# Report for paper

# The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs

A very smart but general-purpose AI assistant (this is like a pre-trained LLM). It knows a lot about the world and can understand and generate text pretty well. However, if you want it to be really good at a specific job, like summarizing legal documents or answering questions about a specific company, you need to teach it more specific things. This teaching process is called **fine-tuning**.

This report is like a guide that tells you everything you need to know to fine-tune these smart AI assistants. It starts by explaining what LLMs are and how they became so advanced, moving from simpler language models to the powerful systems we have today.

The report then outlines a **seven-step process** to fine-tune an LLM:

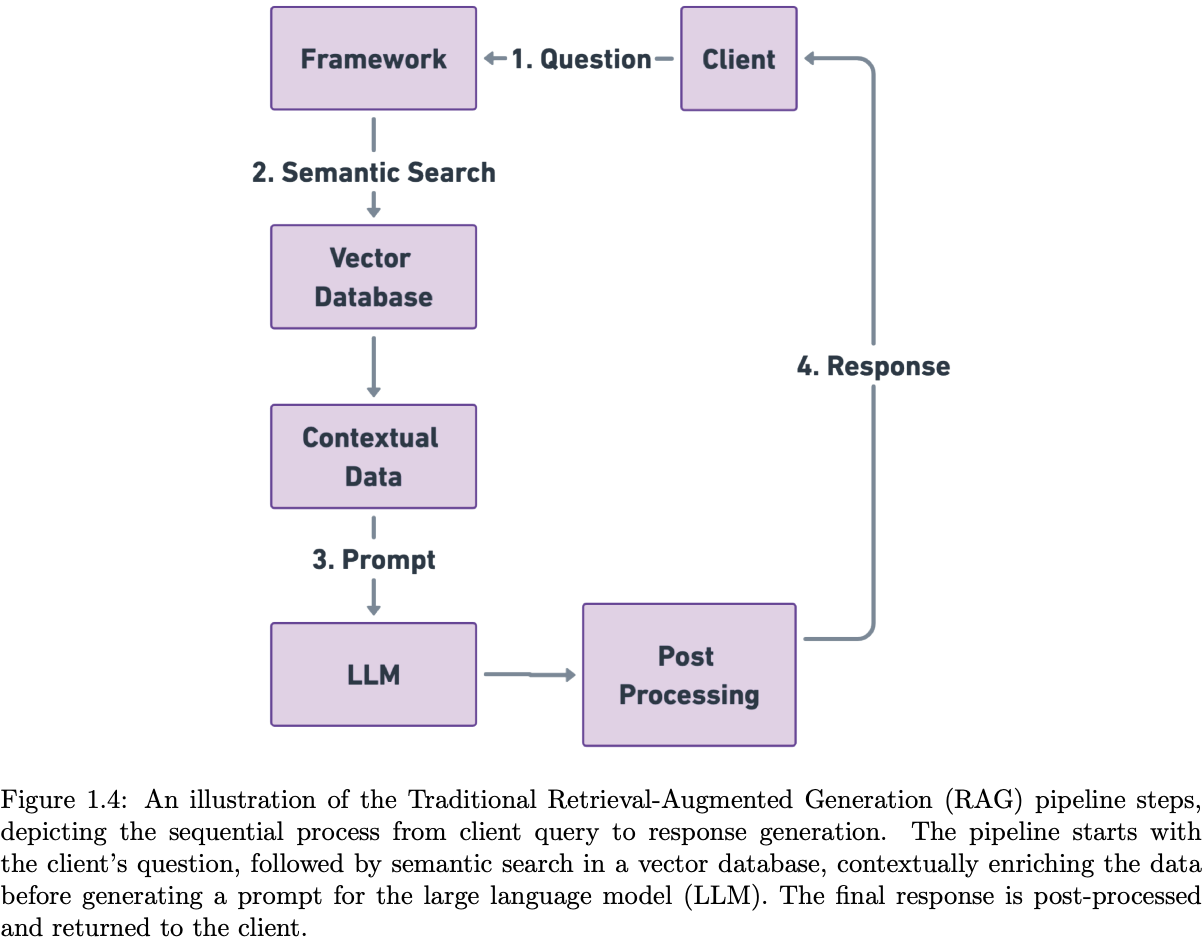
1. **Data Preparation:** Getting the right teaching materials (data) ready. This includes collecting the data, cleaning it up, and organizing it.
2. **Model Initialisation:** Starting with a pre-trained AI assistant as your base.
3. **Training Environment Setup:** Getting the classroom (computer and software) ready for the teaching process.
4. **Fine-Tuning:** Actually teaching the AI assistant using your prepared data. You can teach it everything or just focus on certain parts.
5. **Evaluation and Validation:** Testing how well the AI assistant has learned on new material it hasn't seen before to make sure it's doing a good job.
6. **Deployment:** Putting the newly trained AI assistant to work in the real world.
7. **Monitoring and Maintenance:** Keeping an eye on the AI assistant to make sure it continues to perform well and updating its knowledge when needed.

The report goes into each of these steps in more detail. For example, in **Data Preparation**, it talks about finding the right data and making sure it's in a good format for the AI to learn from. In **Fine-Tuning**, it discusses different ways to teach the AI, like showing it examples of what you want it to do. **Evaluation** is about checking if the AI can correctly answer questions or perform the specific task it was trained for.

**Chapter 1: Introduction**

This chapter lays the groundwork by introducing Large Language Models (LLMs) and the concept of fine-tuning.

