difference

## Base Model

(davinci (GPT-3))

**Prompt:** Tell me how to fine-tune a model

**Completion:**

How can I control the complexity of a model?  
How do I know when my model is done?  
How do I test a model?  
How do I define a task for my model?  
How do I scale a model?  
How do I make a probabilistic model?  
How do I start using a model once it's complete?  
How do I cope with missing data?  
How do I make my modeling efforts sustainable?

## Fine-tuned Model

(text-davinci-003 )

**Prompt:** Tell me how to fine-tune a model

**Completion:**

Fine-tuning a model involves adjusting the parameters of a pre-trained model in order to make it better suited for a given task. There are generally three steps to fine-tuning a model:

1. Select a base model - Select a pre-trained deep learning model that has been trained on a large dataset.
2. Adjust parameters - Adjust parameters of the pre-trained model to better suit the desired task. This may include changing the number of layers, adjusting learning rate, adding regularization, or tweaking the optimizer.
3. Train the model - Train the new model on the desired dataset. The amount of data and the amount of training required will depend on the task and the model.

(Ref: Fine-tuning Large Language Models (LLMs) — w/ Example Code - Shaw Talebi)

## Why? Customization

- ▶ Each domain/task has unique language patterns, terminologies, nuances.
- ▶ Fine-tuning LLMs customizes them to understand domain-specific aspects.
- ▶ Customization aligns model's responses with specific requirements.
- ▶ Ensures accurate and contextually relevant outputs.
- ▶ LLMs excel in legal, medical, business, and other domains with specialized datasets.
- ▶ Fine-tuning empowers leveraging LLMs' power while maintaining accuracy.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)

## Why? Data compliance

- ▶ Industries like healthcare, finance, law have strict data regulations.
- ▶ Fine-tuning LLMs on proprietary/regulated data ensures compliance.
- ▶ Develops LLMs trained specifically on in-house or industry data.
- ▶ Mitigates risk of exposing sensitive information to external models.
- ▶ Enhances security and privacy of data.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)



## Why? Limited labeled data

- ▶ Obtaining labeled data for specific tasks/domains can be challenging.
- ▶ Fine-tuning adapts pre-trained LLM to available labeled dataset.
- ▶ Maximizes utility and performance of pre-existing labeled data.
- ▶ Overcomes constraints of data scarcity.
- ▶ Achieves significant improvements in model's accuracy and relevance.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)

## How? Last layer, Feature extraction (repurposing)

- ▶ Feature extraction (repurposing) is a primary approach for fine-tuning LLMs.
- ▶ Pre-trained LLM treated as fixed feature extractor.
- ▶ Model learns significant language features from vast dataset.
- ▶ Final layers trained on task-specific data while rest of model remains frozen.
- ▶ Leverages rich representations learned by LLM, adapting them to specific task.
- ▶ Offers cost-effective and efficient fine-tuning.

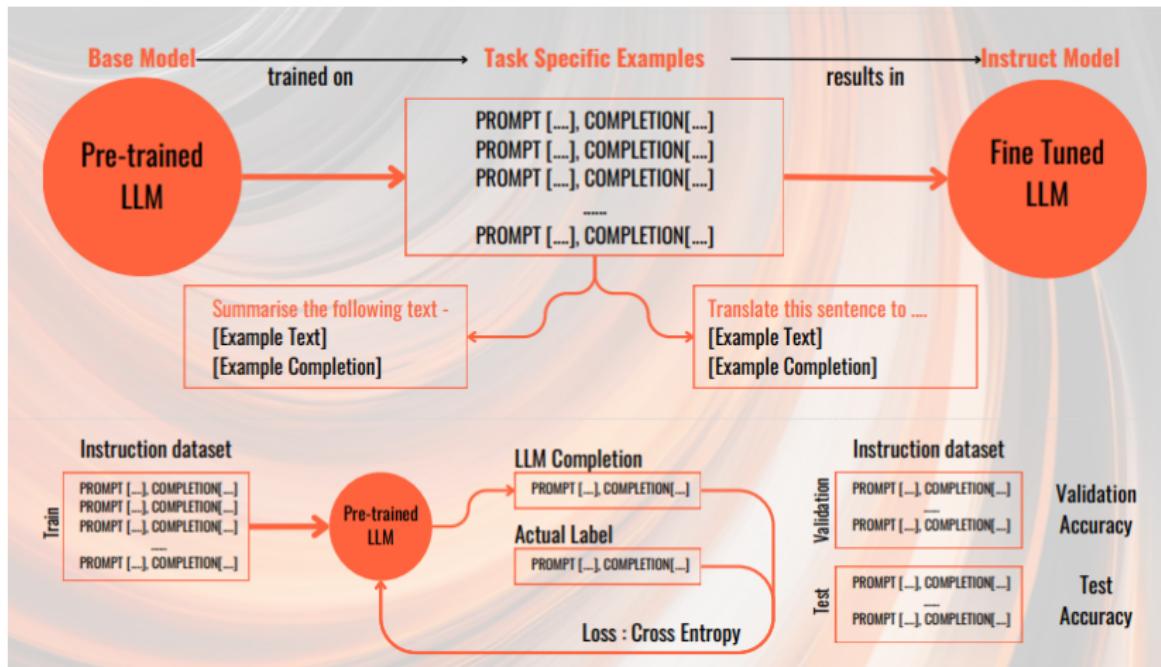
(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)

## How? Full fine-tuning

- ▶ Full fine-tuning is another primary approach for LLM fine-tuning.
- ▶ Differs from feature extraction by training entire model on task-specific data.
- ▶ All model layers adjusted during training.
- ▶ Particularly beneficial for large, significantly different task-specific datasets.
- ▶ Allows deeper adaptation of model to specific task.
- ▶ May lead to superior performance but requires more computational resources and time.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)

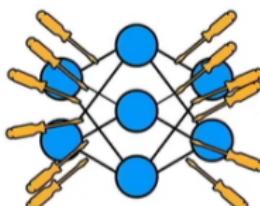
# What is a fine-tuned LLM?



