# Introduction to Fine-Tuning

Yogesh Haribhau Kulkarni

YHK

# Outline



YHK

# Fine-tuning LLMs (Large Language Models)

YHK

# Introduction to Fine-Tuning

- **Definition:** Process of training pre-existing models on smaller, domain-specific datasets to enhance task or domain performance.
- **Example:** Healthcare organization fine-tunes GPT-3 for generating patient reports, adapting to medical terminologies.
- **Base Model:**: Are typically raw, 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).
- **Applicability Beyond LLMs:** Convolutional neural network fine-tuning for detecting trucks on highways.

YHK

# 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 suboptimal performance
- 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.

YHK

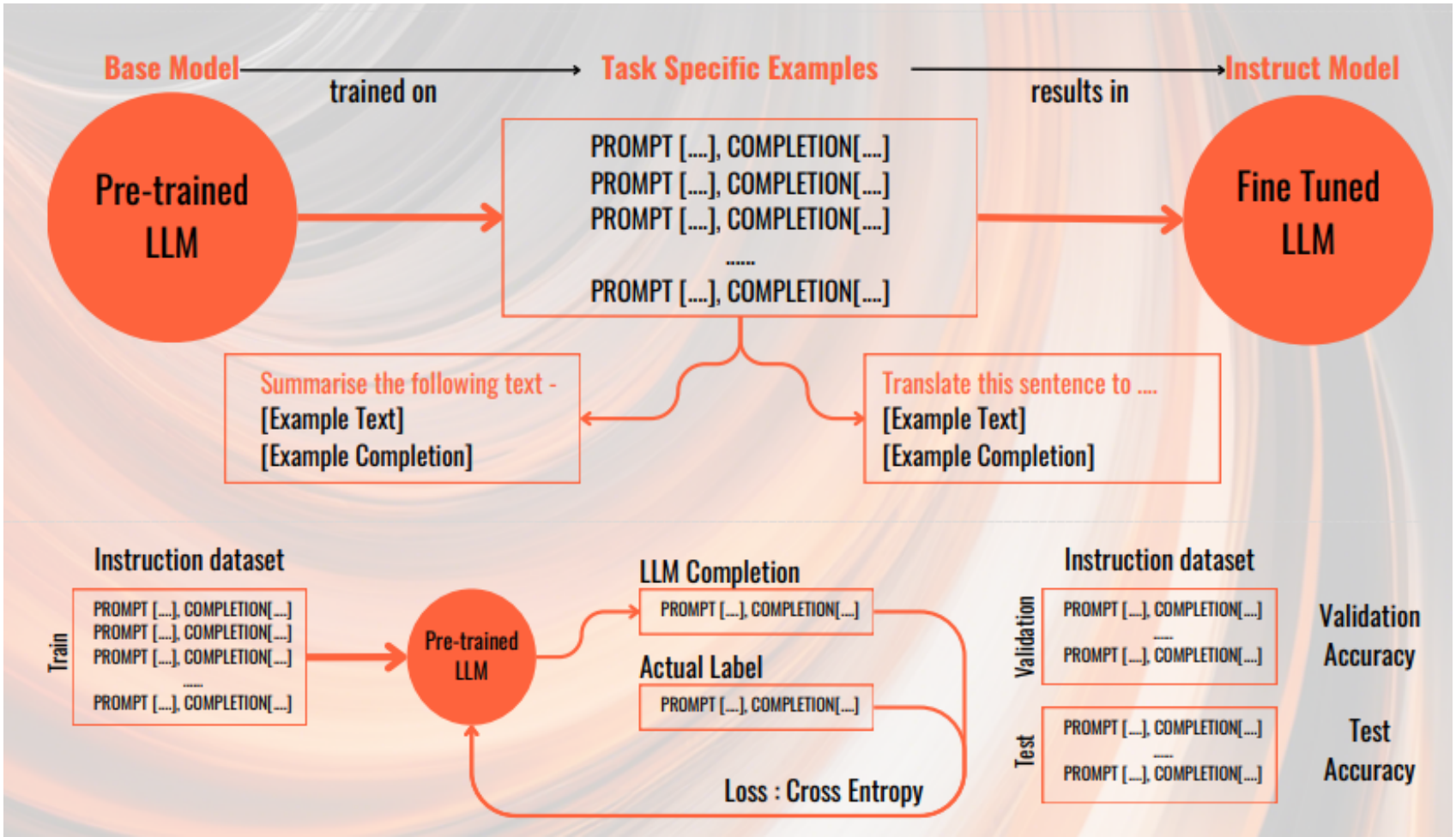
# What is a fine-tuned LLM?

- **Fine Tuning Overview:**
  - Supervised learning process adjusting LLM weights.
  - Uses a labeled dataset of prompt-completion pairs.
- **Instruction Fine Tuning:**
  - Trains LLM on examples of instructions and desired responses.
  - Improves performance on instruction-specific tasks.
- **Full Fine Tuning:**
  - Updates all LLM parameters.
  - Requires sufficient memory for storing and processing gradients and components.