* **Background of Large Language Models (LLMs):** LLMs are sophisticated AI models that can understand and generate human-like text. They are a big step up from older **traditional Natural Language Processing (NLP) models** like **N-gram models**. N-gram models were simpler and had limitations in handling rare words and understanding complex language. LLMs, like **GPT-3** and **GPT-4**, use a technology called the **Transformer architecture** and a mechanism called **self-attention**. Think of self-attention as the model paying closer attention to the most important words in a sentence when trying to understand it. This allows them to handle long sentences and complex relationships between words. Key advancements also include **in-context learning** (where the model can generate coherent text just from a prompt) and **Reinforcement Learning from Human Feedback (RLHF)**. RLHF is like getting feedback from a teacher to make the model's responses better aligned with what humans expect. Techniques like **prompt engineering** (crafting effective instructions), **question-answering**, and **conversational interactions** have significantly improved NLP thanks to LLMs.
* **Evolution from Traditional NLP Models to State-of-the-Art LLMs:** Understanding this evolution is key.  
  + **Statistical Language Models (SLMs):** These models used probabilities to predict the next word in a sequence. For example, the probability of the sentence "I am very happy" would be calculated based on the likelihood of each word following the previous ones. A simple example of how they learn is by counting how often certain word combinations appear in a large text. If "I am" appears frequently, the model learns that "am" is likely to follow "I".
  + **Neural Language Models (NLMs):** NLMs use **neural networks** to predict word sequences, overcoming limitations of SLMs. They use **word vectors** (numerical representations of words) to understand word meanings and relationships. For instance, "king" and "queen" would have similar vect
  + **Pre-trained Language Models (PLMs):** These models, like **BERT**, are trained on massive amounts of text data before being used for specific tasks. This pre-training allows them to learn general language understanding.
  + **Large Language Models (LLMs):** LLMs are essentially scaled-up PLMs with billions of parameters. The increase in size and training data has led to emergent abilities like in-context learning and improved understanding.
* **Overview of Current Leading LLMs:** LLMs are powerful for tasks like translation, summarization, and conversation. Their success is driven by advances in **transformer architectures**, increased **computational power**, and the availability of **large datasets**. Figure 1.3 (page 8) gives an overview of these models and their capabilities.
* **What is Fine-Tuning?** Fine-tuning is the process of taking a **pre-trained LLM** and further training it on a **smaller, task-specific dataset**. This adapts the general-purpose model to perform a particular task or work well in a specific domain.
* **Types of LLM Fine-Tuning:** There are different ways to fine-tune:  
  + **Unsupervised Fine-Tuning:** This uses **unlabelled data** from the target domain to help the model better understand the language of that domain. For example, you could fine-tune a general LLM on a large collection of legal documents without specific labels to make it more familiar with legal terminology. However, it's less precise for specific tasks like classifying documents.
  + **Supervised Fine-Tuning (SFT):** This involves training the LLM on **labelled data** where each piece of text is paired with the desired output or label. For instance, to fine-tune for sentiment analysis, you'd use text snippets labelled as "positive," "negative," or "neutral." While effective, it requires a good amount of labelled data, which can be expensive to get.
  + **Instruction Fine-Tuning via Prompt Engineering:** This method involves training the model using **natural language instructions** to guide its responses. Instead of just giving it examples, you tell it what you want it to do. For example, you might train it with prompts like "Summarize the following article in three sentences:" followed by the article and the desired summary. This can reduce the need for large amounts of labelled data but relies on well-written instructions.
* **Pre-training vs Fine-tuning:** Table 1.1 (page 9) clearly compares these two stages. **Pre-training** is like giving the AI a broad education on a huge amount of general knowledge, taking weeks or months and requiring vast unlabelled data. **Fine-tuning** is like specializing that education for a specific job using smaller, labelled datasets, taking days or weeks. For example, a model like **LLaMA 3** is first pre-trained on a massive dataset, and then it can be fine-tuned for summarization using a dataset of articles and their summaries.
* **Importance of Fine-Tuning LLMs:** Fine-tuning is crucial because it allows us to take these general-purpose LLMs and make them much better at specific tasks or in specific areas. Without fine-tuning, an LLM might be able to answer a wide range of questions but might not have the specialized knowledge or the right output style for a particular application.
* **Retrieval Augmented Generation (RAG):** This is another technique to improve LLM performance by allowing them to **access external data sources** when generating responses. Figure 1.4 (page 11) illustrates the RAG pipeline.  
  + **Traditional RAG Pipeline and Steps:** It involves indexing data (organizing it for quick search), processing the input query, searching and ranking relevant information, and then augmenting the original query with this retrieved information before feeding it to the LLM to generate a response.
  + **Benefits of Using RAG:** It allows the model to provide more accurate and up-to-date information by grounding its answers in external knowledge.
  + **Challenges and Considerations in Serving RAG:** These include business context awareness, service scalability, and security.
  + **Use Cases and Examples:** Examples include question-answering chatbots that can answer questions based on company documents, search engines that provide LLM-generated answers, and knowledge engines for internal company information.
  + **Considerations for Choosing Between RAG and Fine-Tuning:** RAG is better when you need to access external, frequently updated data and want to reduce the risk of the model making things up (**hallucinations**). Fine-tuning is more suitable for adjusting the model's behavior, writing style, or incorporating domain-specific knowledge that doesn't change frequently, especially if you have enough labelled data. Figure 1.5 (page 12) visually compares these approaches.

