# Comprehensive Guide to Fine-Tuning Foundational LLMs for Enterprise Needs with Custom Layers and Data Pipelines

## Introduction

The advent of large language models (LLMs) like GPT-3, LLaMA, and Mistral represents a significant milestone in artificial intelligence, offering unprecedented capabilities in understanding and generating human-like text. Enterprises across various sectors are increasingly exploring these foundational models to enhance productivity, automate processes, and create innovative applications.[1] However, the general-purpose nature of these pre-trained models often limits their direct applicability to specific business contexts. They may lack the specialized jargon, nuanced understanding of internal processes, or adherence to strict compliance standards required for enterprise tasks.[2]

Fine-tuning emerges as the crucial process to bridge this gap, adapting these powerful but generic models into specialized tools tailored to unique enterprise needs.[1] Fine-tuning involves continuing the training process on smaller, domain-specific datasets, enabling the model to learn specific terminologies, adopt desired response styles, and align with business objectives.[2]

Despite its potential, fine-tuning LLMs presents significant challenges for enterprises, particularly concerning the substantial computational resources required, the associated costs, the need for high-quality curated data, ensuring data privacy and security, and managing the deployment and monitoring lifecycle (MLOps).[1] This guide provides a comprehensive, deeply technical, and step-by-step exploration of the LLM fine-tuning process specifically for enterprise applications. It covers the entire lifecycle, from understanding the fine-tuning landscape and preparing domain-specific data to modifying model architectures with custom layers, implementing efficient training workflows, evaluating performance rigorously, and deploying models scalably and securely within an enterprise environment.

## 1. Understanding the Fine-Tuning Landscape

Adapting foundational LLMs requires a clear understanding of the fine-tuning process, its variants, and the strategic considerations involved, especially within an enterprise context.

### 1.1. Defining Fine-Tuning

Fine-tuning is formally defined as the process of taking a pre-trained language model,

which has already learned general language patterns from vast amounts of text data, and continuing its training on a smaller, more specific dataset relevant to a particular task or domain.[1] Unlike the initial pre-training phase, which is typically unsupervised and computationally massive, fine-tuning is usually a supervised learning process (Supervised Fine-Tuning or SFT).[2] This means it utilizes labeled examples—such as prompt-response pairs for instruction following, text paired with sentiment labels for classification, or documents paired with summaries—to guide the model towards the desired behavior.[2]

The core purpose of fine-tuning is specialization.[2] It aims to:

- **Enhance Task Performance:** Improve accuracy and effectiveness on specific downstream tasks like sentiment analysis, question answering, or code generation within a particular domain.[2]
- **Instill Domain Knowledge:** Teach the model industry-specific jargon, company-specific terminology, entities, and contextual nuances absent in the general pre-training data.[1]
- **Align Model Behavior:** Tailor the model's output style, tone, and personality to match brand guidelines or specific interaction requirements (e.g., a professional legal assistant vs. a casual chatbot).[2] It can also be used to improve safety alignment, although this can be complex.[31]
- **Improve Data Privacy/Compliance:** By training on curated, potentially anonymized internal data within a controlled environment, fine-tuning can offer more security and compliance assurances compared to relying solely on external, general-purpose APIs, especially for sensitive data.[2]

Mechanically, fine-tuning involves feeding the task-specific labeled data to the model, calculating the difference (loss) between the model's predictions and the true labels, and using optimization algorithms like gradient descent to adjust the model's internal parameters (weights and biases) to minimize this loss.[2] This process nudges the model's learned representations towards the target task while leveraging the powerful foundation laid during pre-training—a concept known as transfer learning.[1] This contrasts sharply with prompting or in-context learning, which guides the model at inference time using examples in the input prompt but does not modify the model's weights.[1]

### 1.2. Types of Fine-Tuning

Several approaches exist for fine-tuning, differing primarily in the scope of parameter updates and architectural modifications.

- 1.2.1. Full Fine-Tuning:
  This is the traditional approach where all parameters (weights and biases) of the pre-trained model are updated during the fine-tuning process.4 It offers the highest potential for adapting the model deeply to the nuances of the target task or domain, as the entire model capacity is engaged in learning the new data patterns.4 However, full fine-tuning is extremely resource-intensive, demanding substantial GPU memory (often requiring multiple high-end GPUs for large models) and significant training time.9 Furthermore, storing the resulting model is costly, as each fully fine-tuned model is identical in size to the original large base model.17 A significant drawback is its susceptibility to catastrophic forgetting—the tendency for the model to lose some of the general knowledge acquired during pre-training as its weights are heavily modified for the specific fine-tuning task.10
- 1.2.2. Parameter-Efficient Fine-Tuning (PEFT):
  PEFT methods were developed specifically to overcome the limitations of full fine-tuning, particularly the prohibitive computational and storage costs associated with adapting massive LLMs.10 The central idea is to freeze the vast majority of the pre-trained model's parameters and update only a small fraction of them, or add a small number of new, trainable parameters (often called "adapters").10
  This approach yields substantial benefits crucial for enterprise adoption [35]:
  - **Reduced Costs:** Significantly lower computational requirements (memory, processing time) make fine-tuning feasible even on consumer-grade hardware or with limited cloud budgets.[10]
  - **Efficient Storage:** Fine-tuned parameters form small checkpoints (often just megabytes), drastically reducing storage needs compared to saving full model copies.[10]
  - **Portability & Modularity:** Small adapter checkpoints can be easily shared, stored, and loaded on top of a single base model, enabling efficient deployment of multiple task-specific adaptations.[37]
  - **Mitigated Forgetting:** By keeping most base model parameters frozen, PEFT methods generally preserve pre-trained knowledge better than full fine-tuning.[10]
  - **Comparable Performance:** Despite training far fewer parameters, PEFT often achieves performance comparable to, and sometimes better than (especially in low-data scenarios), full fine-tuning.[10]

  Several specific PEFT techniques exist:
  - **Adapter Tuning:** This involves inserting small, independent neural network modules ("adapters") into the layers of the pre-trained transformer.[41] These

adapters typically consist of a down-projection linear layer, a non-linearity, and an up-projection linear layer, forming a bottleneck architecture.[45] They are often added after the multi-head attention and feed-forward network sublayers within each transformer block, and include a residual connection so that the adapter can initially act as an identity function.[65] Only the parameters of these newly added adapters are trained.[10] Variants exist, such as placing adapters in series or parallel [13], and methods like AdapterFusion combine knowledge from multiple pre-trained adapters.[40] LLaMA-Adapter uses a specific zero-initialized attention gating mechanism to integrate adapter prompts.[67]

- **LoRA (Low-Rank Adaptation):** LoRA hypothesizes that the change in weights during adaptation ($\Delta W$) has a low intrinsic rank.[56] Instead of updating the original weight matrix W (which is kept frozen), LoRA introduces two smaller, trainable low-rank matrices, A and B, such that the update is represented by their product, BA. The modified forward pass becomes $h=W_0x+\Delta Wx=W_0x+BAx$.[56] Typically, LoRA is applied to the weight matrices within the self-attention mechanism (query, key, value, output projections).[56] The rank r of matrices A and B is a key hyperparameter, controlling the trade-off between expressiveness and parameter efficiency.[56] A scaling factor $\alpha$ (often related to r) is also used: $h=W_0x+r\alpha BAx$.[56] QLoRA combines LoRA with 4-bit quantization of the base model for further memory savings.[17]
- **Prefix Tuning:** This method keeps the entire pre-trained model frozen and instead prepends a small sequence of continuous, trainable vectors (the "prefix") to the input of each transformer layer.[14] These prefix vectors act as a task-specific "instruction" or context that steers the model's computation without altering its parameters.[82] Related approaches include Prompt Tuning (prefix only at the input embedding layer) [10] and P-Tuning (optimizing prompt embeddings, sometimes using an encoder like LSTM).[35]
- **BitFit (Bias Tuning):** An extremely sparse method where only the bias terms (vectors added after linear transformations) within the transformer model are fine-tuned, while all larger weight matrices remain frozen.[13] This affects a very small percentage of the total parameters (e.g., ~0.08-0.09%) but has shown surprising effectiveness in certain low-data or specific task scenarios.[89]