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

YHK

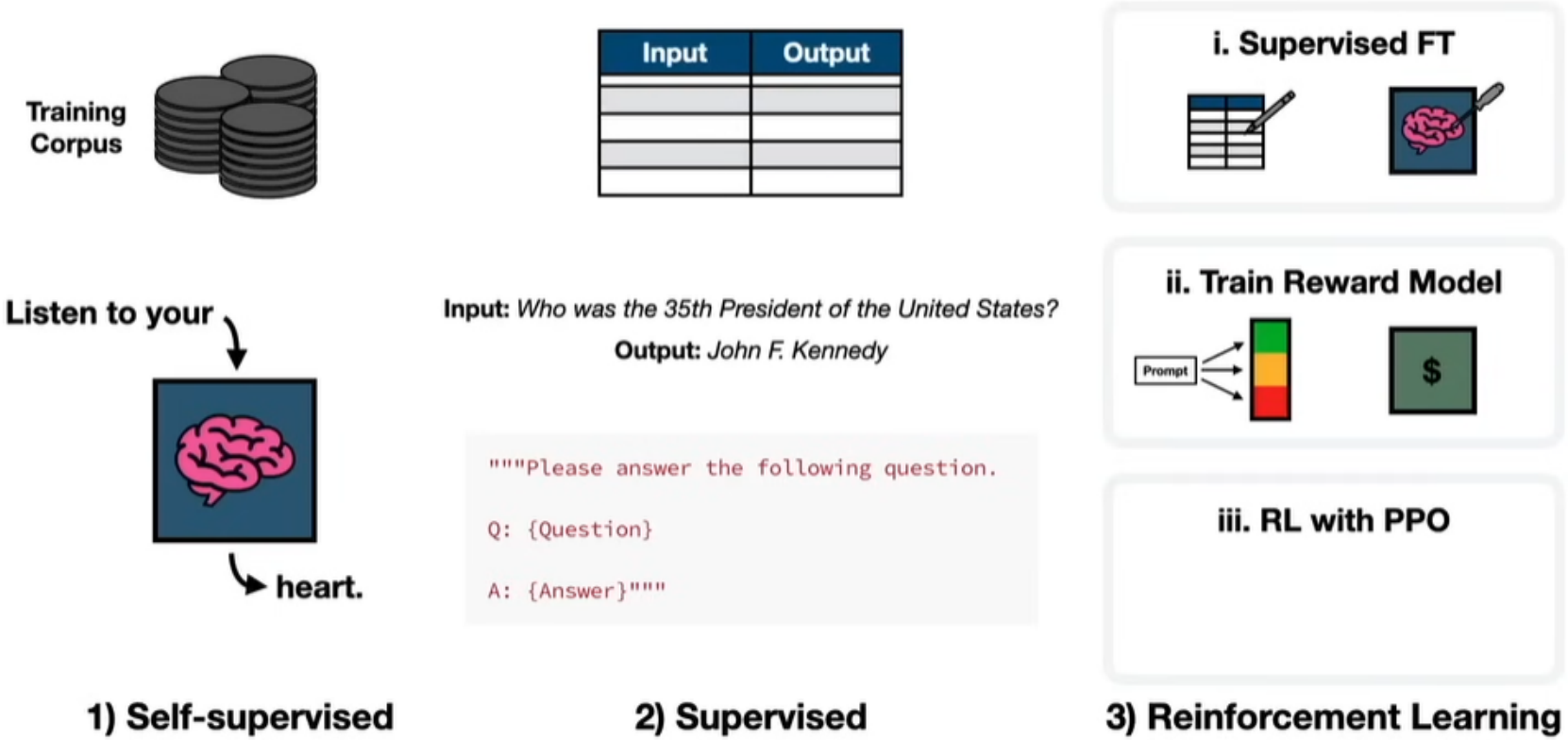
# What is a fine-tuned LLM?




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

YHK

# Ways to fine-tune

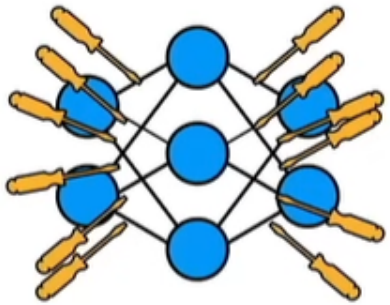



(Ref: Fine-tuning Large Language Models (LLMs) — w/ Example Code - Shaw Talebi)

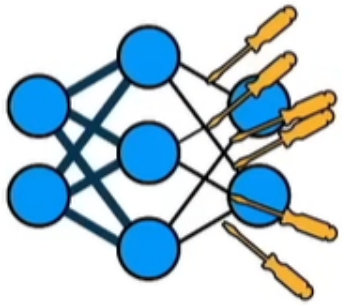
YHK

# Ways to change parameters/weights

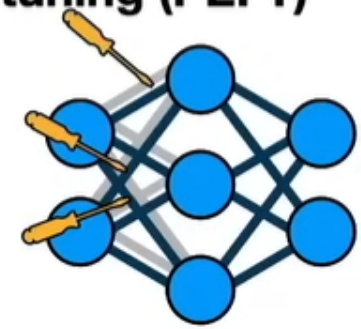
1) Retrain all parameters



2) Transfer Learning



3) Parameter Efficient Fine-tuning (PEFT)



ɔ][6]

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

YHK

# Task specific Fine-Tuning

- Widely used for customizing pre-trained LLMs to specific tasks.
- Involves training on labeled prompt-completion pairs.
- Allows model to adjust weights for task alignment.

YHK

# Unsupervised Fine-Tuning Methods

- **Unsupervised Full Fine-Tuning:**
  - Relevant for updating LLM knowledge base without changing existing behavior.
  - Example: Fine-tuning on legal literature or adapting to a new language using unstructured datasets.
- **Contrastive Learning:**
  - Trains the model to discern between similar and dissimilar examples in the latent space.
  - Beneficial for tasks requiring nuanced understanding of similarities and distinctions.