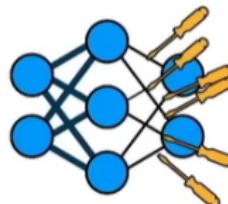
(Ref: Generative AI with Large Language Model - Abhinav Kimothi)

# Ways to change parameters/weights

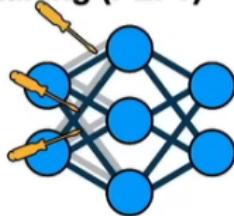
## 1) Retrain all parameters



## 2) Transfer Learning



## 3) Parameter Efficient Fine-tuning (PEFT)



[6]

(Ref: Fine-tuning Large Language Models (LLMs) — w/ Example Code - Shaw Talebi)

YHK

# Fine-Tuning Methods

## ► **Supervised Full Fine-Tuning:**

- ▶ Involves updating all parameters of the language model during training.
- ▶ Resource-intensive but ensures thorough adaptation of the entire model to the task or domain.

## ► **Instruction Fine-Tuning:**

- ▶ Involves training the model using examples with explicit instructions for specific queries or tasks.
- ▶ Suitable for applications where precise task execution is essential.

## ► **Reinforcement Learning from Human Feedback (RLHF):**

- ▶ Incorporates reinforcement learning principles with human evaluators providing ratings.
- ▶ Ratings serve as rewards, guiding the model to optimize parameters based on human preferences.

## ► **Parameter-Efficient Fine-Tuning (PEFT):**

- ▶ Aims to reduce computational expenses by selectively updating a small set of parameters.
- ▶ Example: Low-rank adaptation (LoRA) technique focuses on updating only relevant parameters.

# Parameter Efficient Fine Tuning (PEFT)

YHK

## What PEFT addresses?

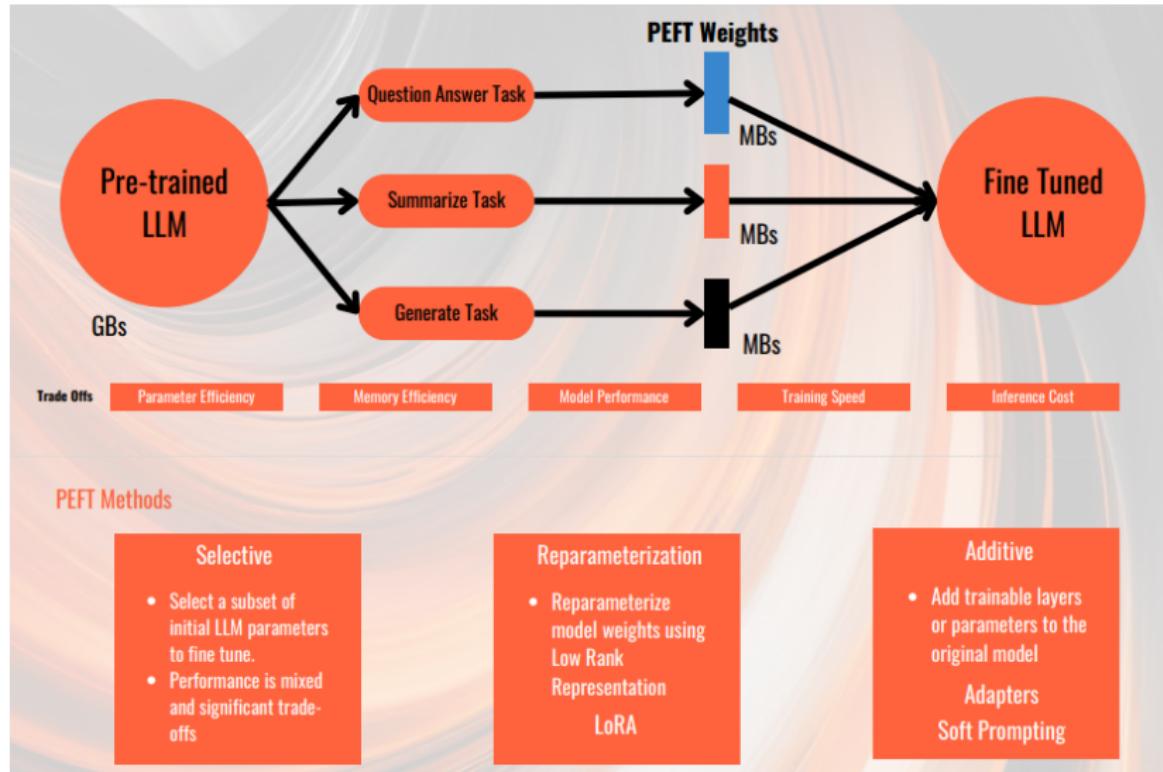
- ▶ **Full Fine Tuning:**
  - ▶ Requires memory for the entire model, optimizers, gradients, etc.
  - ▶ Similar memory demands as pre-training.
- ▶ **Parameter-Efficient Fine Tuning (PEFT):**
  - ▶ Fine-tunes only a subset of model parameters.
  - ▶ In some cases, leaves the original weights untouched.
- ▶ **Addressing Resource Intensity:**
  - ▶ PEFT addresses the resource-intensive nature of fine-tuning Large Language Models (LLMs).
  - ▶ Full fine-tuning modifies all parameters, while PEFT fine-tunes only a small subset, minimizing computational demands.