- 1.2.3. Task-Specific Layers / Heads:
  This approach involves adding one or more new layers, often called a "head," on top of the pre-trained base model.23 The base model, or at least its final layers, outputs contextualized representations (hidden states) for each input token. The task-specific head takes these representations (often just the representation of a special token like `` or the last token) and processes them to produce an output

suitable for the specific task, such as classification labels, regression values, or token-level tags.23 Typically, the base model's parameters are frozen, and only the parameters of the newly added head are trained, making it a form of parameter-efficient adaptation.25 However, it can also be combined with full or PEFT fine-tuning of the base model layers. This method is primarily used when the target task is not sequence generation, for example:

- **Classification:** A linear layer maps the final hidden state of a representative token (e.g., `` or last token) to the number of output classes.[23]
- **Extractive Question Answering:** Two linear layers predict the start and end token positions of the answer span within the context.[103]
- **Sequence Labeling (NER):** A linear layer maps each token's hidden state to the probability distribution over NER tags.[108]

### 1.3. When to Use Each Method

The choice of fine-tuning strategy depends heavily on the specific task, available resources (compute, data, time), performance requirements, and deployment constraints.

- **Full Fine-Tuning:** Best suited for scenarios with ample computational resources (multiple high-end GPUs, significant budget) where achieving the absolute maximum performance on a single, critical task is the priority, and potential loss of general knowledge (catastrophic forgetting) is either acceptable or can be mitigated.[9] Due to its cost, it's less common for adapting the largest foundation models in typical enterprise settings.
- **PEFT (General):** The preferred approach in most enterprise scenarios involving large models due to resource constraints.[10] It's particularly advantageous when:
  - Computational resources (GPU memory, time) are limited.
  - Storage costs need to be minimized, especially when deploying multiple task-specific models.
  - The fine-tuning dataset is relatively small (PEFT can be more robust to overfitting).
  - Preserving the base model's general capabilities is important (mitigating catastrophic forgetting).
  - Portability and easy swapping of task adaptations are desired.
  - *LoRA* is often the default starting point for PEFT due to its strong performance, efficiency, widespread support, and lack of inference latency when merged.[14]
  - *Adapters* offer modularity, making them suitable for multi-task scenarios where different adapters can be activated [40], though they might introduce

slight inference latency.[58]
    - *Prefix/Prompt Tuning* are extremely parameter-light and can be effective if the task can be well-defined through instructions or context modification.[10]
    - *BitFit* is an option for extreme parameter efficiency, potentially useful for minimal adaptation or very scarce data.[90]
- **Task-Specific Heads:** Necessary when the output format is fundamentally different from text generation, such as classification, regression, or token-level tagging.[23] This is often combined with either freezing the base model or applying PEFT/full fine-tuning to the base model layers as well.
- **RAG vs. Fine-tuning:** This is a crucial strategic decision.
    - Use **RAG** when the primary need is to incorporate external, dynamic, or proprietary factual knowledge into the model's responses at inference time, especially if this knowledge changes frequently.[8] RAG is also advantageous when source attribution or verifiability is required, and it can be more secure for sensitive data as the data isn't directly embedded in model weights.[12] It often has lower upfront training costs but potentially higher inference latency and cost due to the retrieval step.[12]
    - Use **Fine-tuning** when the goal is to teach the model a new *skill*, *style*, *format*, or deeply embed domain-specific reasoning patterns and terminology that go beyond simple fact retrieval.[8] It's better for adapting the model's inherent behavior. Fine-tuning generally results in lower inference latency once deployed.[129]
    - **Combination:** RAG and fine-tuning are not mutually exclusive. A powerful approach involves fine-tuning an LLM (often using PEFT) to better understand the domain, follow instructions specific to the RAG task (e.g., how to synthesize answers from retrieved documents, how to handle conflicting information, or ignore irrelevant retrieved documents as in RAFT [132]), or adopt a specific persona/style, and *then* using RAG at inference time to provide the up-to-date factual context.[12]

### 1.4. Enterprise Considerations

Several factors unique to the enterprise environment heavily influence the choice and implementation of fine-tuning strategies:

- **Cost:** Full fine-tuning of large models is often prohibitively expensive for many enterprises.[5] PEFT methods offer a much more cost-effective alternative by drastically reducing compute and storage requirements.[10] RAG shifts costs from upfront training to ongoing inference and data management.[12] Total cost of ownership (TCO) needs careful consideration.[21]

- **Security & Compliance:** Handling proprietary or sensitive customer data (e.g., financial, healthcare) is paramount.[2] Fine-tuning models on-premise or within a secure private cloud environment provides greater data control than using external APIs.[6] PEFT might inherently carry lower privacy risks than full fine-tuning due to fewer parameter changes [38], but the process of fine-tuning itself, regardless of the method, can compromise the model's original safety alignments, potentially making it easier to elicit harmful or biased outputs (jailbreaking).[31] RAG can be architected to keep sensitive data isolated in secure databases.[12] Adherence to regulations like HIPAA and GDPR necessitates robust data anonymization, access controls, and potentially specific infrastructure choices.[13]
- **Resources & Expertise:** Full fine-tuning demands significant GPU infrastructure and deep ML expertise.[9] PEFT lowers the hardware barrier [17] but still requires skilled ML practitioners to select methods, tune hyperparameters, and manage the process.[18] RAG requires expertise in information retrieval, vector databases, and potentially complex pipeline orchestration.[12]
- **Scalability & Deployment:** The chosen fine-tuning method impacts deployment. PEFT allows multiple lightweight adapters to share a single deployed base model, improving resource utilization and simplifying management.[18] Fully fine-tuned models require separate deployments for each task. Inference serving infrastructure must be designed for scalability regardless (discussed further in Section 7).

The adoption of PEFT is largely driven by the practical need to make LLM adaptation feasible within enterprise budget and resource limits. While PEFT significantly reduces computational demands and storage costs, it introduces its own nuances. For instance, the observation that LoRA can lead to different internal model structures ("intruder dimensions") and forgetting patterns compared to full fine-tuning, even when achieving similar task accuracy, underscores the need for careful evaluation beyond the primary fine-tuning objective.[33] This implies that PEFT methods are not just computationally cheaper versions of full fine-tuning but may represent fundamentally different adaptation pathways.

Furthermore, the strategic decision between embedding knowledge via fine-tuning versus retrieving it via RAG is becoming central. Fine-tuning excels at teaching skills, styles, and deeply ingrained domain logic, while RAG is superior for accessing volatile factual information or ensuring source traceability.[8] The optimal enterprise strategy often involves a hybrid approach, leveraging fine-tuning (likely PEFT) to adapt the model's core behavior and RAG to provide real-time, verifiable context.[12]

## Table 1: Comparison of LLM Fine-Tuning Approaches

| Approach | Parameters Updated | Typical Parameter Count | Compute Cost | Storage Cost | Inference Latency Impact | Catastrophic Forgetting Risk | Key Use Case/Strength |
|---|---|---|---|---|---|---|---|
| Full Fine-Tuning | All | 100% | High | High | None | High | Maximum task adaptation, high-resource settings |
| Adapter Tuning | Small Adapters | < 5% (often < 1%) | Low-Medium | Low | Minimal/ Potential | Medium-Low | Modularity, multi-task adaptation |
| LoRA (Low-Rank Adaptation) | Low-Rank Matrices (ΔW) | < 1% (often < 0.2%) | Low | Very Low | None (if merged) | Medium-Low | General PEFT, efficiency, performance balance, portability |
| Prefix Tuning | Continuous Prefixes | < 0.1% | Very Low | Very Low | Minimal | Low | Steering model via context, very parameter-light |
| BitFit (Bias Tuning) | Bias Terms Only | < 0.1% | Very Low | Very Low | None | Low | Extreme parameter efficienc |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | y, minimal adaptation |
| Task-Specific Head | New Head Layers Only | < 0.1% | Very Low | Very Low | Minimal | Low (for base model) | Non-generative tasks (classification, NER, QA) |

---

## 2. Data Preparation for Fine-Tuning

The success of any fine-tuning effort hinges critically on the quality, relevance, and format of the data used. This section details the process of sourcing, preparing, and formatting datasets for enterprise LLM fine-tuning.