YHK

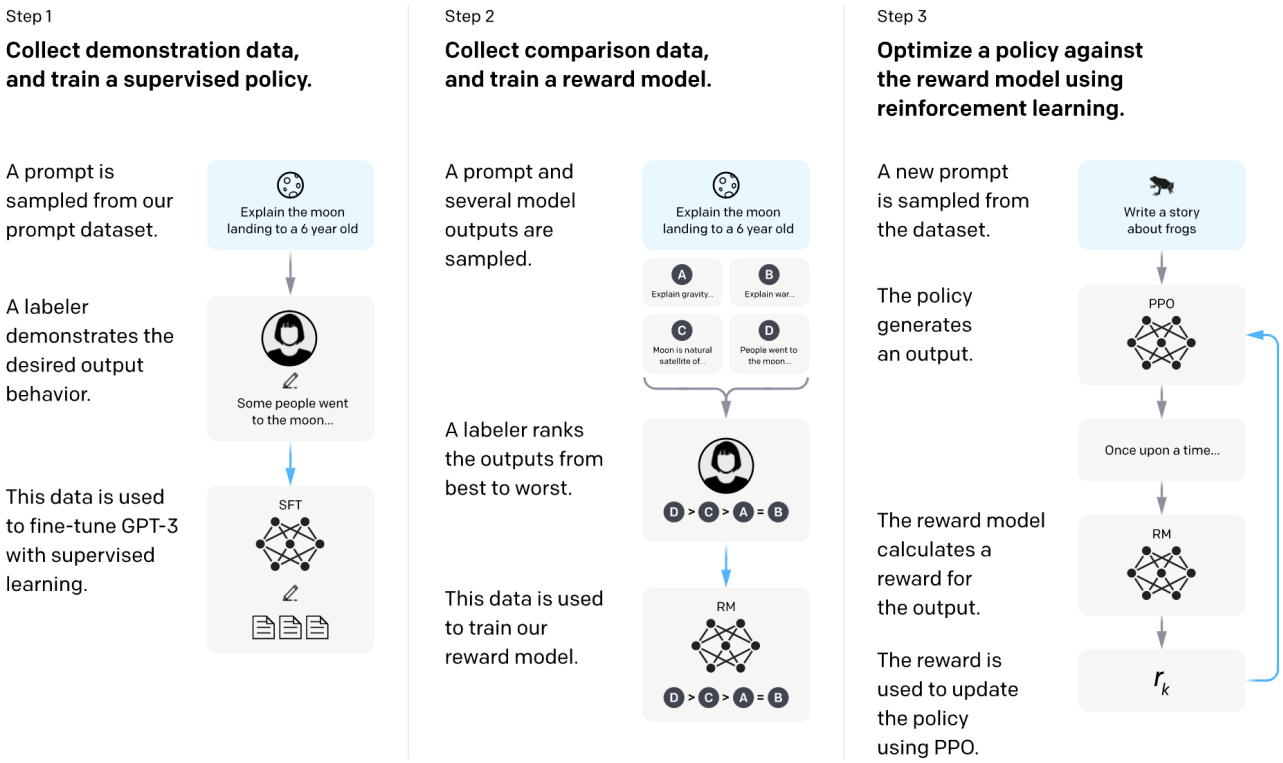
# Supervised Fine-Tuning Methods

- **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.
- **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.

YHK

# Reinforcement Learning from Human Feedback (RLHF)

- RLHF enhances language models by incorporating human feedback.
- Aims to align models more closely with intricate human values.




(Ref: Applied LLMs Mastery 2024 - Aishwarya Reganti)

YHK

# Parameter Efficient Fine Tuning (PEFT)

YHK

# What is PEFT?

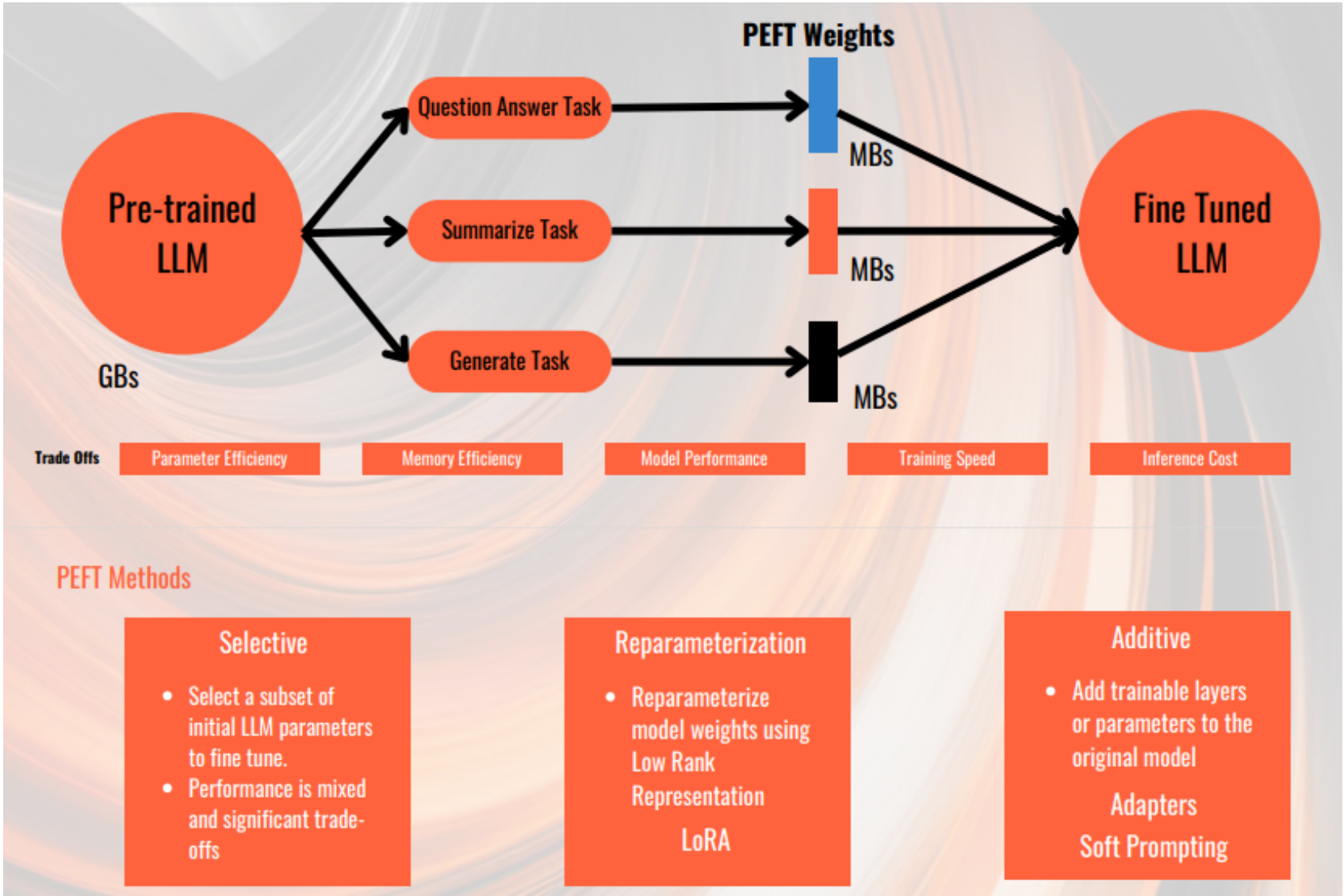
- **Full Fine Tuning:**
  - Requires memory for the entire model, optimizers, gradients, etc.
  - Similar memory demands as pre-training.
- **Parameter-Efficient Fine Tuning (PEFT):**
  - Fine-tunes only a subset of model parameters.
  - In some cases, leaves the original weights untouched.
- **Addressing 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)