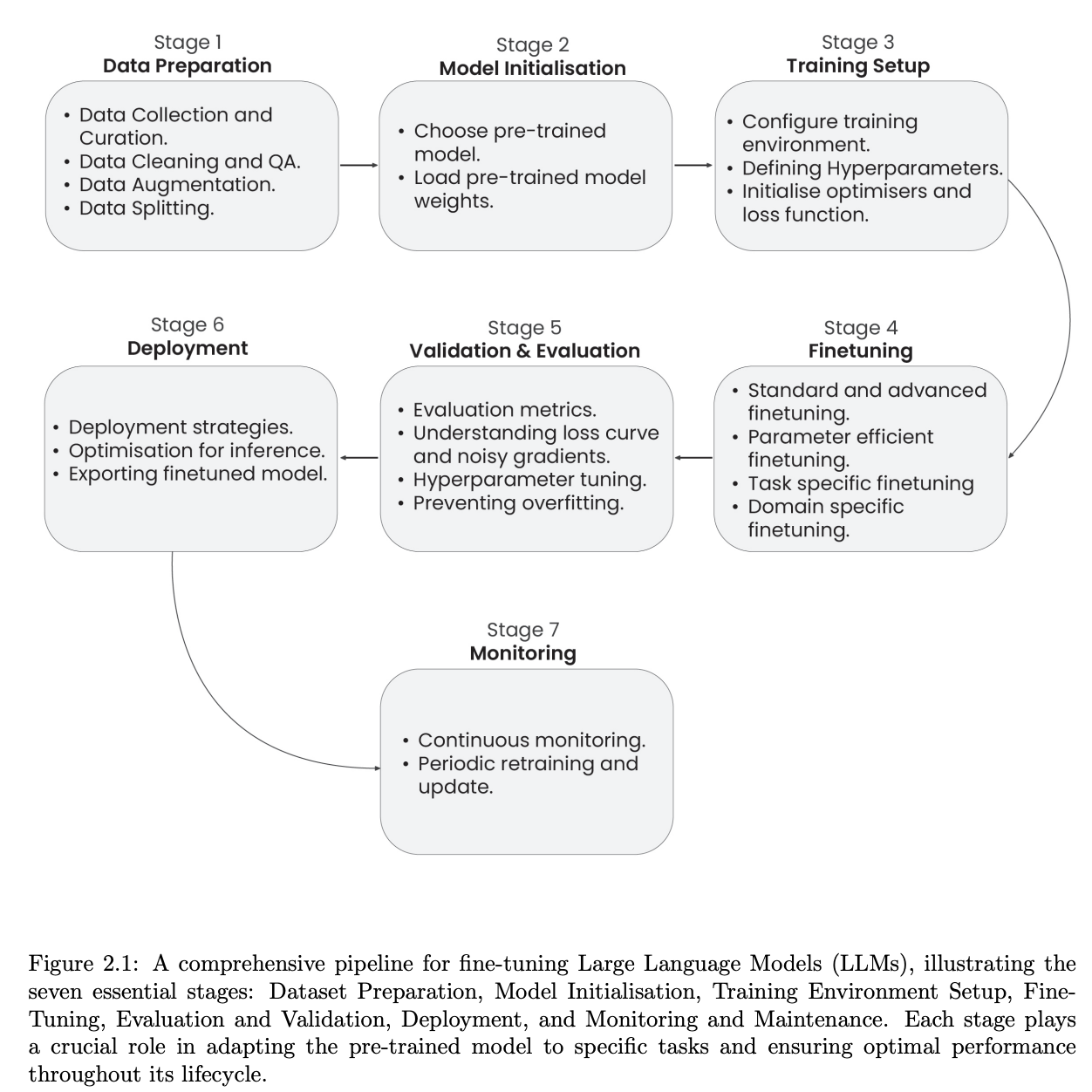


* **Objectives of the Report:** The main goal is to provide a comprehensive analysis of fine-tuning techniques for LLMs, covering theory, practical strategies, and challenges. The report aims to help researchers and practitioners effectively fine-tune LLMs for specific applications and domains.
* **Overview of the Report Structure:** The report will cover the fine-tuning pipeline in detail, practical applications, model evaluation, deployment, and ongoing research challenges.

**Chapter 2: Seven Stage Fine-Tuning Pipeline for LLM**

This chapter provides a high-level roadmap for fine-tuning LLMs, breaking it down into seven key stages. Figure 2.1 (page 14) visually represents this pipeline.

* **Stage 1: Dataset Preparation:** This involves collecting, cleaning, preprocessing, and splitting the data that will be used to fine-tune the model.
* **Stage 2: Model Initialisation:** This is about selecting and loading the pre-trained LLM that you want to fine-tune.
* **Stage 3: Training Environment Setup:** This involves setting up the necessary hardware and software, defining hyperparameters (settings for the training process), and initializing optimizers (algorithms that help the model learn) and loss functions (metrics that measure how well the model is doing).
* **Stage 4: Partial or Full Fine-Tuning:** This is the core training stage where the model's parameters are updated using the prepared dataset. **Full fine-tuning** updates all the model's parameters, while **partial fine-tuning** (like **Half fine-tuning (HFT)** or using **Parameter-Efficient Fine-Tuning (PEFT)** techniques such as **adapter layers**) only updates a small portion. This can be more efficient and prevent **overfitting** (where the model learns the training data too well and doesn't generalize to new data).
* **Stage 5: Evaluation and Validation:** This stage involves assessing the performance of the fine-tuned model on data it hasn't seen during training to ensure it generalizes well and meets the desired goals. **Evaluation metrics** like cross-entropy are used to measure prediction errors, and **validation** helps detect issues like overfitting.
* **Stage 6: Deployment:** This is about making the fine-tuned model operational and accessible for its intended use, including setting it up on hardware or cloud platforms and integrating it with applications.
* **Stage 7: Monitoring and Maintenance:** After deployment, it's important to continuously monitor the model's performance, update its knowledge if needed, and ensure it remains reliable and secure.