PEFT is a library from Hugging Face which comes with several options to train models efficiently, one of them is LoRA.

(Ref: Generative AI with Large Language Model - Abhinav Kimothi)



# What is Parameter Efficient Fine Tuning?



# Low Rank Adaptation (LoRA)

# Introduction

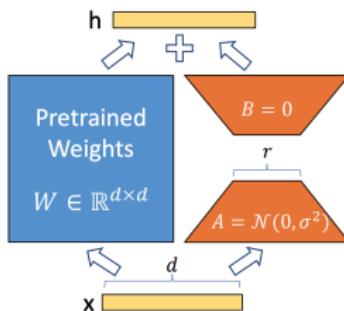
- ▶ Improved fine-tuning method.
- ▶ Fine-tunes two smaller matrices approximating larger weight matrix.
- ▶ These matrices form LoRA adapter.
- ▶ Fine-tuned adapter loaded into pre-trained model for inference.
- ▶ Outcome: unchanged original LLM and smaller “LoRA adapter.”
- ▶ Adapter often represents single-digit percentage of original LLM size (in MBs).
- ▶ During inference, LoRA adapter combined with original LLM.
- ▶ Advantage: ability to reuse original LLM, reducing overall memory requirements for multiple tasks/use cases.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)



## Why LoRA?

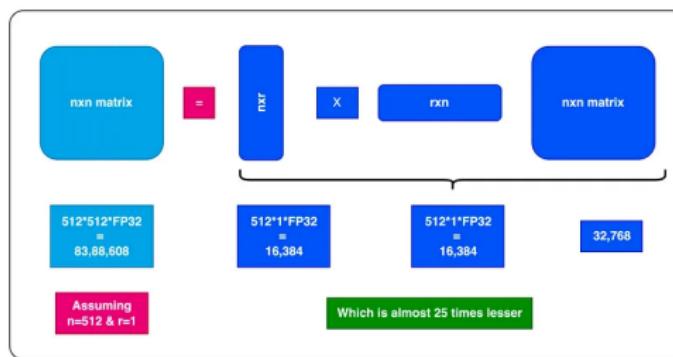
- ▶ Fine-tuning large language models (LLMs) is computationally expensive
- ▶ Updating all parameters of LLMs requires significant memory and compute resources
- ▶ LoRA aims to reduce the computational cost of fine-tuning while maintaining performance
- ▶ LoRA is one of the PEFT techniques, others being 'Prefix tuning', 'P tuning' and 'Prompt Tuning'.
- ▶ LoRA allows to update only a small subset of 'extra' weights while keeping original model 'frozen'



(Ref: <https://heidloff.net/article/efficient-fine-tuning-lora/>)

## LoRA: Low-Rank Adaptation

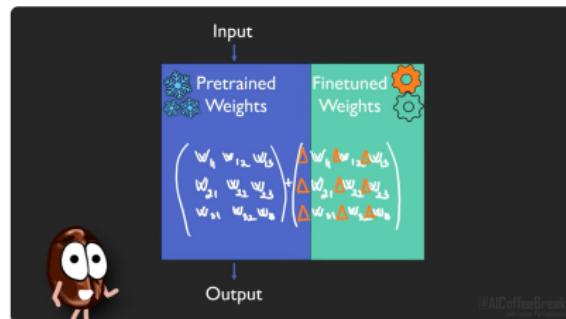
- ▶ Introduces trainable rank decompositions of weight updates
- ▶ Instead of updating the full weight matrices, LoRA adds a small number of rank-decomposed weight matrices
- ▶ Significantly reduces the number of trainable parameters during fine-tuning
- ▶ As LoRA keeps original weights as is, it helps stopping catastrophic forgetting (new training, washing old learning)



(Ref: <https://abvijaykumar.medium.com/fine-tuning-lm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4>)

## LoRA: Low-Rank Adaptation

- ▶ Pretrained original model weights are frozen
- ▶ Another set of weights are fine-tuned, meaning a difference-like-delta-weights are created such that when these deltas are added to original weights, they act as if the combined model is tuned for the new corpus.
- ▶ So, have we doubled the parameters? original and fine-tuned sets? But that's on hard-disk. When we load it in GPU for inference, addition is done first and then loaded.



(Ref: What is LoRA? Low-Rank Adaptation for finetuning LLMs EXPLAINED-AI Coffee Break with Letitia)

## LoRA: Low-Rank Adaptation

- ▶ LoRA does a trick by which the number of fine-tuned weights are reduced.
- ▶ Leverages Rank of the matrix. In matrix, if some rows/columns are linearly dependent on others, they are redundant, as they can be generated by others. After such removal, the matrix dimension is the rank.
- ▶ So, just tune the weights of low rank, column and row vector.
- ▶ rank 'r' is the hyper parameter, we can chose.
- ▶ matrix 'A' is initialized from Gaussian distribution and 'B' with 0 and let the back propagation, settle the weights.