YHK

# What is Parameter Efficient Fine Tuning?




YHK

# 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.

YHK

# 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.

YHK

# Summary of PEFT

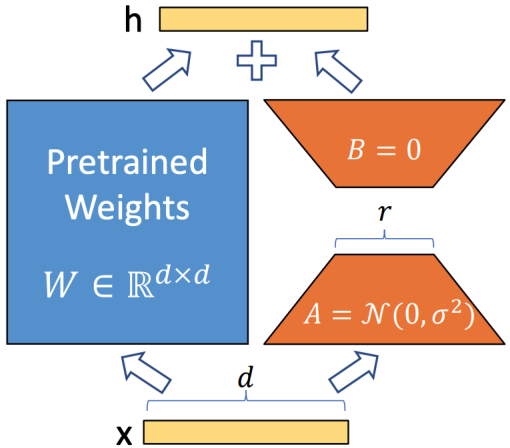
- PEFT offers a **practical and efficient solution** for fine-tuning large language models.
- Addresses **computational and memory challenges** while maintaining performance on downstream tasks.

YHK

# Low Rank Adaptation (LoRA)

# Why LoRA?

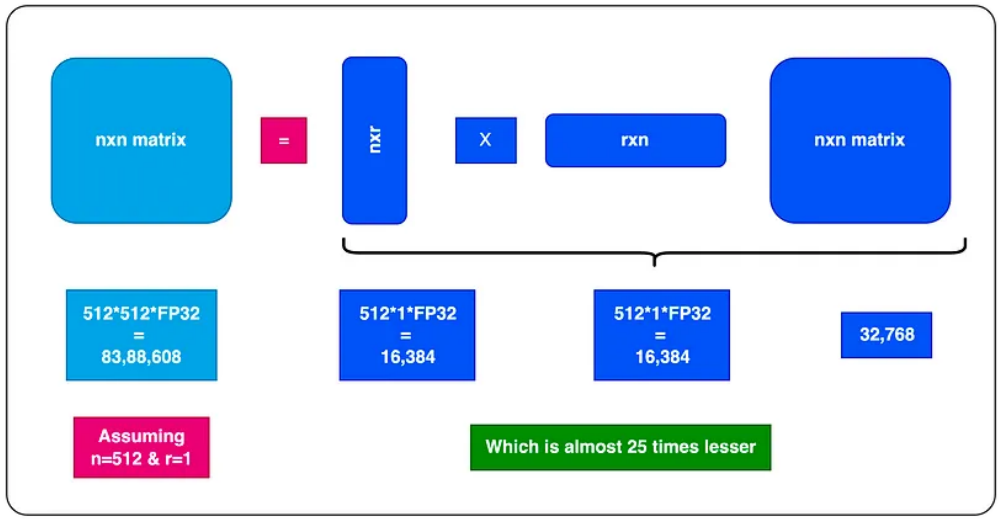
- Fine-tuning large language models (LLMs) is computationally expensive
- Updating all parameters of LLMs requires significant memory and compute resources
- LoRA aims to reduce the computational cost of fine-tuning while maintaining performance
- LoRA is one of the PEFT techniques, others being 'Prefix 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)

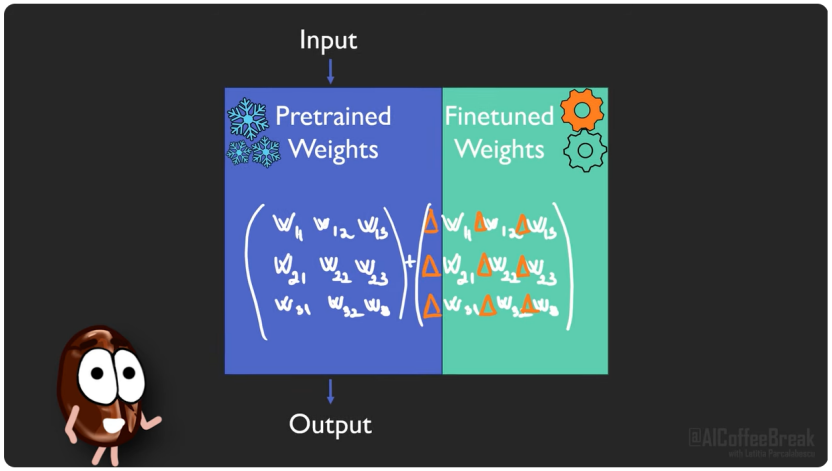



(Ref: https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4)