### 2.1. Importance of High-Quality Data

It cannot be overstated: the dataset is the foundation upon which the fine-tuned model's performance is built.[3] Using low-quality, noisy, irrelevant, or biased data will invariably lead to suboptimal or even harmful model behavior.[29] Key considerations include:

- **Relevance:** Data must directly pertain to the target domain and specific task the model is being fine-tuned for.
- **Quality:** Data should be clean, consistent, and accurate. Errors or noise in the training data will be learned by the model.
- **Diversity:** The dataset should cover the expected range of inputs, scenarios, and linguistic variations the model will encounter in production to ensure generalization. Lack of diversity can lead to overfitting or biased performance.[150]
- **Size:** While more data is often better, quality frequently trumps quantity, especially in the initial stages.[3] Starting with a smaller, high-quality dataset (e.g., a few hundred to a few thousand examples) for initial fine-tuning and evaluation is often a pragmatic approach before scaling up.[149]

### 2.2. Data Sourcing Strategies for Enterprises

Enterprises possess unique data assets that can be leveraged for fine-tuning,

alongside external resources.

- **Internal Data:** This is often the most valuable source for domain specialization. Examples include:
  - Customer interactions: Support tickets, chat logs, emails, call transcripts.[2]
  - Documentation: Internal wikis, product manuals, technical specifications, process documents.[2]
  - Domain-specific content: Legal contracts, financial reports (e.g., SEC filings), clinical notes, research papers, code repositories.[2]
- **Public Datasets:** Numerous domain-specific datasets are publicly available (e.g., PubMed Central for biomedical text, financial news archives, legal databases like CaseLaw). These can supplement internal data or be used when internal data is scarce.
- **Web Scraping:** Public websites can be scraped for relevant domain information (e.g., industry news, forums, competitor websites). However, this requires adherence to ethical guidelines, robots.txt protocols, and website terms of service.[154] Techniques may be needed to handle JavaScript-heavy sites, and LLMs themselves can sometimes assist in structuring extracted data.[154]
- **Synthetic Data Generation:** When high-quality labeled data is limited, larger LLMs (like GPT-4 or Claude 3) can be prompted to generate synthetic examples (e.g., instruction-following pairs, question-answers based on provided documents).[5] This requires careful prompt engineering and rigorous quality control by domain experts to ensure the synthetic data is realistic, diverse, and doesn't introduce biases.[6]

### 2.3. Data Formats for Fine-Tuning Tasks

The structure of the fine-tuning data must align with the task and the tools being used (e.g., Hugging Face Trainer, OpenAI API).

- **General File Formats:**
  - **JSON Lines (JSONL):** Each line is a valid JSON object. Highly flexible, supports nested structures, streams well, and is commonly used in Hugging Face datasets and by OpenAI.[76] Preferred format for complex data.
  - **CSV/TSV:** Suitable for simple, flat structured data like text-label pairs.[76] Less ideal for nested or multi-turn data.
  - **Parquet:** Efficient columnar binary format, excellent for very large datasets, especially in cloud storage environments.[157]
  - **Raw Text (.txt):** Used primarily for continued pre-training on large unstructured corpora.[161]
- **Task-Specific Structures (often within JSONL):**

- **Instruction Tuning:** Requires examples demonstrating how the model should respond to instructions. Typically includes fields like instruction, input (optional context), and output or completion.[2] Separators (e.g., ### Instruction:) might be included in the formatted string.[163]
  - *Example:* {"instruction": "Translate to French", "input": "Hello", "output": "Bonjour"}
- **Dialogue Tuning (Chatbots):** Needs to represent multi-turn conversations.
  - *ShareGPT format:* A list under a key like "conversations", containing dictionaries with "from": "human"/"gpt" (or "user"/"assistant") and "value": "message text".[159]
  - *OpenAI/ChatML format:* A list under a key like "messages", containing dictionaries with "role": "system"/"user"/"assistant" and "content": "message text".[30] Special tokens or specific chat templates might be required by the model/tokenizer.[17]
  - *Example (ShareGPT):* {"conversations":}
- **Text Classification:** Usually pairs of text and a corresponding label.[27]
  - *Example (JSONL):* {"text": "Service was excellent!", "label": "positive"}
- **Named Entity Recognition (NER):** Requires token-level annotations.
  - *CoNLL format:* Word-per-line with its tag (e.g., EU B-ORG).[183] Often uses IOB/BIO tagging (Inside, Outside, Beginning of entity).
  - *Instruction format:* Frame NER as a generation task, asking the model to extract entities of specific types.[167]
  - *Example (Instruction):* {"instruction": "Find person names and organizations.", "input": "Alice works at Acme Corp.", "output": "Person: [\"Alice\"], Organization: [\"Acme Corp\"]"}
- **Extractive Question Answering (QA):** Needs context, question, and the exact answer span (text and start character index) within the context. The SQuAD dataset format is the standard.[27]
  - *Example (JSONL):* {"context": "The Eiffel Tower is in Paris.", "question": "Where is the Eiffel Tower?", "answers": {"text": ["Paris"], "answer_start": }}

### 2.4. Data Preprocessing Steps

Raw data is rarely suitable for direct use in fine-tuning. Rigorous preprocessing is essential.

- **Cleaning:** Remove irrelevant content like HTML tags, boilerplate text (e.g., standard disclaimers), excessive whitespace, or special characters that might interfere with tokenization or model understanding.[6] Correcting errors from sources like Optical Character Recognition (OCR) is also vital.[6]

- **Normalization:** Standardize text to ensure consistency. This may include converting text to lowercase (though sometimes case is important), normalizing Unicode characters, and standardizing formats for dates, currencies, or other structured elements within the text.[29]
- **Deduplication:** Identifying and removing duplicate or near-duplicate entries is crucial, especially with large datasets or web-scraped content.[29] Duplicates can bias the model towards common patterns and waste computational resources during training. Techniques often involve computing hash codes for text segments (e.g., paragraphs) and comparing them.[156]
- **PII Masking/Anonymization:** This is a non-negotiable step when dealing with enterprise data containing Personally Identifiable Information (PII) or Protected Health Information (PHI) to comply with regulations like GDPR and HIPAA.[13]
    - *Identification:* Detect sensitive information such as names, addresses, phone numbers, email addresses, social security numbers, credit card numbers, medical record numbers, etc..[150]
    - *Techniques:* Various methods exist, each with trade-offs between privacy protection and data utility:
        - Redaction: Complete removal of the PII.[150]
        - Replacement: Substituting PII with generic placeholders (e.g., <PERSON>, <EMAIL>) or synthetic but realistic data (using libraries like Faker).[2]
        - Masking: Replacing characters with a masking character (e.g., ***).[194]
        - Hashing/Encryption: Transforming PII into irreversible hashes or reversible encrypted values (requires key management).[194]
    - *Tools:* Libraries like Microsoft Presidio [196], pii-anonymizer [194], and pii-masker [195] leverage techniques like Named Entity Recognition (NER) and regular expressions to automate PII detection and apply chosen anonymization strategies.[194]
    - *Compliance Considerations:* For HIPAA, methods like Safe Harbor (removing 18 specific identifiers) or Expert Determination (statistical assessment of re-identification risk) must be considered.[138] Simply removing direct identifiers might be insufficient as models can sometimes re-identify individuals from combinations of quasi-identifiers.[139]
- **Formatting:** Convert the cleaned, processed data into the specific file format (JSONL, CSV, etc.) and structure (e.g., instruction/output pairs, chat messages) required by the chosen fine-tuning framework or script.[2]

### 2.5. Tokenization

Tokenization is the final preprocessing step, converting text into numerical IDs that the

LLM can ingest.