**Chapter 3: Stage 1: Data Preparation**

This chapter dives into the crucial first step of the fine-tuning pipeline: getting your data ready.

* **Steps Involved in Data Preparation:**
  + **Data Collection:** Gathering data from various sources like CSV files, web pages, SQL databases, or cloud storage (e.g., S3). Python libraries like **pandas** (for reading CSV and other formats), **BeautifulSoup** and **requests** (for web scraping), **SQLAlchemy** (for databases), and **boto3** (for AWS S3) are commonly used for this. Table 3.1 (page 17) lists these tools.
  + **Data Preprocessing and Formatting:** Cleaning the data (removing errors, duplicates), handling missing values, and formatting it into a structure that the LLM can understand. Table 3.2 (page 18) mentions Python libraries like **spaCy**, **NLTK**, **HuggingFace Transformers**, and **KNIME** for text preprocessing. Examples of preprocessing include converting text to lowercase, removing punctuation, or tokenizing (splitting text into individual words or sub-words).
  + **Handling Data Imbalance:** If your dataset has significantly more examples of one category than others (e.g., many "positive" reviews but few "negative" ones in sentiment analysis), this can bias your model. Techniques to address this include **oversampling** (duplicating examples from the minority class), **undersampling** (removing examples from the majority class), and **stratified sampling** (ensuring each mini-batch during training has a proportional representation of each class). Python's **scikit-learn** library provides tools for stratified sampling.
  + **Splitting Dataset:** Dividing your prepared data into at least two parts: a **training set** (used to train the model) and a **validation set** (used to evaluate the model during training to monitor its progress and prevent overfitting). A common split is 80% for training and 20% for validation. Techniques for splitting include **random sampling**, **stratified sampling** (to maintain class balance in both sets), **K-Fold Cross Validation** (splitting the data into K parts and training/validating K times using different parts as validation), and **Leave-One-Out Cross Validation** (using one data point for validation and the rest for training, repeated for every data point). Python's **scikit-learn** library provides functions for these splitting techniques.
* **Existing and Potential Research Methodologies:**
  + **Data Annotation:** For supervised fine-tuning, you need to label your data with the correct answers or categories. This can be done manually by humans (which is accurate but can be slow and expensive for large datasets) using tools like Excel, Prodigy, or Innodata. There's also research into automated or semi-automated annotation methods.
  + **Data Augmentation:** Creating new training examples by modifying existing ones (e.g., paraphrasing text, adding noise to audio) to make the model more robust and prevent overfitting.
  + **Synthetic Data Generation using LLMs:** Using pre-trained LLMs to generate artificial training data that mimics the characteristics of your desired data.
* **Challenges in Data Preparation for Fine-Tuning LLMs:** Preparing high-quality, relevant data can be challenging due to factors like data scarcity, noise, biases in the data, the need for accurate annotations, handling rare cases, and ethical considerations (avoiding harmful or biased content).
* **Available LLM Fine-Tuning Datasets:** The report mentions that there are existing datasets available for fine-tuning LLMs.
* **Best Practices:** These include collecting high-quality data, effective preprocessing, managing data imbalance, augmenting and annotating data carefully, ethical data handling (removing biases, ensuring privacy), and regular evaluation and iteration of your data preparation process.

**Chapter 4: Stage 2: Model Initialisation**

This chapter focuses on getting your initial LLM ready for fine-tuning. Figure 4.1 (page 22) shows the steps involved.

* **Steps Involved in Model Initialisation:**
  1. **Setup the Environment:** Installing necessary software libraries like **PyTorch** or **TensorFlow** and the **Hugging Face Transformers library**, which provides easy access to many pre-trained models.
  2. **Import Necessary Libraries:** Bringing the required tools into your code.
  3. **Choose the Language Model:** Selecting a pre-trained LLM from platforms like **Hugging Face's Model Hub** based on your task requirements. Examples include **BERT**, **GPT-3**, **LLaMA**, etc..
  4. **Download the Model from the Repository:** Using the chosen framework's functions to download the model's pre-trained weights and configuration. For example, using Transformers, you might use AutoModel.from\_pretrained('model\_name').
  5. **Load the Model in the Memory:** Making the downloaded model accessible in your computer's memory, ready for use in inference (generating text) or fine-tuning.
* **Tools and Libraries for Model Initialisation:** The **Hugging Face Transformers library** is highlighted as a key tool because it provides a unified interface for working with thousands of pre-trained models. It simplifies the process of downloading, loading, and using these models.
* **Challenges in Model Initialisation:** These can include ensuring compatibility between the chosen model and your hardware/software, managing large model files, and understanding the specific requirements and capabilities of different pre-trained models.
* **Tutorials:** The report mentions that tutorials are available for model initialization.