YHK

# 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 wrights, 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.

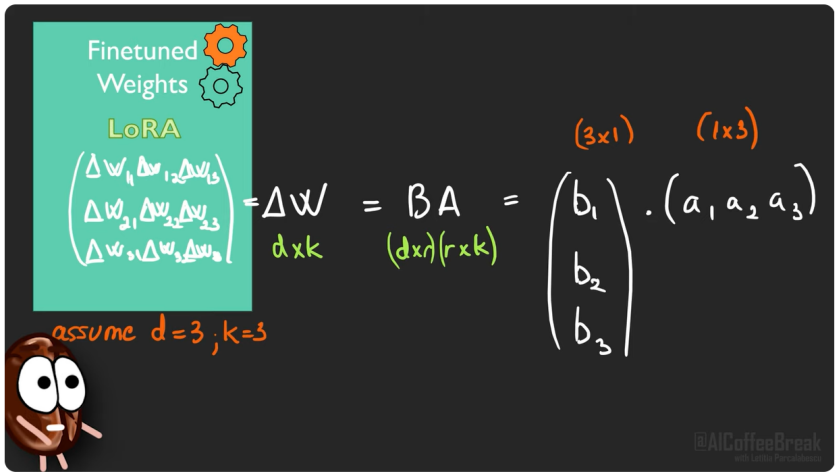



(Ref: What is LoRA? Low-Rank Adaptation for finetuning LLMs EXPLAINED-AI Coffee Break with Letitia)

YHK

# 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.

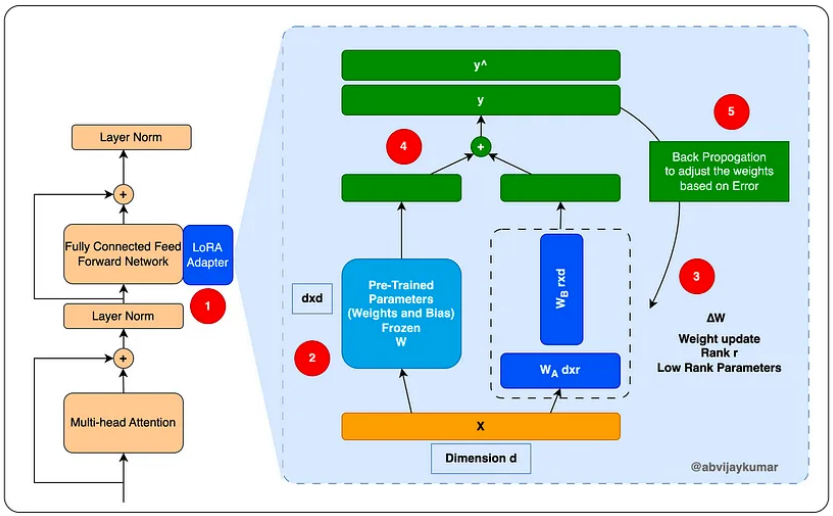



(Ref: What is LoRA? Low-Rank Adaptation for finetuning LLMs EXPLAINED-AI Coffee Break with Letitia)

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: code

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from peft import LoraConfig, get_peft_model, prepare_model_for_int8_training,
    TaskType

model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-xxl",
    load_in_8bit=True, device_map="auto")
lora_config = LoraConfig(
 r=16,
 lora_alpha=32,
 target_modules=["q", "v"],
 lora_dropout=0.05,
 bias="none",
 task_type=TaskType.SEQ_2_SEQ_LM
)
model = prepare_model_for_int8_training(model)
model = get_peft_model(model, lora_config)