- **Importance:** LLMs operate on sequences of integers (token IDs), not raw text. Tokenization bridges this gap.[17]
- **Matching the Base Model's Tokenizer:** It is absolutely critical to use the *exact same tokenizer* that was used during the pre-training of the base LLM.[17] This includes the specific algorithm (e.g., BPE, SentencePiece), the vocabulary, merge rules, and handling of special tokens. Using a different tokenizer will result in a mismatch between the input token IDs and the model's learned embeddings, leading to poor performance or complete failure of the fine-tuning process.[206] Libraries like Hugging Face transformers (AutoTokenizer.from_pretrained) automatically download and configure the correct tokenizer associated with a given pre-trained model checkpoint.[17] Models like LLaMA and Mistral commonly use SentencePiece tokenizers.[206]
- **Handling Sequence Length Limits:** LLMs have a maximum context window size (e.g., 4096 tokens for Llama 2, 8192 or 32768 for GPT-4, potentially much larger for newer models like Llama 3.1 or specialized long-context models).[27] Input sequences exceeding this limit must be handled. Common strategies include:
    - **Truncation:** Cutting off tokens from the beginning or end of the sequence to fit the limit.[27] This is simple but risks losing important information.
    - **Padding:** Adding special padding tokens to shorter sequences in a batch so all sequences have the same length (either the maximum length in the batch or the model's maximum context length).[27] Attention masks are used to tell the model to ignore padding tokens.[96]
    - **Chunking:** Splitting long documents into smaller, overlapping chunks that fit within the context window.[122] This requires strategies for aggregating information across chunks during inference (e.g., map-reduce summarization [239]).
    - **Packing:** Combining multiple short examples into a single sequence, separated by EOS tokens, to fill the context window more efficiently and speed up training.[163]

The data preparation pipeline, encompassing sourcing, cleaning, PII handling, formatting, and tokenization, is often the most demanding yet crucial part of the fine-tuning process. It requires a blend of automated tools, careful manual review, and domain expertise to create datasets that effectively adapt LLMs for enterprise tasks while adhering to security and compliance standards. The necessity of robust PII masking, in particular, highlights a key difference between general-purpose LLM use and enterprise deployment, demanding specific tools and methodologies to balance data utility with privacy protection. Furthermore, the absolute requirement to use the

base model's specific tokenizer is a technical detail that, if overlooked, guarantees failure, emphasizing the need for careful environment and configuration management.

**Table 2: Data Formats for Common LLM Fine-Tuning Tasks**

| Task | Typical Format(s) | Key Data Fields | Example Snippet (JSONL) | Common Use Case |
|---|---|---|---|---|
| Instruction Tuning | JSONL | instruction, input (opt.), output | {"instruction": "Classify sentiment", "input": "Loved it!", "output": "positive"} | Task adaptation, chatbots |
| Dialogue/Chat | JSONL | messages ([role, content]) or conversations ([from, value]) | {"messages": [{"role": "user", "content": "Query?"}, {"role": "assistant", "content": "Answer."}]} | Customer service chatbots |
| Text Classification | JSONL, CSV | text, label | {"text": "...", "label": "category_A"} | Sentiment, topic analysis |
| NER (Token Class.) | CoNLL, JSONL | tokens, tags (IOB format) | {"tokens": ["Alice", "works", "at", "Acme"], "tags":} | Entity extraction |
| NER (Instruction) | JSONL | instruction, input, output (structured) | {"instruction": "Extract ORG", "input": "...", "output": "ORG: [\"Acme\"]"} | Entity extraction |
| QA (Extractive) | JSONL (SQuAD-like) | context, question, answers (text, | {"context": "...", "question": "...", "answers": {"text": | Document Q&A |

| | | | | |
|---|---|---|---|---|
| | | answer_start) | ["answer"], "answer_start": }} | |

---

### 3. Model Modification: Adding Custom Layers

While fine-tuning adjusts existing model parameters, certain tasks or efficiency techniques require modifying the model's architecture itself. This typically involves adding new layers, either as task-specific output heads or as part of Parameter-Efficient Fine-Tuning (PEFT) strategies.

#### 3.1. Rationale for Model Modification

Modifying the LLM architecture serves two primary purposes:

1. **Adapting Output Format:** Foundational LLMs are primarily generative, predicting the next token in a sequence. For tasks like classification, regression, or span prediction, the desired output is not free-form text but a specific structure (e.g., a class label, a numerical score, start/end indices). Adding custom "head" layers transforms the model's final internal representations into the required output format.[23]
2. **Implementing PEFT:** Many PEFT methods, such as Adapter Tuning and LoRA, work by introducing new, trainable parameters (adapter layers or low-rank matrices) into the existing model architecture while freezing the original weights.[10] This architectural modification is key to achieving parameter efficiency.

This contrasts with full fine-tuning, where the architecture often remains unchanged, but all existing weights are subject to updates. PEFT and custom heads involve targeted architectural additions or modifications, coupled with selective parameter training.

#### 3.2. Custom Task-Specific Heads

For tasks where the output is not a continuation of the input text, a custom head is typically added on top of the base transformer model.[25] The base model acts as a powerful feature extractor, providing rich contextual embeddings (usually the last_hidden_state output from the final transformer layer). These embeddings are then fed into the custom head, which performs the final task-specific computation.

- **Implementation:** Custom heads are typically implemented using standard neural network layers from frameworks like PyTorch (torch.nn.Module, torch.nn.Linear,

torch.nn.Dropout).[91] The head is appended to the base model, and often only the head's parameters (and perhaps the top few layers of the base model) are trained.

- **Classification Head:**
  - *Mechanism:* A common approach is to take the hidden state corresponding to a special token (like `` in BERT [91]) or the last token in the sequence for causal LMs, and pass it through a linear layer (nn.Linear) whose output dimension equals the number of classes.[23] An optional dropout layer can be added for regularization. The output logits are typically passed through a softmax function during inference to obtain class probabilities.
  - *Hugging Face:* The AutoModelForSequenceClassification class automatically handles adding a standard classification head to various base models.[91] When loading a pre-trained model using this class, the original pre-training head (if any) is discarded, and a new, randomly initialized classification head suitable for the specified number of labels is added.[92] Custom heads can also be defined and attached.[99]
- **Question Answering (Extractive) Head:**
  - *Mechanism:* This head predicts the start and end positions of the answer span within the provided context. It typically consists of two separate linear layers applied to the entire sequence of final hidden states from the base model. One linear layer outputs logits for each token being the *start* of the answer, and the second outputs logits for each token being the *end* of the answer.[123] Post-processing logic is needed during inference to select the most likely valid span.
  - *Hugging Face:* AutoModelForQuestionAnswering provides this functionality for compatible base models.[103]
- **Sequence Labeling (Token Classification / NER) Head:**
  - *Mechanism:* For tasks like Named Entity Recognition, where each token needs a label (e.g., B-PERSON, I-PERSON, O), a linear layer is applied independently to *each token's* final hidden state.[108] The output dimension of this linear layer corresponds to the number of possible NER tags. A softmax function can be applied per token during inference.
  - *Hugging Face:* AutoModelForTokenClassification adds such a token-level classification head.[99] It handles the application of the linear layer across the sequence dimension.