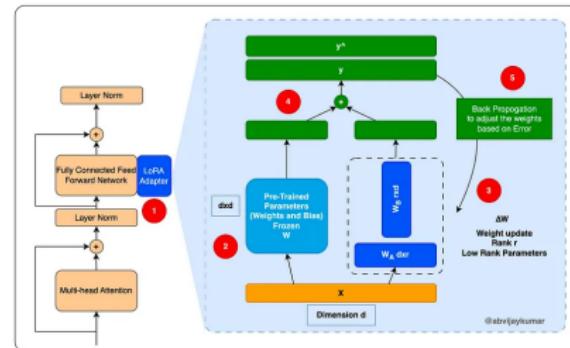
The diagram shows a green box labeled "Finetuned Weights" containing three gears. Below it, the text "LoRA" is written. To the right, a mathematical equation is shown:  $\begin{pmatrix} \Delta w_1, \Delta w_2, \Delta w_3 \\ \Delta w_2, \Delta w_2, \Delta w_3 \\ \Delta w_3, \Delta w_2, \Delta w_3 \end{pmatrix}_{d \times k} = \Delta W = BA = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}_{(3 \times 1)} \cdot \begin{pmatrix} a_1, a_2, a_3 \end{pmatrix}_{(1 \times 3)}$ . A note at the bottom left says "assume d=3, k=3". A cartoon character with a thinking bubble is at the bottom left, and the text "AI Coffee Break" is at the bottom right.

(Ref: What is LoRA? Low-Rank Adaptation for finetuning LLMs EXPLAINED-AI Coffee Break with Letitia)

YHK

## LoRA: Low-Rank Adaptation

- ▶ Original pre-trained parameters ( $W$ ) are frozen
- ▶ New low-rank weight vectors  $W_A$  and  $W_B$  are added
- ▶ Dimensions:  $W_A (d \times r)$ ,  $W_B (r \times d)$ , where  $r < d$
- ▶  $r$  (rank) is a crucial parameter, lower  $r$  means faster training but may impact performance
- ▶ Result computed as dot product of original and low-rank networks
- ▶ Loss function calculated, and  $W_A$  and  $W_B$  adjusted during backpropagation



(Ref: <https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4>)

## LoRA Weight Update

Let  $W \in \mathbb{R}^{m \times n}$  be a weight matrix in the pre-trained LLM.

- ▶ LoRA decomposes the weight update as  $\Delta W = BA^T$
- ▶  $A \in \mathbb{R}^{r \times n}$  and  $B \in \mathbb{R}^{m \times r}$  are trainable rank- $r$  matrices
- ▶  $r \ll \min(m, n)$ , so the number of trainable parameters is greatly reduced



## LoRA Training

During fine-tuning, the weight matrix  $W$  is updated as:

$$W' = W + \Delta W = W + BA^T$$

- ▶ Only the low-rank matrices  $A$  and  $B$  are trained
- ▶ Pre-trained weights  $W$  remain frozen
- ▶ Efficient memory-wise and compute-wise

## LoRA Inference

During inference, the updated weight matrix  $W'$  is used:

$$W' = W + BA^T$$

- ▶ No additional compute or memory required
- ▶ LoRA weight update is applied on-the-fly

## LoRA: code

```
1 from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
2 from peft import LoraConfig, get_peft_model, prepare_model_for_int8_training,
3     TaskType
4
5 model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-xxl",
6     load_in_8bit=True, device_map="auto")
7 lora_config = LoraConfig(
8     r=16,
9     lora_alpha=32,
10    target_modules=["q", "v"],
11    lora_dropout=0.05,
12    bias="none",
13    task_type=TaskType.SEQ_2_SEQ_LM
14)
15
16 model = prepare_model_for_int8_training(model)
17 model = get_peft_model(model, lora_config)
18
19 model.print_trainable_parameters()
20 # trainable params: 18874368 || all params: 11154206720 || trainable%:
21 # 0.16921300163961817
```

## What is QLoRA?

- ▶ Memory-efficient iteration of LoRA.
- ▶ Quantizes weights of LoRA adapters to lower precision (e.g., 4-bit instead of 8-bit).
- ▶ Further reduces memory footprint and storage requirements.
- ▶ Pre-trained model loaded into GPU memory with quantized 4-bit weights.
- ▶ Despite reduction in bit precision, maintains comparable effectiveness to LoRA.

(ref: Fine-Tuning LLMs : Overview, Methods, and Best Practices - Turing)



# Conclusions

## Why Fine-Tune Large Language Models?

- ▶ Adapt to specific tasks, domains, and nuances for enhanced performance.  
Fine-tuning for document analysis in the legal domain.
- ▶ Align models with new data distributions and out-of-distribution examples.  
Example: Fine-tuning a speech recognition model for a new regional accent.
- ▶ Transfer general knowledge from pre-trained models to specialized tasks.  
Example: Transferring medical knowledge to a healthcare chatbot.
- ▶ Optimize model parameters for task-specific objectives and user preferences.  
Example: Optimizing a pre-trained model for code generation in software development.
- ▶ Handle modest datasets and mitigate sub-optimal performance
- ▶ Bridges gap between general-purpose and specialized models.
- ▶ Enable continual learning and adaptation to evolving data and user needs.
- ▶ Improve factual responses and reduce hallucinations.
- ▶ Cost and resource efficiency compared to training from scratch.



# Advantages of PEFT

## ▶ Computational Efficiency:

- ▶ PEFT fine-tunes LLMs with significantly fewer parameters than full fine-tuning.
- ▶ Feasible on less powerful hardware or in resource-constrained environments.