model.print_trainable_parameters()
# trainable params: 18874368 || all params: 11154206720 || trainable%:
    0.16921300163961817
```

YHK

# 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

YHK

# 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

YHK

# 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

YHK

# 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

YHK

# 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

YHK

# Conclusion

- 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

YHK

# Fine-tuning Using Ludwig

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

# 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.

YHK

# Large Language Modeling Fine-Tuning using Ludwig

- Ludwig: Open-source AI toolbox by Uber
- Supports fine-tuning of pre-trained language models
- Supports popular models like BERT, RoBERTa, GPT-2
- 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 (Horovod, Ray)
- Serialization and deployment of trained models
- Active development and community support

YHK

# Setup

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

YHK

# Fine-tuning Process

- Load pre-trained LLM (e.g., GPT-3) using Ludwig.
- Specify task-specific data for fine-tuning.
- Train the model with Ludwig's `train` command.

YHK

# 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.

YHK

# Evaluation

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

YHK

# Hyperparameter Optimization

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

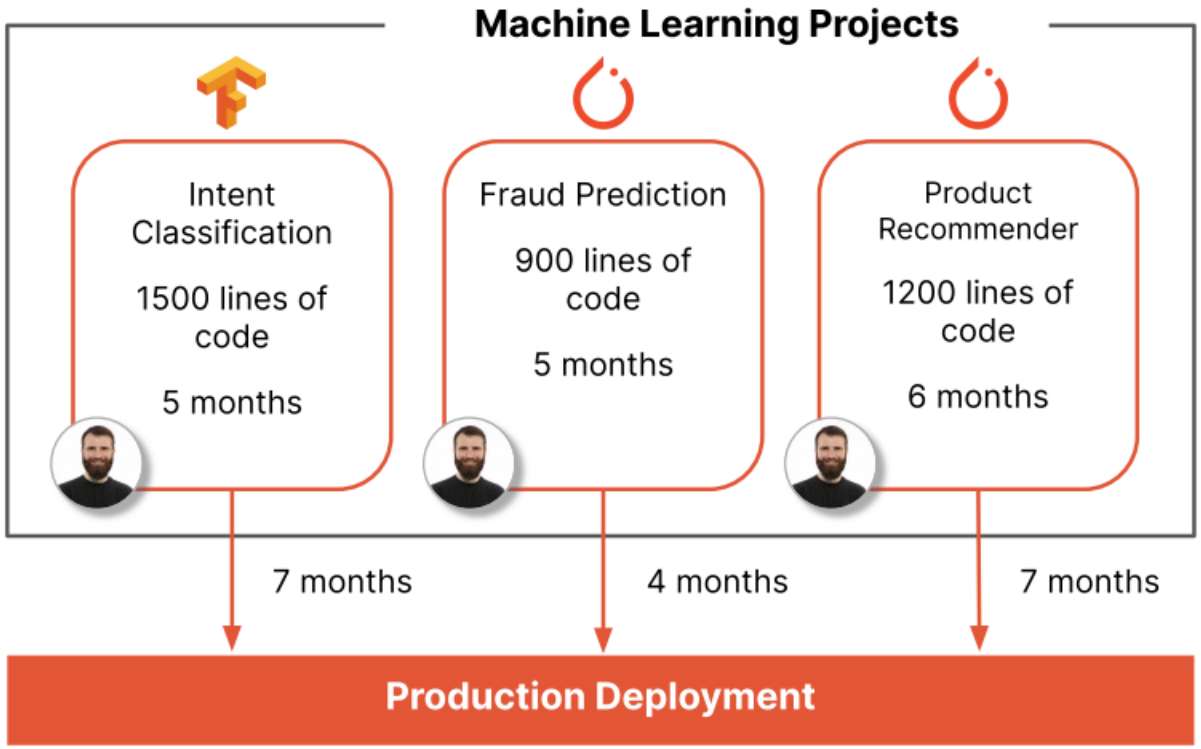
YHK

# Deployment

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

YHK

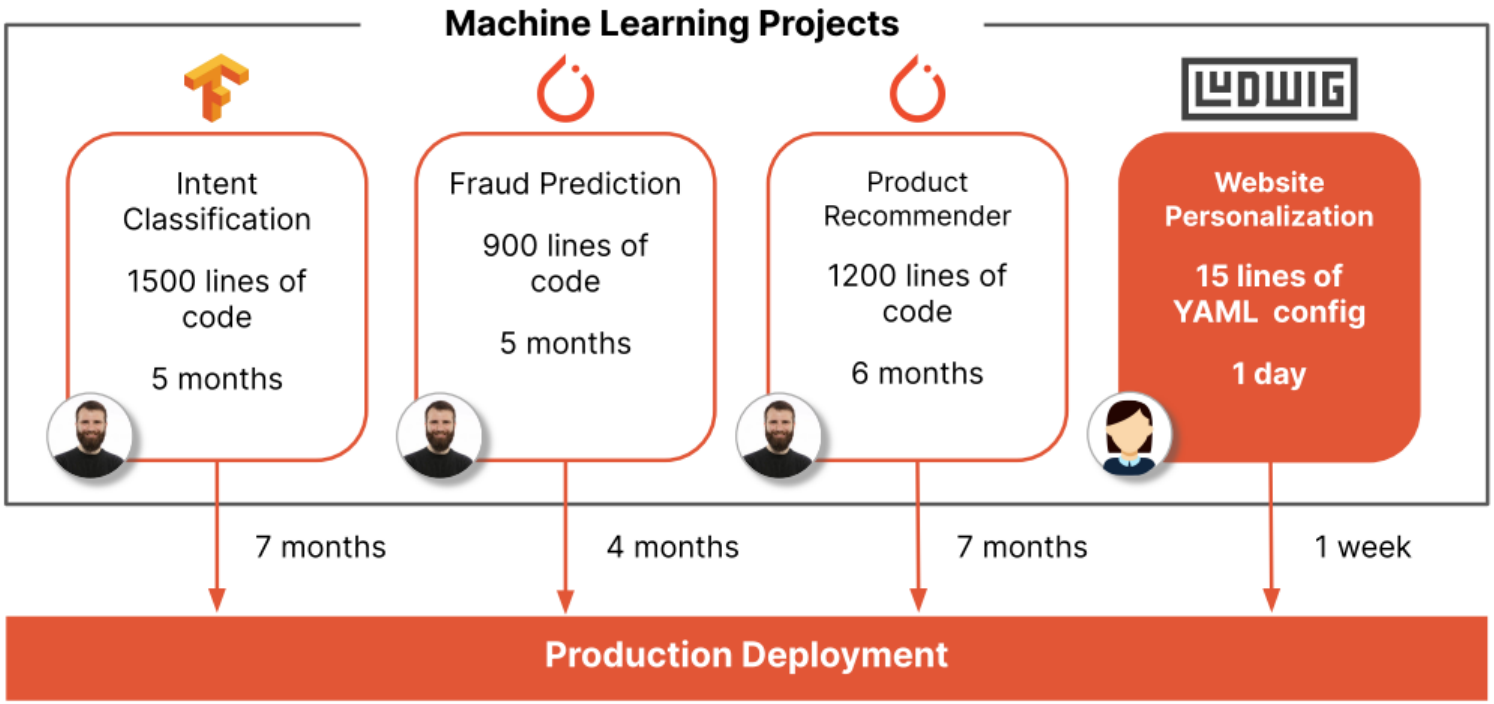
# Current State of ML Projects




**There must be a better way**

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, ...]
  ...
```