**Chapter 5: Stage 3: Training Setup**

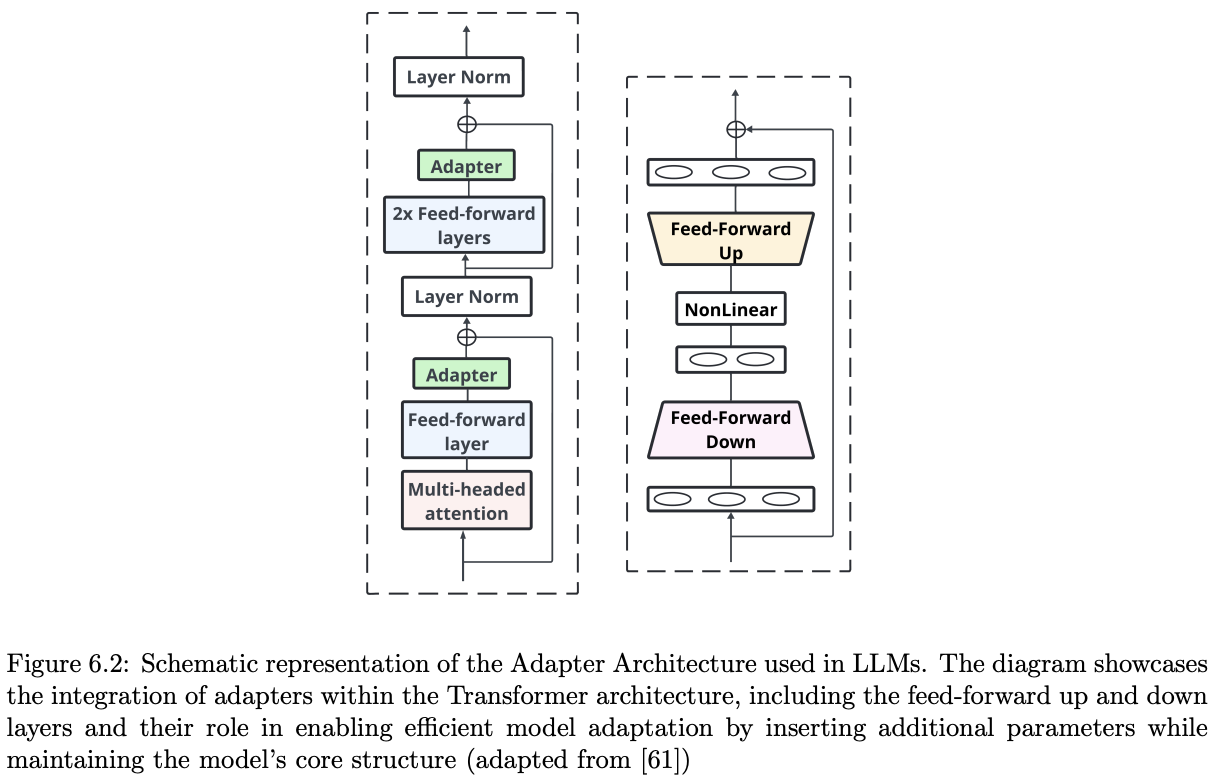
This chapter covers the preparation of the training environment and the configuration of the learning process.

* **Steps Involved in Training Setup:**
  1. **Setting up Training Environment:** This involves selecting and configuring the hardware (e.g., GPUs for faster training) and software environment (libraries, dependencies). Cloud platforms often provide pre-configured environments for machine learning.
  2. **Defining Hyperparameters:** These are settings that control how the model learns. Examples include:
     + **Learning Rate:** Determines how much the model's weights are adjusted during each training step. A too-high rate can lead to instability, while a too-low rate can make training very slow.
     + **Batch Size:** The number of training examples processed before the model's weights are updated. Larger batch sizes can sometimes lead to more stable training but require more memory.
     + **Epochs:** The number of times the entire training dataset is passed through the model during training. Training for too many epochs can lead to overfitting. Methods for **hyperparameter tuning** (finding the best values for these settings) are also mentioned.
  3. **Initialising Optimisers and Loss Functions:**
     + **Optimisers:** Algorithms that determine how the model's weights are updated based on the training data and the calculated errors. The report describes several optimizers, including:
       - **Gradient Descent:** A basic optimizer that updates weights in the direction that reduces the error. It can be slow on large datasets.
       - **Stochastic Gradient Descent (SGD):** Updates weights using only one randomly chosen training example at a time, making it faster but potentially more noisy.
       - **Mini-batch Gradient Descent:** A compromise between GD and SGD, using a small batch of examples for each update.
       - **AdaGrad, RMSprop, AdaDelta, Adam, AdamW:** More advanced optimizers that adapt the learning rate for each parameter during training, often leading to faster and more stable convergence. **Adam** and **AdamW** are particularly popular.
     + **Loss Functions:** Functions that measure the difference between the model's predictions and the actual target values in the training data. The goal of training is to minimize this loss. The specific loss function used depends on the task (e.g., cross-entropy loss for classification, mean squared error for regression).
* **Challenges in Training Setup:** These can include choosing the right hyperparameters, ensuring efficient use of computing resources, and dealing with potential instability during training.
* **Best Practices:** These include careful selection and tuning of hyperparameters, using appropriate optimizers and loss functions for the task, implementing techniques like **gradient clipping** (to prevent large gradient updates that can destabilize training), and maintaining thorough documentation of the training setup for reproducibility.