## ▶ Memory Efficiency:

- ▶ Freezing pretrained model weights minimizes excessive memory usage.
- ▶ Suitable for tasks with memory constraints.

## ▶ Catastrophic Forgetting Mitigation:

- ▶ PEFT prevents catastrophic forgetting observed in full fine-tuning.
- ▶ Ensures retention of valuable information during adaptation to new tasks.

## ▶ Versatility Across Modalities:

- ▶ PEFT is effective in various modalities such as computer vision and audio.
- ▶ Applicable to a wide range of downstream tasks beyond natural language processing.

## Advantages of PEFT (Contd.)

- ▶ **Modular Adaptation for Multiple Tasks:**
  - ▶ PEFT's modular nature allows the same pretrained model to be adapted for multiple tasks.
  - ▶ Small task-specific weights are added, avoiding the need for full copies for different applications.
- ▶ **INT8 Tuning:**
  - ▶ PEFT includes INT8 (8-bit integer) tuning, showcasing adaptability to different quantization techniques.
  - ▶ Enables fine-tuning even on platforms with limited computational resources.

## Summary of PEFT

- ▶ PEFT offers a **practical and efficient solution** for fine-tuning large language models.
- ▶ Addresses **computational and memory challenges** while maintaining performance on downstream tasks.

## LoRA Advantages

- ▶ Significantly reduces the number of trainable parameters
- ▶ Maintains performance comparable to full fine-tuning
- ▶ Efficient memory and compute usage
- ▶ Suitable for a wide range of tasks and models
- ▶ Allows fine-tuning on consumer-grade hardware



## LoRA Limitations

- ▶ Performance may degrade for very low-rank approximations
- ▶ Potential for instability or divergence with extreme rank values
- ▶ May not be as effective for very small or very large models
- ▶ Requires careful tuning of rank and other hyperparameters

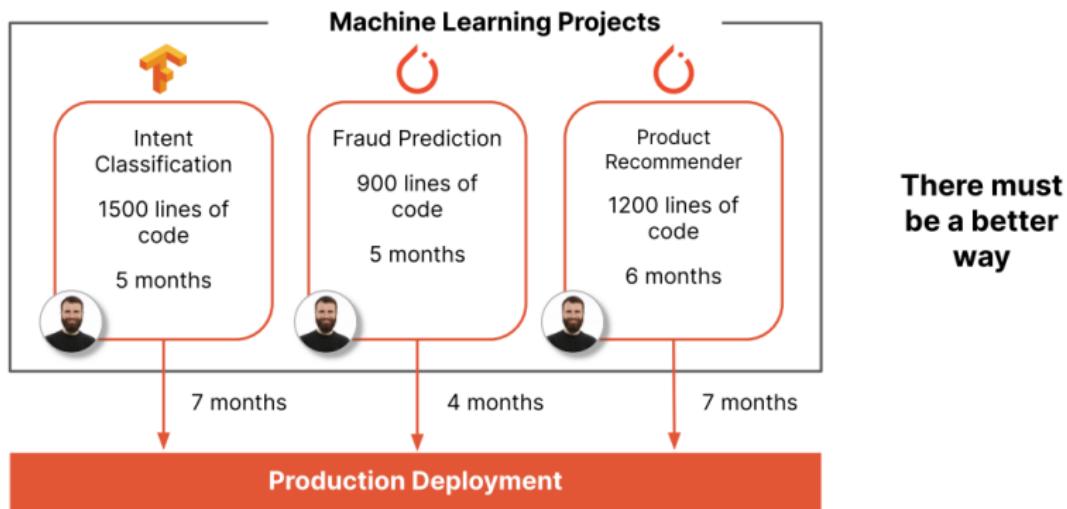
## Summary of LoRA

- ▶ LoRA is an efficient fine-tuning technique for large language models
- ▶ Reduces computational cost while maintaining performance
- ▶ Enables fine-tuning on consumer-grade hardware
- ▶ Applicable to various tasks and models
- ▶ Active area of research with ongoing improvements and extensions

# Fine-tuning Using Ludwig

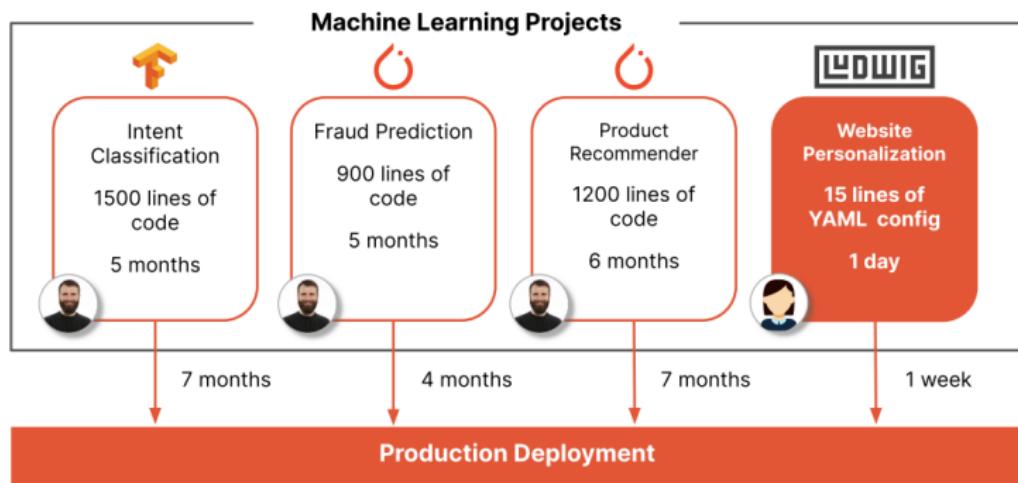
(Ref: Efficiently Build Custom LLMs on Your Data - <https://www.youtube.com/watch?v=NAyKpcOdHLE> - Piero Molino, Arnav Garg)

# Current State of ML Projects



Lot of time needed, many pieces are boiler-plate.