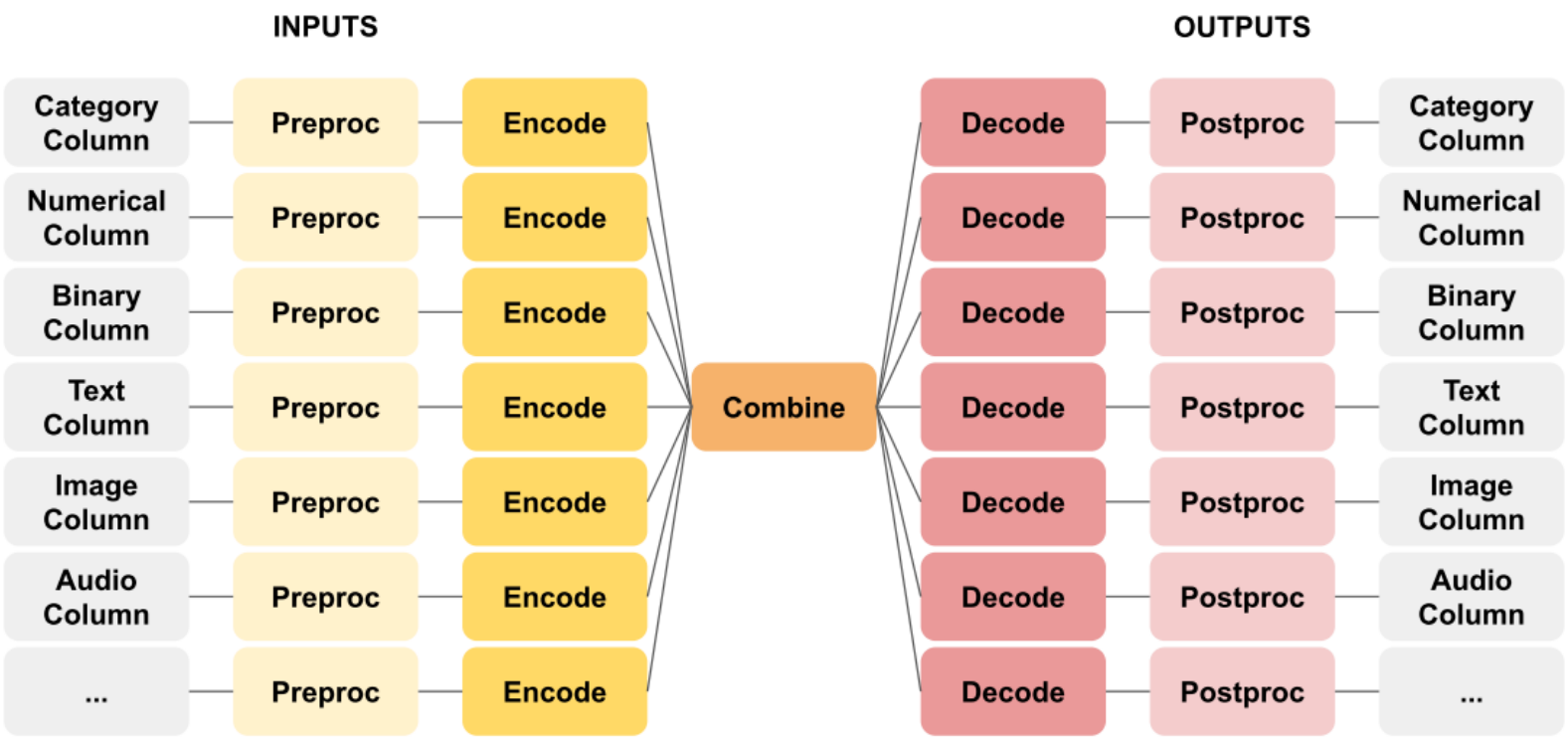
Hyperparameter search
State-of-the-art models
Distributed training

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!!