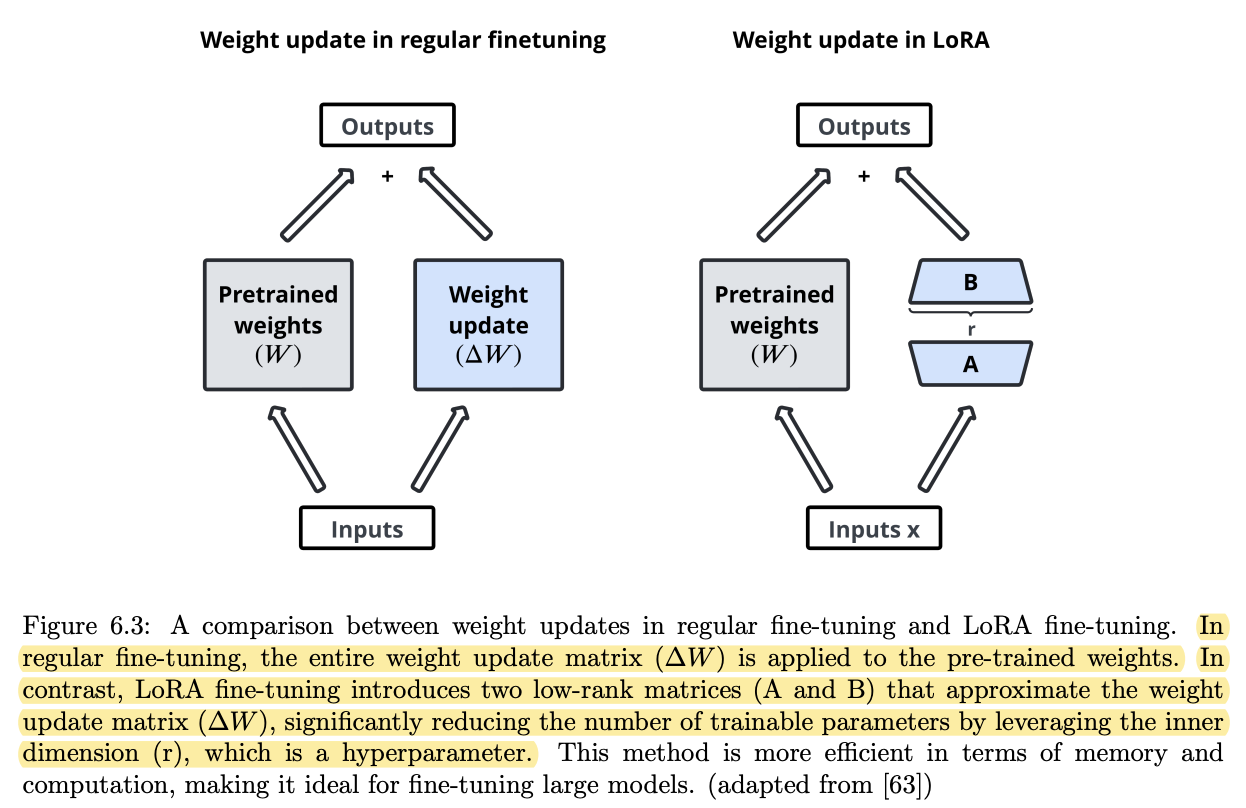
**Chapter 6: Stage 4: Selection of Fine-Tuning Techniques and Appropriate Model Configurations**

This chapter delves into the different strategies and methods you can use to actually fine-tune your LLM.

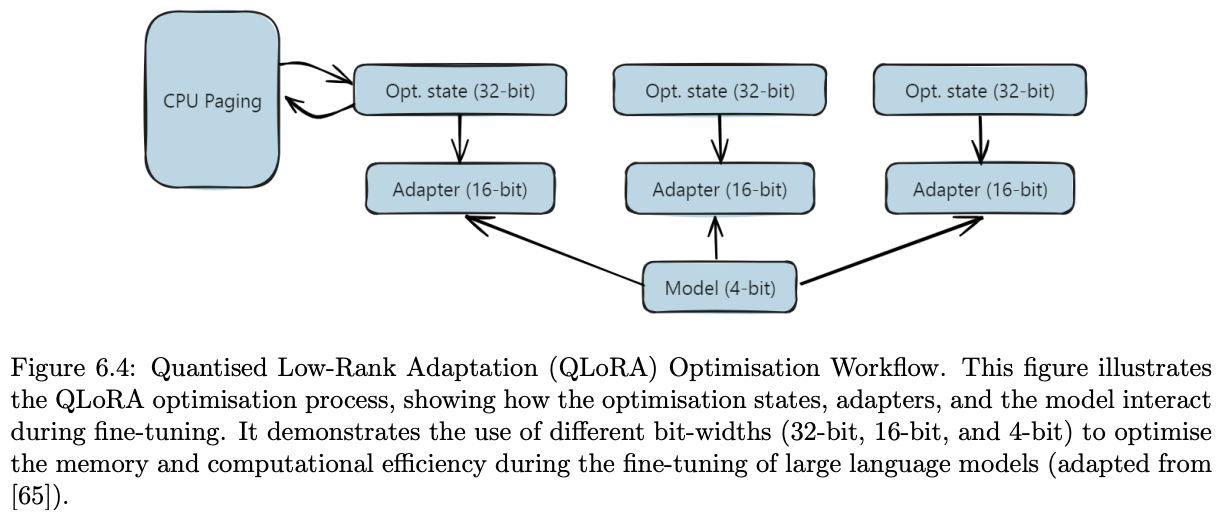
* **Steps Involved in Fine-Tuning:** This involves loading the pre-trained tokenizer (to convert text into numerical tokens the model can understand) and the pre-trained model itself. Then, you feed your prepared training data to the model and adjust its internal parameters based on the chosen fine-tuning technique.
* **Fine-Tuning Strategies for LLMs:**
  + **Task-Specific Fine-Tuning:** Tailoring the model for a particular task. Table 6.1 (page 35) provides examples like using **BERTSUM**, **GPT-3**, or **T5** for **text summarization**, **Codex** or **CodeBERT** for **code generation**, and **BERT**, **RoBERTa**, or **GPT-4** for **classification** or **question-answering**.
  + **Domain-Specific Fine-Tuning:** Adapting the model to understand and generate text specific to a certain domain or industry. The report gives examples like **LAWGPT** for the legal domain (fine-tuned on legal datasets like JEC-QA) and **PharmaGPT** for the pharmaceutical industry (fine-tuned on biopharmaceutical and chemical data).
* **Parameter-Efficient Fine-Tuning (PEFT) Techniques:** These methods aim to fine-tune LLMs by only modifying a small number of (or adding a small number of) parameters while keeping most of the pre-trained parameters frozen. This significantly reduces computational and storage costs and can help prevent catastrophic forgetting. Figure 6.1 (page 37) shows a taxonomy of PEFT methods.  
  + **Adapters:** Add small, new layers to the pre-trained model and only train these new layers. The original weights of the LLM remain unchanged.