# Solution by LUDWIG



# Flexibility Levels

An open-source declarative ML framework started at Uber

## Easy to start

```
input_features:  
  name: sentence  
  type: text  
output_features:  
  name: intent  
  type: category
```

From months to days  
No ML code required  
Readable & Reproducible

## Expert level control

```
input_features:  
  name: sentence  
  type: text  
  encoder: bert  
output_features:  
  name: intent  
  type: category  
  trainer:  
    regularize: 0.1  
    dropout: 0.05
```

Easy to Iterate  
Extensible

## Advanced functionalities

```
input_features:  
  name: sentence  
  type: text  
output_features:  
  name: intent  
  type: category  
hyperopt:  
  dropout: [0.1, ...]  
  encoder: [llama, ...]  
  ...
```

Hyperparameter search  
State-of-the-art models  
Distributed training

Just specify what you want (declarative) and not how to get it done. Iterating on different customization is easy. Just change the config and run!!

# Introduction

- ▶ Large Language Models (LLMs) have revolutionized NLP tasks.
- ▶ Fine-tuning LLMs for specific tasks is crucial.
- ▶ Ludwig framework from Pedibase offers efficient fine-tuning.



# Large Language Modeling Fine-Tuning using Ludwig

- ▶ Ludwig: Open-source AI toolbox by Uber
- ▶ Supports fine-tuning of pre-trained language models
- ▶ Provides easy-to-use data preprocessing and model training
- ▶ Supports multi-task learning and transfer learning
- ▶ Flexible data input formats (CSV, JSON, pandas DataFrame)
- ▶ Automatic metric computation and visualizations
- ▶ Distributed training support
- ▶ Serialization and deployment of trained models
- ▶ Active development and community support



# Setup

- ▶ Install Ludwig framework: `pip install ludwig`.
- ▶ Prepare data in required format.
- ▶ Define model architecture and parameters.

## Fine-tuning Process

- ▶ Load pre-trained LLM using Ludwig.
- ▶ Specify task-specific data for fine-tuning.
- ▶ Train the model with Ludwig's `train` command.

## Task-Specific Tuning

- ▶ Adapt pre-trained LLM to specific tasks (e.g., text generation, classification).
- ▶ Configure task-specific parameters (e.g., learning rate, batch size).
- ▶ Fine-tune on task-specific data.

## Evaluation

- ▶ Assess fine-tuned model performance using evaluation metrics.
- ▶ Validate against task-specific benchmarks.
- ▶ Iterate fine-tuning process if necessary.

# Hyperparameter Optimization

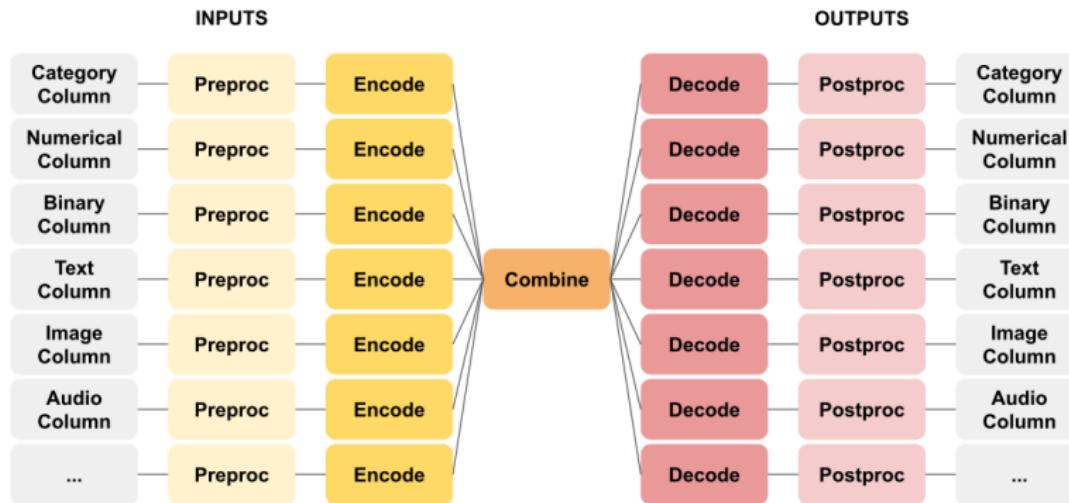
- ▶ Tune hyperparameters for optimal performance.
- ▶ Ludwig supports hyperparameter search.
- ▶ Utilize techniques like grid search or random search.