### 3.3. Adapter Layers (PEFT)

Adapters are small, plug-in modules added to a pre-trained model to enable

parameter-efficient fine-tuning.[41]

- **Mechanism:** They are typically inserted within each transformer block, often twice: once after the multi-head self-attention sublayer and once after the feed-forward network sublayer.[45] An adapter module usually consists of a down-projection linear layer (reducing dimensionality significantly, e.g., to a bottleneck size r), a non-linear activation function, and an up-projection linear layer (restoring the original dimensionality).[50] A residual connection is added around the adapter, allowing the network to bypass it initially or learn to blend its output with the original representation.[65]
- **Parameter Training:** During fine-tuning, the weights of the original transformer model are kept frozen, and only the parameters within the newly added adapter modules are trained.[10]
- **Initialization:** To ensure training stability and prevent disruption of the pre-trained model's capabilities at the start, adapters are often initialized such that they approximate an identity function.[65] LLaMA-Adapter employs a specific zero-initialized attention mechanism with a learnable gating factor to control the influence of adaptation prompts, preserving initial knowledge while progressively incorporating instructional signals.[67]
- **Libraries:** The Hugging Face PEFT library offers tools for implementing various adapter strategies.[37]

### 3.4. LoRA Layers (PEFT)

LoRA (Low-Rank Adaptation) is currently one of the most popular and effective PEFT techniques.[58]

- **Mechanism:** LoRA operates on the principle that the change in weights ($\Delta W$) needed to adapt a pre-trained model is often low-rank.[56] Instead of modifying the original weight matrix $W_0$ directly, LoRA introduces a parallel path that computes the update $\Delta W$ using the product of two smaller, low-rank matrices: $\Delta W = BA$, where A is a $d \times r$ matrix and B is an $r \times k$ matrix, with the rank $r$ being much smaller than the original dimensions $d$ and $k$.[56] The output of the modified layer becomes $h = W_0x + \Delta Wx = W_0x + BAx$. A scaling factor $\alpha/r$ is typically applied to the update term: $h = W_0x + r\alpha BAx$.[56] LoRA is most commonly applied to the query, key, value, and output projection matrices within the self-attention layers of transformers.[56]
- **Parameter Training:** Only the parameters of the low-rank matrices A and B are trained during fine-tuning; the original weight matrix $W_0$ remains frozen.[47]
- **Hyperparameters:** Key configuration choices include the rank $r$ (a smaller $r$ means fewer parameters but potentially less capacity; common values are 8, 16, 32, 64) [56], the scaling factor lora_alpha (often set to $2 \times r$) [56], the dropout rate

lora_dropout applied to the LoRA path [74], and target_modules specifying which layers (by name, e.g., "q_proj", "v_proj") should be adapted.[56]
- **Initialization:** The default initialization in libraries like PEFT often uses Kaiming uniform for matrix A and zeros for matrix B, ensuring the LoRA path initially adds zero to the output (identity transformation).[56] Other initializations exist, such as Gaussian [56] or more advanced methods like LoftQ (designed for QLoRA to minimize quantization error) [56] and PiSSA.[77]
- **Libraries:** The Hugging Face PEFT library provides LoraConfig to specify hyperparameters and the get_peft_model function to automatically modify a compatible transformers base model by injecting the LoRA layers.[37]

### 3.5. Tools for Implementation

Implementing these modifications is facilitated by several key libraries within the Python ecosystem:

- **Hugging Face transformers:** Provides the foundational pre-trained models (e.g., AutoModelForCausalLM, AutoModelForSequenceClassification) and their associated tokenizers. It serves as the base upon which modifications are applied.[27]
- **Hugging Face peft:** The primary library for applying various PEFT techniques like LoRA, Adapter Tuning, Prefix Tuning, etc., to transformers models. Its PeftConfig classes (e.g., LoraConfig) and get_peft_model function abstract away the complexities of architectural modification.[17]
- **Hugging Face trl:** (Transformer Reinforcement Learning) Primarily used for advanced fine-tuning techniques like Reinforcement Learning from Human Feedback (RLHF), but also provides the convenient SFTTrainer class for supervised fine-tuning, which integrates seamlessly with peft.[37]
- **PyTorch:** The underlying deep learning framework. Necessary for defining truly custom head architectures or modifications beyond what standard libraries provide.[91]

The decision to modify the model architecture depends directly on the target task and the chosen fine-tuning strategy. For classification or structured prediction, adding a custom head is standard practice. For adapting generative behavior or domain knowledge efficiently, PEFT methods like LoRA or Adapters, which involve injecting trainable parameters, are the go-to solutions in enterprise settings. LoRA, in particular, has gained prominence due to its compelling balance of parameter efficiency, strong empirical performance, and ease of integration without adding inference latency when merged. The Hugging Face PEFT library significantly simplifies

the implementation of these techniques, making sophisticated model adaptation more accessible.

---

### 4. Fine-Tuning Workflow (Code-Level Implementation)

This section provides a practical, code-level walkthrough of the fine-tuning process, focusing on using PEFT (specifically LoRA) with the Hugging Face ecosystem, a common scenario in enterprise settings due to resource constraints.

#### 4.1. Setting Up the Environment

A robust fine-tuning environment requires several key libraries and potentially specific hardware configurations.

- **Core Libraries:**
  - torch: The fundamental deep learning framework.[57]
  - transformers: For loading base models and tokenizers.[17]
  - datasets: For loading and preprocessing fine-tuning data.[17]
  - peft: For applying Parameter-Efficient Fine-Tuning methods like LoRA.[17]
  - accelerate: For simplifying distributed training (multi-GPU) and handling mixed precision.[17]
- **Efficiency Tools:**
  - bitsandbytes: Enables model quantization (e.g., loading models in 8-bit or 4-bit precision) to reduce memory footprint, crucial for fitting large models onto available hardware.[17] Often used for QLoRA.
- **Distributed Training Frameworks (Optional):**
  - For training very large models or large datasets across multiple GPUs/nodes, frameworks like DeepSpeed or PyTorch's Fully Sharded Data Parallel (FSDP) are used.[37]
  - Hugging Face accelerate provides a unified interface to configure and launch training using these backends via accelerate config and accelerate launch.[265]
- **Reinforcement Learning (Optional):**
  - The trl library facilitates advanced alignment techniques like Direct Preference Optimization (DPO) or Proximal Policy Optimization (PPO), often used after initial SFT.[37] It also provides the SFTTrainer used in this workflow example.
- **Hardware:**
  - GPUs are essential for practical fine-tuning. The required VRAM depends heavily on the base model size (e.g., 7B, 13B, 70B parameters), sequence length, batch size, and whether quantization/PEFT is used.[17] PEFT and 4-bit quantization (QLoRA) can make fine-tuning feasible on consumer GPUs (e.g.,

RTX 3090/4090) or single enterprise GPUs (e.g., A100/H100) that would be insufficient for full fine-tuning.[17]

### 4.2. Loading Model and Tokenizer

The first step in the code is to load the base model and its corresponding tokenizer.

Python

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

# Define the base model ID (e.g., a LLaMA or Mistral model)
model_id = "mistralai/Mistral-7B-Instruct-v0.1" # Or "meta-llama/Llama-2-7b-hf", etc.