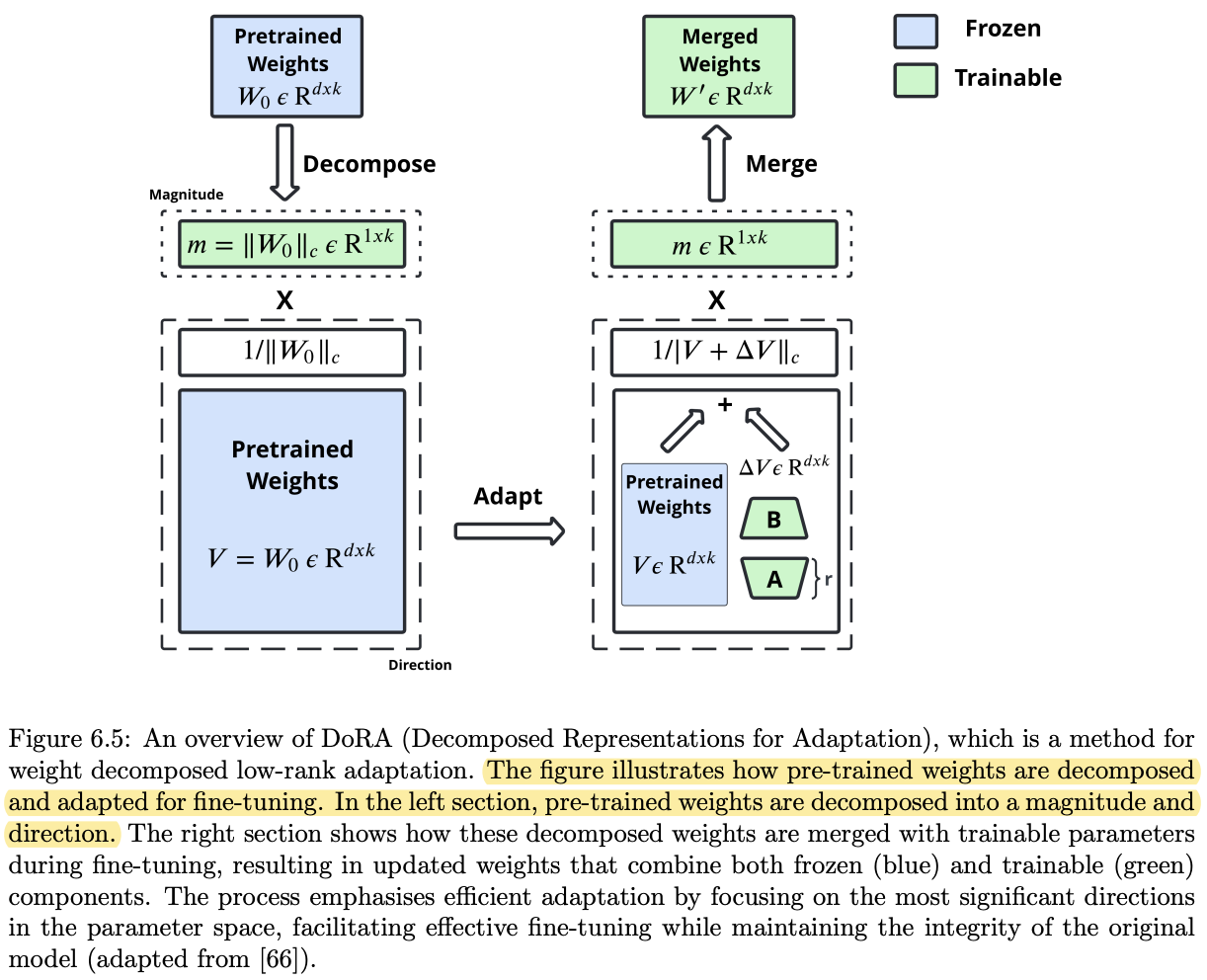
* + **Low-Rank Adaptation (LoRA):** Instead of directly training all the weights in certain layers, LoRA adds small, low-rank matrices to these layers and only trains these smaller matrices. This drastically reduces the number of trainable parameters. Table 6.2 (page 40) compares LoRA with another technique called DoRA. Tutorials for fine-tuning with LoRA are often available.



* + **QLoRA:** An extension of LoRA that further reduces memory usage by quantizing (reducing the precision of) the pre-trained model weights to 4 bits and then fine-tuning low-rank adapters. Figure 6.4 (page 40) illustrates the QLoRA workflow.



* + **Weight-Decomposed Low-Rank Adaptation (DoRA):** This method decomposes the weight updates in LoRA into magnitude and direction components, allowing for more effective learning.