# Deployment

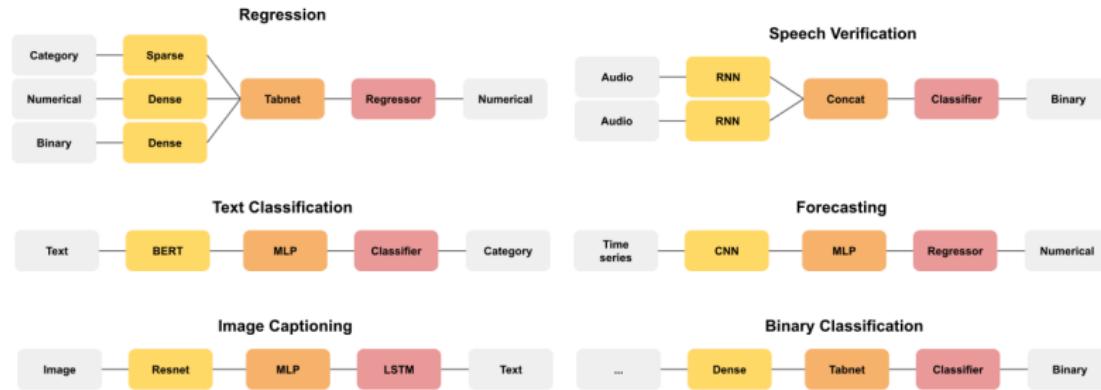
- ▶ Deploy fine-tuned model for inference.
- ▶ Integrate with existing applications or services.
- ▶ Monitor model performance in production.

# Ludwig Architecture

Many NLP tasks can be abstracted to Sequence To Sequence model.



# Ludwig Applications



{Input: [Category|Numerical|Binary], Output: [Numerical]} **is** a Regression problem

2

{Input: [Text], Output: [Category]} **is** a Text Classification problem

# Prompt Templating

## Prompt Template Definition

```
model_type: llm
base_model: Llama-2-7b-hf
prompt:
  task: "Rate this book review with from 1 to 5"
  template: |
    Task: {task}.
    Review: "{title} {review}".
    What score would you assign?
```

## Data

	title	review	score
	Amazing story!	This book made me dream of ...	4

## Input to LLM

```
Task: Classify this book review with a score from 1 to 5.
Review: "Amazing story! This book made me dream of ...".
What score would you assign?
```

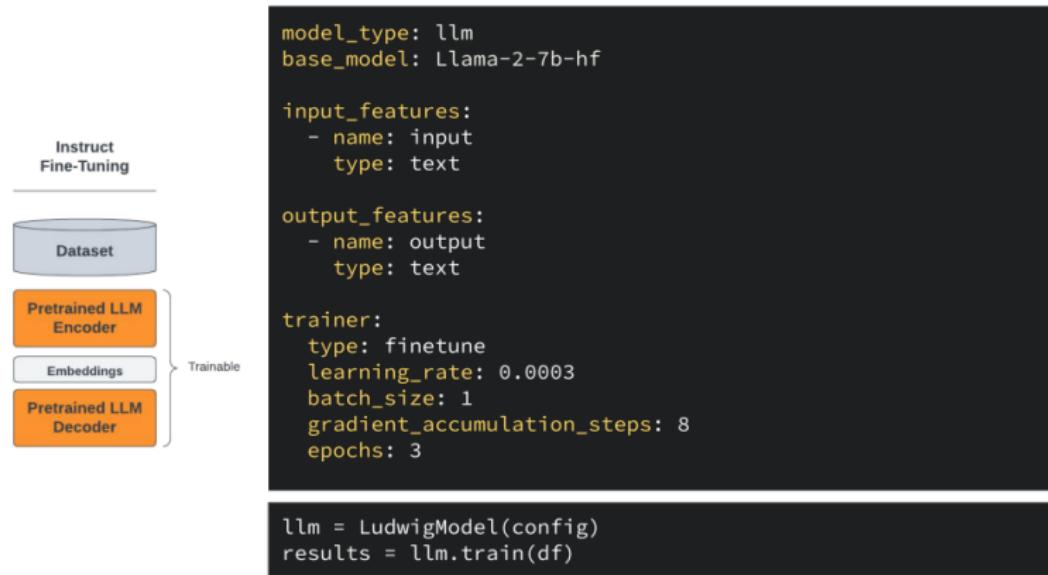
```
llm = LudwigModel(config)
llm.create_model()
results = llm.predict(df)
```

Ludwig looks at 'Prompt Template Definition' (using column names in 'Data') and 'Data' (as 'df') and then internally generates 'Input to LLM' for the row shown



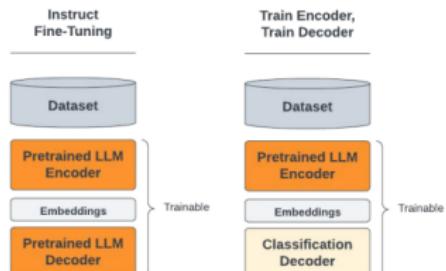
## Declarative-ly Fine-Tune LLMs

Full-Instruction based fine-tuning for a task. Trains all the layers including embedding, encoding and decoding parts.



# Declaratively Fine-Tune LLMs

New head 'Classification Decoder' is added for the sentiment analysis task.