# Configure quantization (optional, for efficiency)
# Use 4-bit quantization with NF4 type and double quantization
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16, # Use bfloat16 for compute
    bnb_4bit_use_double_quant=True,
)

# Load the base model
# device_map="auto" automatically distributes model layers across available GPUs
# quantization_config applies the 4-bit settings
base_model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto", # Automatically map model layers to available devices
    trust_remote_code=True, # Trust code execution from the model hub (use with caution)
    use_cache=False # Disable caching for training
)
# [49, 57, 78, 79, 212, 252]

# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
# Set pad token for causal LMs (often set to EOS token)
```

```
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right" # Important for causal LMs
# [17, 27, 57, 94, 163, 209, 212]

print(f"Model loaded on {base_model.device}")
print(f"Memory footprint: {base_model.get_memory_footprint() / 1e9:.2f} GB")
# [57]
```

### 4.3. Applying PEFT (Example: LoRA)

Next, configure and apply the chosen PEFT method. This example uses LoRA.

Python

```
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

# Prepare the quantized model for k-bit training (important for stability)
model = prepare_model_for_kbit_training(base_model)
# [30, 57]

# Define LoRA configuration
lora_config = LoraConfig(
    r=16,  # Rank of the update matrices. Higher rank means more parameters.
    lora_alpha=32, # Alpha parameter for scaling. Usually 2*r.
    target_modules=["q_proj", "v_proj"], # Target modules for LLaMA/Mistral
    lora_dropout=0.05, # Dropout probability for LoRA layers.
    bias="none", # Bias configuration. 'none' means bias terms are not trained.
    task_type="CAUSAL_LM" # Task type for Causal Language Modeling.
)
# [30, 54, 56, 57, 74, 75, 76, 209, 212]

# Apply LoRA configuration to the model
model = get_peft_model(model, lora_config)
# [37, 54, 56, 57, 74, 77, 160, 247, 248, 250]

# Print trainable parameters
model.print_trainable_parameters()
# Example output: trainable params: 4,718,592 |
| all params: 7,246,454,784 |
```

```
| trainable%: 0.06511
# [54]
```

This code snippet demonstrates loading a 7B parameter Mistral model with 4-bit quantization and applying LoRA adapters to the query (q_proj) and value (v_proj) projection layers within the attention mechanism. The print_trainable_parameters output confirms that only a tiny fraction (<< 1%) of the total parameters are actually being trained.

### 4.4. Preparing the Dataset

Load and format the dataset according to the task (e.g., instruction following).

Python

```
from datasets import load_dataset

# Load a sample dataset (e.g., instruction dataset)
dataset_name = "databricks/databricks-dolly-15k" # Example dataset
dataset = load_dataset(dataset_name, split="train")
# [27, 30, 49, 57, 75, 78, 92, 122, 123, 157, 161, 163, 171, 209, 212, 242, 264]

# Define a formatting function to structure the data as a prompt
# This depends heavily on the dataset structure and the desired input format
def formatting_prompts_func(example):
    output_texts =
    for i in range(len(example['instruction'])):
        text = f"### Instruction:\n{example['instruction'][i]}\n\n### Input:\n{example['context'][i]}\n\n### Response:\n{example['response'][i]}"
        output_texts.append(text)
    return output_texts
# [57, 159, 163, 209, 212, 242, 243]

# (Optional) Split dataset if no validation split exists
# dataset = dataset.train_test_split(test_size=0.1)
# train_dataset = dataset["train"]
# eval_dataset = dataset["test"]
# [2, 27, 92, 163, 209]

# Note: SFTTrainer handles tokenization implicitly when using formatting_func
```

```
# If using standard Trainer, explicit tokenization via.map() is needed here.
# tokenized_datasets = dataset.map(lambda examples: tokenizer(formatting_prompts_func(examples)), batched=True)
```

### 4.5. Training with Hugging Face Trainer / SFTTrainer

The Trainer (or the specialized SFTTrainer from trl) handles the training loop, optimization, evaluation, and checkpointing.

Python

```
from trl import SFTTrainer
from transformers import TrainingArguments

# Define Training Arguments
output_dir = "./results_mistral_lora"
training_args = TrainingArguments(
    output_dir=output_dir,
    per_device_train_batch_size=4, # Batch size per GPU
    gradient_accumulation_steps=4, # Effective batch size = 4 * 4 = 16
    learning_rate=2e-4,
    lr_scheduler_type="cosine", # Learning rate scheduler
    save_strategy="epoch", # Save checkpoints every epoch
    logging_steps=10, # Log training info every 10 steps
    num_train_epochs=1, # Number of training epochs
    max_steps=-1, # Set to positive value for step-based training instead of epoch-based
    fp16=True, # Enable mixed precision training (requires compatible GPU)
    # bf16=True, # Use bfloat16 for Ampere+ GPUs
    optim="paged_adamw_8bit", # Memory-efficient optimizer for quantized models
    # evaluation_strategy="steps", # Evaluate periodically
    # eval_steps=50, # Evaluate every 50 steps
    # save_total_limit=2, # Keep only the last 2 checkpoints
    # max_grad_norm=1.0, # Gradient clipping
    # warmup_ratio=0.03, # Warmup ratio for LR scheduler
    # report_to="wandb", # Optional: Log to Weights & Biases
    push_to_hub=False # Set to True to push model to Hugging Face Hub
)
# [49, 57, 75, 76, 79, 92, 97, 163, 209, 212, 243, 258, 259, 260, 274, 275, 276, 277, 278, 279]
```

```
# Initialize SFTTrainer
trainer = SFTTrainer(
    model=model, # The PEFT model
    args=training_args,
    train_dataset=dataset, # Use the full dataset or train_dataset split
    # eval_dataset=eval_dataset, # Optional: pass eval dataset
    formatting_func=formatting_prompts_func, # Function to format prompts
    max_seq_length=1024, # Maximum sequence length
    tokenizer=tokenizer,
    packing=True, # Pack multiple short examples into one sequence for efficiency
    # dataset_text_field="text" # Use if dataset has a single text column and no formatting_func
    # compute_metrics=compute_metrics_function, # Optional: function for evaluation metrics
)
# [27, 37, 49, 57, 75, 76, 79, 92, 97, 163, 209, 212, 242, 243, 244, 247, 256, 258, 264, 275]


# Start training
print("Starting training...")
trainer.train()
# [92, 97, 242, 243, 244, 256]

# Save the final adapter model
final_adapter_path = f"{output_dir}/final_adapter"
trainer.save_model(final_adapter_path)
print(f"Training finished. Final adapter saved to {final_adapter_path}")
# [37, 46, 50, 54, 57, 160, 248, 249, 250, 251, 252, 254]
```

- **Mixed Precision:** Using fp16=True or bf16=True significantly speeds up training and reduces memory usage on compatible GPUs (Volta+ for FP16, Ampere+ for BF16) by performing calculations in lower precision.[76]
- **Gradient Accumulation:** gradient_accumulation_steps allows simulating a larger batch size than fits in GPU memory by accumulating gradients over several smaller steps before performing an optimizer update.[243]
- **Gradient Clipping:** max_grad_norm helps stabilize training by preventing gradients from becoming excessively large.[258]
- **LR Scheduler & Warmup:** A learning rate scheduler (e.g., cosine, linear) gradually decreases the learning rate during training, often combined with an initial warmup phase where the LR increases from 0, which aids convergence and stability.[75]

### 4.6. Saving Checkpoints and Final Model

The Trainer automatically handles saving checkpoints based on the save_strategy and save_steps defined in TrainingArguments.[92]

- **Saving Adapters:** When using PEFT, calling trainer.save_model(output_path) or model.save_pretrained(output_path) saves only the trained adapter weights (e.g., adapter_model.safetensors) and the adapter configuration (adapter_config.json) to the specified directory.[37] This keeps checkpoints very small.
- **Loading Adapters:** To use the fine-tuned model for inference or to resume training, first load the original base model (potentially with the same quantization config), and then load the adapter weights onto it:
  Python
  ```
  from peft import PeftModel

  # Load base model (as before)
  base_model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_config, device_map="auto", trust_remote_code=True)
  tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)
  tokenizer.pad_token = tokenizer.eos_token
  tokenizer.padding_side = "right"

  # Load the PEFT adapter
  model = PeftModel.from_pretrained(base_model, final_adapter_path)
  model = model.eval() # Set to evaluation mode for inference
  # [46, 54, 160, 247, 248, 250, 252, 254]
  ```

- **Merging Adapters (Optional):** For deployment scenarios where the PEFT library might not be available or to potentially simplify the inference stack, LoRA adapters can be merged into the base model's weights. This creates a new standalone model with the fine-tuning baked in.
  Python
  ```
  # Merge the adapter weights into the base model
  merged_model = model.merge_and_unload()