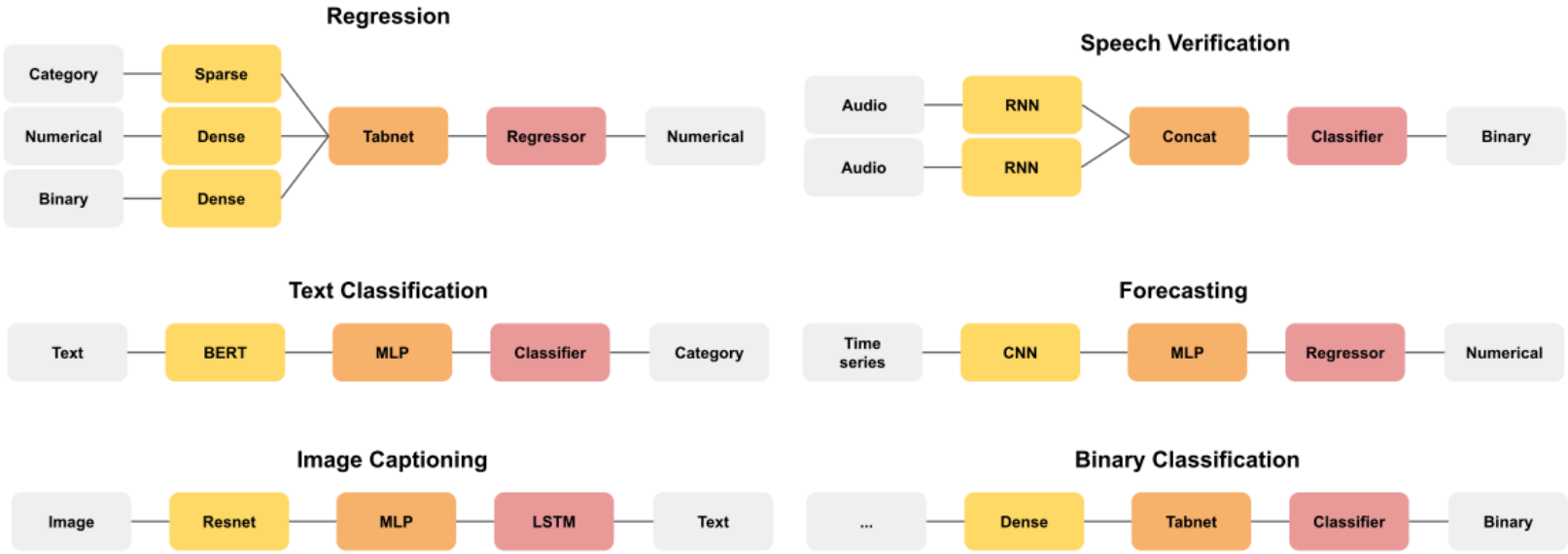
YHK

# 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

{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?
```

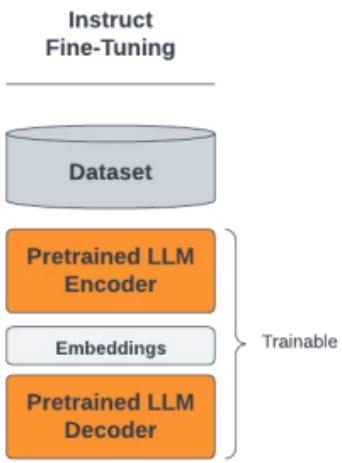
```
llm = LudwigModel(config)
llm.create_model()
results = llm.predict(df)
```

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

YHK

# Declarative-ly Fine-Tune LLMs

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



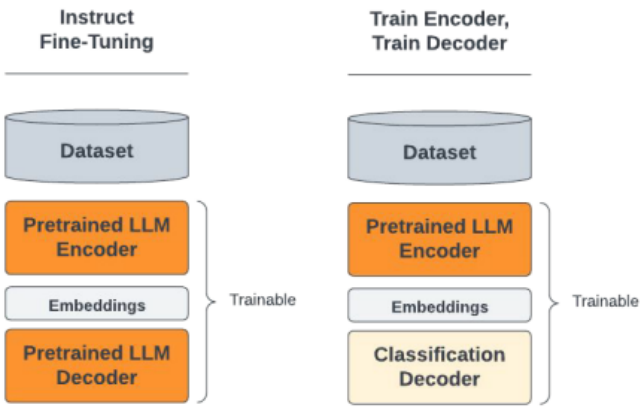

```yaml
model_type: llm
base_model: Llama-2-7b-hf

input_features:
  - name: input
    type: text

output_features:
  - name: output
    type: text

trainer:
  type: finetune
  learning_rate: 0.0003
  batch_size: 1
  gradient_accumulation_steps: 8
  epochs: 3
```

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

YHK

# Declaratively Fine-Tune LLMs

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



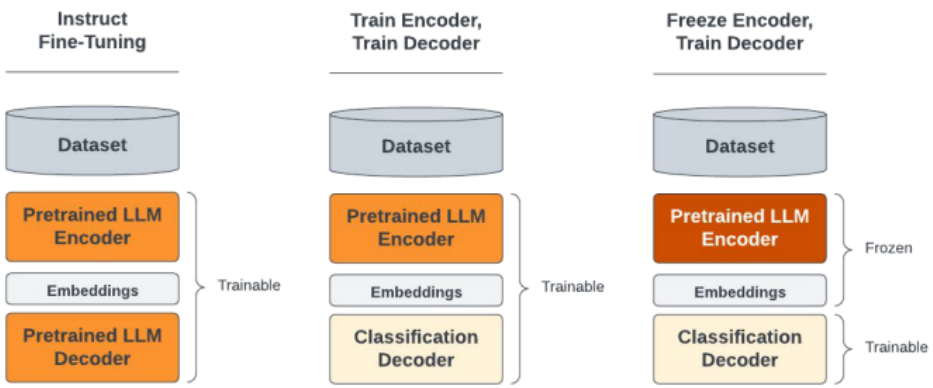

```
input_features:
  - name: review
    type: text
    encoder:
      type: auto_transformer
      pretrained_model_name_or_path: Llama-2-7b-hf
      trainable: true

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

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

YHK

# Declaratively Fine-Tune LLMs

Freezing the Large Language Model.




```yaml
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
```

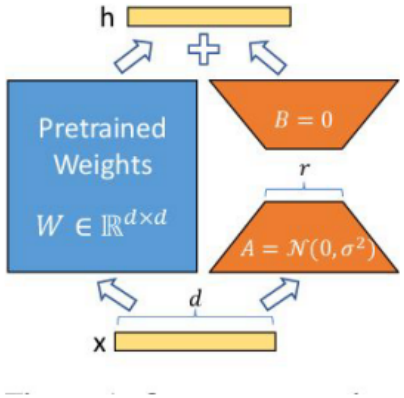
```
llm = LudwigModel(config)
results = llm.train(df)
```

YHK

# Parameter efficient fine-tuning

- LoRA
- AdaLoRA
- Adaptation Prompt

  (aka, LLaMA Adapter)
- QLoRA

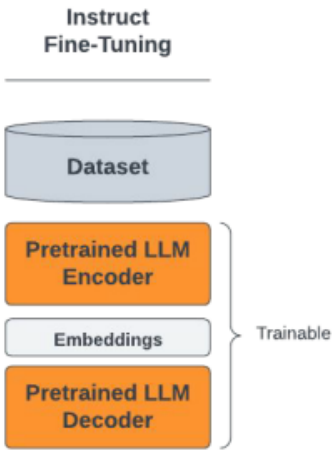



```yaml
adapter:
  type: lora
  r: 16
  alpha: 32
  dropout: 0.1
```

```yaml
adapter:
  type: lora

quantization:
  bits: 4
```

YHK

# Putting it all together




```yaml
model_type: llm
base_model: Llama-2-7b-hf

adapter:
  type: lora
quantization:
  bits: 4

prompt:
  template: |
    ### Instruction:
    {instruction}

    ### Input:
    {input}

    ### Response:

input_features:
  - name: input
    type: text

output_features:
  - name: output
    type: text

trainer:
  type: finetune
  learning_rate: 0.0003
  batch_size: 1
  gradient_accumulation_steps: 8
  epochs: 3
```

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

YHK

# 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”

YHK

# 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.

YHK

# References

Many publicly available resources have been refereed 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/finetuning-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/Vasanthengineer4949/NLP-Projects-NHV/LLMs Related/LLM Crash Course/Finetune_any_LLM_crash_course.ipynb


YHK

# 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")

YHK