* + **Fine-Tuning with Multiple Adapters:** You can train separate adapters for different tasks and then potentially combine them into a single multi-task adapter. Figure 6.6 (page 43) shows how this can work.
* **Half Fine Tuning (HFT):** This technique involves freezing half of the model's parameters in each layer during a fine-tuning round and updating the other half, and then alternating which half is frozen in subsequent rounds. Figure 6.7 (page 45) shows this process. Table 6.3 (page 46) compares HFT and LoRA.
* **Lamini Memory Tuning:** This is a specialized approach focused on reducing **hallucinations** (when the model generates incorrect or nonsensical information) in LLMs. It uses a **Massive Mixture of Memory Experts (MoME)** architecture where specific facts are stored in dynamically selected "expert" modules. Figure 6.8 (page 47) shows the Lamini-1 architecture.
* **Mixture of Experts (MoE):** This architecture uses multiple feedforward blocks (experts) in each layer, and for each input token, a router network selects two experts to process the information. **Mixtral 8x7B** is an example of an MoE model. Figure 6.9 (page 48) illustrates the MoE architecture. This allows the model to have a large capacity while only activating a small portion of its parameters for each input.
* **Mixture of Agents (MoA):** This is similar to MoE but operates at the model level rather than within the layers of a single model. It involves using multiple full-fledged LLMs (agents) that collaborate to produce an output. Figure 6.10 (page 49) illustrates this concept.
* **Proximal Policy Optimisation (PPO):** This is a **reinforcement learning** technique often used to fine-tune LLMs based on human preferences, particularly for tasks like improving the helpfulness and safety of model outputs. It's known for its stability and sample efficiency. Tutorials for training models with PPO are available.
* **Direct Preference Optimisation (DPO):** This is another preference-based learning method that aims to directly optimize the LLM's policy based on comparisons between preferred and less preferred outputs. Figure 6.12 (page 52) shows the DPO process. It's often seen as a simpler alternative to PPO for aligning LLMs. Tutorials for DPO training are also available.
* **Odds-Ratio Preference Optimization (ORPO):** This is yet another technique for aligning LLMs with desired preferences by optimizing an odds ratio between chosen and rejected responses.
* **Pruning LLMs:** This involves removing less important connections (weights) from a trained LLM to reduce its size and computational cost without significantly impacting performance. It's often done *after* fine-tuning.

**Chapter 7: Stage 5: Evaluation and Validation**

This chapter focuses on how to assess the performance of your fine-tuned LLM to ensure it's working as expected.

* **Steps Involved in Evaluating and Validating Fine-Tuned Models:** This includes running the fine-tuned model on a separate **evaluation dataset** (data it hasn't seen during training) and measuring its performance using relevant metrics. You also monitor the model's performance during training using the **validation set** to detect issues like overfitting.
* **Setting Up Evaluation Metrics:** The choice of metrics depends on the specific task. For example:  
  + For **text generation tasks** (like summarization or translation), metrics like **BLEU**, **ROUGE**, and **METEOR** are commonly used to compare the generated text to reference outputs.
  + For **classification tasks** (like sentiment analysis), metrics like **accuracy**, **precision**, **recall**, and **F1-score** are important.
  + In the context of **Retrieval-Augmented Generation (RAG)**, metrics like **Prompt Perplexity** (how well the model understands the input), **Context Relevance** (how relevant the retrieved information is), **Completeness** (how fully the response addresses the query), and **Chunk Attribution and Utilisation** (how effectively the retrieved chunks are used) are relevant.
* **Benchmark Datasets for LLM Evaluation:** The report lists numerous benchmark datasets used to evaluate different capabilities of language models. These datasets cover a wide range of tasks and skills, including:  
  + **HellaSwag:** Evaluates sentence completion.
  + **TruthfulQA:** Measures the truthfulness of responses.
  + **MMLU (Massive Multitask Language Understanding):** Tests the model's ability to perform tasks across diverse domains.
  + **IFEval (Instruction Following Evaluation):** Tests the ability to follow instructions.
  + **BBH (Big Bench Hard):** A set of challenging tasks for evaluating advanced reasoning.
  + **MATH:** Tests the ability to solve high-school math problems.
  + **GPQA (General-Purpose Question Answering):** A challenging knowledge dataset.
  + **MuSR (Multimodal Structured Reasoning):** Tests reasoning with long-range context parsing, often multimodal.
  + **MMLU-PRO:** A refined, more challenging version of MMLU.
  + **ARC (AI2 Reasoning Challenge):** Tests reasoning on science questions.
  + **COQA (Conversational Question Answering):** Evaluates understanding in conversational contexts.
  + **DROP (Discrete Reasoning Over Paragraphs):** Tests discrete reasoning over text.
  + **SQuAD (Stanford Question Answering Dataset):** A reading comprehension dataset.
  + **TREC:** A benchmark for text retrieval.
  + **WMT:** A dataset for machine translation evaluation.
  + **XNLI:** A dataset for cross-lingual language understanding.
  + **PiQA:** Evaluates understanding of physical interactions.
  + **Winogrande:** Tests commonsense reasoning. The choice of benchmark depends on the specific capabilities you want to evaluate.
* **Evaluating Fine-Tuned LLMs on Safety Benchmark:** Due to the potential for LLMs to generate harmful content, evaluating their safety is crucial. **Jailbreaking prompts** can sometimes bypass safety guidelines. **DecodingTrust** is mentioned as a framework for comprehensively evaluating the trustworthiness of LLMs, covering aspects like robustness, privacy, hallucination detection, tone appropriateness, ethics, and fairness.
* **Evaluating Safety of Fine-Tuned LLM using AI Models:** Several AI models are being developed to help evaluate and safeguard against unsafe content generation.  
  + **Llama Guard:** An LLM-based model for categorizing both input prompts and model responses based on a safety risk taxonomy (e.g., violence & hate, sexual content, illegal weapons). Llama Guard 2 and the newer Llama Guard 3 are specifically designed for this purpose.
  + **Shield Gemma:** Another generative AI content moderation model based on the Gemma family of models.
  + **WILDGUARD:** An open-source tool for detecting harmful intent in prompts, safety risks in responses, and appropriate refusals of unsafe requests. It was trained on the WILDGUARD MIX dataset and has shown strong performance compared to other open-source tools and even approaches GPT-4 in some aspects.