  # Now `merged_model` is a standard transformers model (not PeftModel)
  # It can be saved like any regular transformers model
  merged_model_path = f"{output_dir}/final_merged_model"
  merged_model.save_pretrained(merged_model_path)
  tokenizer.save_pretrained(merged_model_path)
  # [56, 78, 249, 252, 253, 254, 280]
  ```

Merging eliminates the need for the peft library during inference but results in a full-size model checkpoint and loses the ability to easily swap different adapters onto the same base model.[18]

The Hugging Face ecosystem, particularly the combination of transformers, datasets, peft, accelerate, and trl, provides a cohesive and powerful toolkit that significantly simplifies the fine-tuning workflow. Abstracting complexities like distributed training, quantization integration, PEFT application, and the training loop allows developers to focus on dataset quality and hyperparameter tuning. Effective tuning of parameters like learning rate, batch size, scheduler, and LoRA-specific settings (r, alpha) remains crucial for achieving optimal results and stable training, especially within the resource constraints typical of enterprise environments. Understanding the checkpointing mechanism for PEFT—saving adapters separately unless explicitly merged—is vital for correct model persistence and deployment.

---

## 5. Evaluation and Metrics

Evaluating the performance of a fine-tuned LLM is crucial to understand its effectiveness, compare different approaches, and ensure it meets enterprise requirements before deployment. This involves using appropriate datasets and task-specific metrics.

### 5.1. Importance of Evaluation

Systematic evaluation serves several key purposes:

- **Performance Assessment:** Quantify how well the fine-tuned model performs on the specific task it was adapted for.[2]
- **Model/Strategy Comparison:** Objectively compare the results of different base models, fine-tuning techniques (e.g., LoRA vs. full fine-tuning), or hyperparameter settings.[281]
- **Debugging and Improvement:** Identify weaknesses, biases, or failure modes (e.g., overfitting, underfitting, hallucinations) to guide further refinement.[11]
- **Business Alignment:** Verify that the model meets the predefined success criteria and business objectives for the target application.[28]
- **Regression Testing:** Ensure that model updates or retraining cycles do not degrade performance on critical benchmarks.[158]

### 5.2. Validation vs. Test Sets

Proper dataset splitting is fundamental for reliable evaluation:

- **Training Set:** The data used to update the model's parameters during the fine-tuning process. The model learns directly from this data.
- **Validation Set (or Development Set):** A separate portion of the data, unseen during training updates, used *during* the fine-tuning process. Its primary roles are:
    - **Hyperparameter Tuning:** Selecting the best learning rate, batch size, LoRA rank, number of epochs, etc., based on performance on the validation set.[2]
    - **Model Selection:** Choosing the best performing model checkpoint (e.g., based on lowest validation loss or highest task metric).[2]
    - **Early Stopping:** Monitoring performance on the validation set to stop training when performance plateaus or starts to degrade, preventing overfitting to the training data.[26]
- **Test Set:** A final, held-out portion of the data that is used *only once* after all training and model selection are complete.[2] It provides an unbiased estimate of the final model's generalization performance on completely unseen data. Using the test set for tuning or model selection leads to inflated performance estimates.

Ensuring these sets are distinct and representative of the overall data distribution is crucial.[158] Data leakage between sets (e.g., having the same examples in train and test) invalidates the evaluation.

### 5.3. Task-Specific Metrics

The choice of evaluation metrics depends heavily on the fine-tuning task.

- **Classification Tasks (e.g., Sentiment Analysis, Intent Recognition):**
    - **Accuracy:** The proportion of correctly classified instances ($ (TP + TN) / (TP + TN + FP + FN) ).Simpleandintuitive,butcanbemisleadingonimbalanceddatasetswhereamodel mightachievehighaccuracybysimplypredictingthemajorityclass.[11,27,130,261,282,283,284,285,286,287,288]***Precision:**Theproportionofinstancespredictedaspositivethatareactuallypositive( TP / (TP + FP) ).Measuresthereliabilityofpositivepredictions.Highprecisionisimportantwhenfalsepositivesarecostly.[11,282,283,284,285,287,288,289,290,291]***Recall(Sensitivity,TruePositiveRate):**Theproportionofactualpositiveinstancesthatwerecorrectlyidentified( TP / (TP + FN) ).Measuresthemodel'sabilitytofindallpositiveinstances.Highrecalliscrucialwhenfalsenegativesarecostly(e.g.,diseasedetection).[11,282,283,284,285,287,288,289,290,291]***F1Score:**Theharmonicmeanofprecisionandrecall( 2 \times

(Precision \times Recall) / (Precision + Recall) $). Provides a single score that balances both metrics, particularly useful for imbalanced datasets.[11]
    - **AUC (Area Under the ROC Curve):** The area under the Receiver Operating Characteristic curve, which plots the True Positive Rate against the False Positive Rate at various classification thresholds. Measures the model's ability to discriminate between positive and negative classes across all thresholds. An AUC of 1.0 is perfect, 0.5 is random guessing.[283]
- **Question Answering (Extractive - SQuAD style):**
    - **Exact Match (EM):** The percentage of predictions where the predicted answer span exactly matches one of the ground truth answer spans (after normalization like removing punctuation and articles).[124] A strict metric.
    - **F1 Score:** Calculated at the token level. It measures the overlap between the predicted answer tokens and the ground truth answer tokens, considering both precision (tokens in prediction also in ground truth) and recall (tokens in ground truth also in prediction).[124] It's the harmonic mean of token-level precision and recall, providing a more lenient score than EM.
- **Text Generation (e.g., Summarization, Translation, Chat):** Evaluating generated text is challenging as there's often no single correct answer. Common metrics compare the generated text to one or more human-written reference texts:
    - **BLEU (Bilingual Evaluation Understudy):** Primarily used for machine translation. Measures precision of n-gram overlap (typically unigrams to 4-grams) between the generated text and references, with a brevity penalty for short outputs.[11] Higher is better.
    - **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** Commonly used for summarization. Measures recall of n-gram overlap (ROUGE-N) or longest common subsequence (ROUGE-L) between generated text and references.[11] Higher is better.
    - **METEOR (Metric for Evaluation of Translation with Explicit ORdering):** Considers precision and recall of unigrams, but also incorporates stemming and synonym matching, aiming for better correlation with human judgment than BLEU/ROUGE.[286]
    - **BERTScore:** Leverages contextual embeddings (from BERT or similar models) to compute semantic similarity between tokens in the generated text and references, calculating precision, recall, and F1 based on cosine similarity of embeddings.[291] Aims to capture meaning beyond exact word overlap.
    - **Perplexity (PPL):** Measures how well the language model predicts the test set. Lower perplexity indicates the model is less "surprised" by the sequence, suggesting better fluency and coherence.[11] It's calculated as the

exponentiated average negative log-likelihood of the sequence. While useful for evaluating language modeling capability, it doesn't directly measure task-specific quality like summarization accuracy or factual correctness.[298]

- **Human Evaluation:** Often considered the gold standard, especially for generative tasks. Involves humans rating model outputs based on criteria like fluency, coherence, relevance, factual accuracy, helpfulness, and harmlessness.[130] Can be expensive and time-consuming but provides invaluable qualitative insights. Pairwise comparisons (which response is better?) are common.[158]

### 5.4. Evaluation Frameworks and Libraries

Several frameworks and libraries facilitate standardized LLM evaluation:

- **Hugging Face evaluate:** A library providing easy access to implementations of many standard metrics (accuracy, F1, BLEU, ROUGE, BERTScore, seqeval for NER, etc.) and tools for comparison and measurement.[27] Integrates well with Trainer.
- **LM Evaluation Harness (lm-eval-harness):** An extensible framework by EleutherAI for evaluating LLMs on a wide range of benchmarks and tasks, supporting various model interfaces (Hugging Face, APIs) and evaluation modes (loglikelihood, generation).[305]
- **HELM (Holistic Evaluation of Language Models):** A comprehensive benchmark framework from Stanford CRFM aiming for broad coverage across many scenarios (e.g., accuracy, robustness, fairness, efficiency) and standardized evaluation protocols.[310]