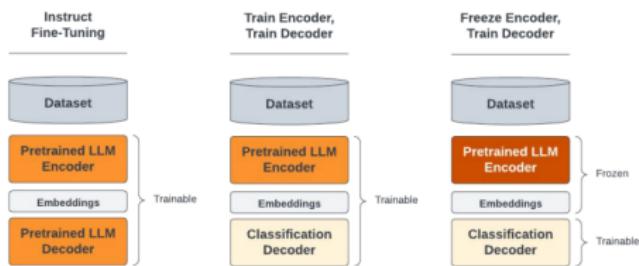
```
input_features:
- name: review
  type: text
  encoder:
    type: auto_transformer
    pretrained_model_name_or_path: Llama-2-7b-hf
    trainable: true

output_features:
- name: sentiment
  type: category
```

```
llm = LudwigModel(config)
results = llm.train(df)
```

# Declaratively Fine-Tune LLMs

## Freezing the Large Language Model.

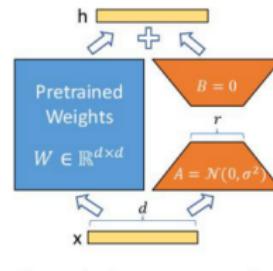


```
input_features:  
  - name: review  
    type: text  
    encoder:  
      type: auto_transformer  
      pretrained_model_name_or_path:  
        Llama-2-7b-hf  
      trainable: false  
      preprocessing:  
        cache_encoder_embeddings: true  
  
output_features:  
  - name: sentiment  
    type: category
```

```
llm = LudwigModel(config)  
results = llm.train(df)
```

# Parameter efficient fine-tuning

- LoRA
- AdaLoRA
- Adaptation Prompt  
(aka, LLaMA Adapter)
- QLoRA

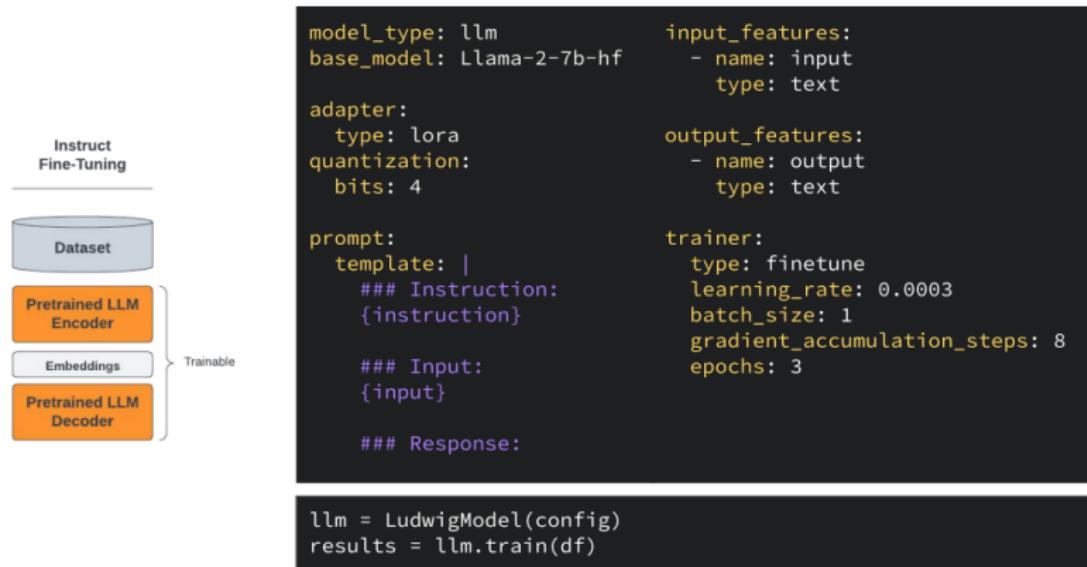


```
adapter:  
  type: lora  
  r: 16  
  alpha: 32  
  dropout: 0.1
```

```
adapter:  
  type: lora  
quantization:  
  bits: 4
```

# Putting it all together



## Example Application

Hands-on Tutorial available at:

<https://medium.com/google-cloud/gemma-for-gst-4595d5f60b6b> Or at  
“llm\_Gemma\_QnA\_GST\_FAQs\_Ludwig.ipynb”



## Conclusion

- ▶ Ludwig framework simplifies fine-tuning of LLMs.
- ▶ Enables efficient adaptation to diverse tasks.
- ▶ Empowers practitioners to leverage state-of-the-art language models effectively.



## References

Many publicly available resources have been referred for making this presentation. Some of the notable ones are:

- ▶ When to Fine-Tuning Large Language Models? - Thierry Teisseire
- ▶ Fine-tuning Large Language Models - Deeplearning ai  
<https://learn.deeplearning.ai/courses/fintuning-large-language-models/lesson/1/introduction>
- ▶ Fine-tuning LLMs with PEFT and LoRA <https://youtu.be/Us5ZFp16PaU>
- ▶ Adapters: the game changer for fine-tuning - Geoffrey Angus  
<https://youtu.be/BQYD9HivFLY>
- ▶ Fine-tuning Large Language Models (LLMs) — w/ Example Code - Shaw Talebi  
<https://www.youtube.com/watch?v=eC6Hd1hFvos>
- ▶ LLM Crash Course Part 1 - Finetune Any LLM for your Custom Usecase End to End in under[1 hour]!! - Neural Hacks with Vasanth.  
<https://www.youtube.com/watch?v=whbuNo6APVs> ,  
[https://github.com/Vasantheengineer4949/NLP-Projects-NHV/LLMs\\_Related/LLM\\_Crash\\_Course/Finetune\\_any\\_LLM\\_crash\\_course.ipynb](https://github.com/Vasantheengineer4949/NLP-Projects-NHV/LLMs_Related/LLM_Crash_Course/Finetune_any_LLM_crash_course.ipynb)

Thanks ...

- ▶ Search "**Yogesh Haribhau Kulkarni**" on Google and follow me on LinkedIn and Medium
- ▶ Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- ▶ Email: yogeshkulkarni at yahoo dot com



(Generated by Hugging Face QR-code-AI-art-generator,  
with prompt as "Follow me")