- **Domain-Specific Benchmarks:** Many benchmarks exist for specific domains or capabilities (e.g., SuperGLUE for NLU, HumanEval for code generation, MMLU for multitask knowledge, BBQ for bias).[261]
- **scikit-learn:** Provides implementations for standard classification metrics (accuracy, precision, recall, F1, AUC) often used for evaluating classification heads.[282]

Rigorous evaluation using appropriate metrics on distinct validation and test sets is indispensable for developing reliable and effective fine-tuned LLMs for enterprise applications. Relying solely on training loss or simple metrics like accuracy can be misleading, especially for complex tasks or imbalanced data. Utilizing standardized frameworks can facilitate reproducible and comparable evaluations.

**Table 3: Common Evaluation Metrics for Fine-Tuning Tasks**

| Task | Metric(s) | Description | Interpretation |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Classification** | Accuracy | Overall percentage of correct predictions. | Higher is better (use cautiously with imbalance). |
| | Precision | Proportion of positive predictions that are correct (TP/(TP+FP)). | Higher is better (low false positives). |
| | Recall | Proportion of actual positives correctly identified (TP/(TP+FN)). | Higher is better (low false negatives). |
| | F1 Score | Harmonic mean of Precision and Recall. | Higher is better (balances P & R). |
| | AUC-ROC | Area under the True Positive Rate vs. False Positive Rate curve. | Higher is better (closer to 1.0). |
| **QA (Extractive)** | Exact Match (EM) | Percentage of predictions matching ground truth exactly. | Higher is better (strict). |
| | F1 Score (Token-level) | Harmonic mean of token precision and recall between prediction and ground truth. | Higher is better (more lenient). |
| **Text Generation** | BLEU | N-gram precision overlap with references (translation focus). | Higher is better. |
| | ROUGE (N, L) | N-gram/LCS recall overlap with references (summarization focus). | Higher is better. |
| | METEOR | Unigram P/R alignment with | Higher is better. |

| | | | |
|---|---|---|---|
| | | stemming/synonyms. | |
| | BERTScore | Semantic similarity using contextual embeddings (P, R, F1). | Higher is better. |
| **Language Modeling** | Perplexity (PPL) | Measure of how well the model predicts the test set (lower means less "surprised"). | Lower is better. |

---

## 6. Inference Optimization and Model Serving

After fine-tuning and evaluation, the next critical step is deploying the model for inference efficiently and reliably. This involves optimizing the model for speed and memory, choosing appropriate serving infrastructure, and ensuring scalability.

### 6.1. Saving and Loading Models for Inference

The method for saving the fine-tuned model depends on the technique used:

- **Full Fine-Tuning:** The entire model checkpoint (potentially tens or hundreds of GBs) is saved using standard methods like model.save_pretrained() or trainer.save_model(). Loading involves using AutoModelFor....from_pretrained() with the path to the saved checkpoint.
- **PEFT (Adapters/LoRA):**
  - **Saving Adapters Only:** The standard approach is to save only the small adapter weights and configuration using model.save_pretrained(adapter_path).[37] This results in very small artifact sizes (MBs).[37]
  - **Loading Adapters:** For inference, load the original base model first (potentially quantized), then load the adapter weights onto it using PeftModel.from_pretrained(base_model, adapter_path).[46]
  - **Merging Adapters:** Alternatively, LoRA adapters can be merged into the base model's weights using model.merge_and_unload().[56] The resulting merged_model is a standard transformers model that no longer requires the peft library for inference and can be saved using merged_model.save_pretrained().
  - **Adapter Only vs. Merged:** Saving only adapters offers significant storage

savings and flexibility (easily swap adapters for different tasks on the same base model).[18] Merging eliminates the peft dependency at inference and avoids the minor overhead of applying adapter weights separately, but results in a full-size model checkpoint and loses modularity.[50] The choice depends on deployment constraints and workflow.

### 6.2. Quantization for Inference

Quantization reduces the numerical precision of model weights (and sometimes activations) from standard 32-bit floating-point (FP32) or 16-bit (FP16/BF16) to lower bit formats like 8-bit integers (INT8) or 4-bit integers/floats (INT4/NF4).[17] This significantly reduces model size, memory bandwidth requirements, and can accelerate inference, especially on hardware with specialized low-precision compute units.[80]

- **Techniques:**
  - **Post-Training Quantization (PTQ):** Applied after the model is trained. Weights are converted to lower precision without retraining.[269] Simpler but may lead to some accuracy degradation.
    - *GPTQ (Generalized PTQ for LLMs):* A popular PTQ method that quantizes weights layer-by-layer, often to 4-bit, minimizing reconstruction error. Requires a calibration dataset.[269] Known for good performance, especially on GPUs.[269]
    - *AWQ (Activation-aware Weight Quantization):* Another PTQ method, similar to GPTQ, but protects salient weights based on activation magnitudes, potentially offering better performance or speed-ups.[269]
    - *bitsandbytes:** A library providing easy integration for 4-bit (NF4, FP4) and 8-bit quantization, often used directly during model loading in Hugging Face (load_in_4bit=True, load_in_8bit=True).[17] It performs quantization on-the-fly during loading and often dequantizes during computation.[269]
  - **Quantization-Aware Training (QAT):** Simulates quantization effects during the fine-tuning process itself, allowing the model to adapt to the lower precision. Generally yields better accuracy than PTQ but requires retraining.[273] QLoRA is a form of QAT where the base model is quantized (PTQ) but the LoRA adapters are trained with awareness of this quantization.[22]
- **Weight-Only Quantization:** A common strategy for LLM inference where only the large weight matrices are quantized (e.g., to INT4 or INT8), while activations are kept at higher precision (e.g., FP16).[80] This significantly reduces the memory footprint and bandwidth needed to load weights, which is often the bottleneck in LLM inference, especially with large batch sizes.[319] Weights are typically

dequantized back to FP16 just before computation.[80]
- **Impact:** Quantization offers a trade-off between efficiency (reduced size, faster inference, lower energy) and potential accuracy degradation.[269] 4-bit quantization often provides a good balance for many LLMs, but evaluation on the specific task is crucial.[269]

### 6.3. Optimized Serving Frameworks

Standard model loading and inference loops (e.g., basic Hugging Face generate) are often inefficient for production deployment. Specialized serving frameworks optimize throughput and latency.

- **vLLM:** An open-source library designed for fast LLM inference and serving.[322]
  - *Key Feature: PagedAttention:* An efficient memory management technique for the KV cache that avoids fragmentation and pre-allocation, allowing for higher batch sizes and longer sequences.[240]
  - *Continuous Batching:* Processes requests dynamically as they arrive and finish, rather than waiting for static batches, maximizing GPU utilization and throughput.[240]
  - *Other Features:* Supports tensor parallelism for multi-GPU inference, optimized CUDA kernels, compatibility with Hugging Face models, OpenAI-compatible API server, streaming outputs.[322] Often achieves significantly higher throughput compared to basic HF implementations.[323]
- **Text Generation Inference (TGI):** Hugging Face's production-ready solution for serving LLMs.[332]
  - *Features:* Tensor parallelism, continuous batching, optimized transformers code (FlashAttention, PagedAttention variants), quantization support (bitsandbytes, GPTQ), token streaming, watermarking, guidance (JSON/regex constraints), OpenAI-compatible API.[326] Written in Rust and Python for performance.[332] Powers Hugging Chat and other services.[332]
- **NVIDIA Triton Inference Server:** A general-purpose inference serving solution supporting multiple frameworks (TensorFlow, PyTorch, TensorRT, ONNX, Python backends) and hardware (GPUs, CPUs).[337]
  - *Features:* Concurrent model execution, dynamic batching, sequence batching for stateful models, model ensembles/pipelines, backend extensibility, metrics endpoint (Prometheus), integration with MLOps platforms.[338] Often used for enterprise deployments requiring support for diverse model types.
- **Other Options:** TensorRT-LLM (NVIDIA's library focused on optimizing LLMs for TensorRT) [316], DeepSpeed-Inference, custom solutions using FastAPI/Flask with optimized kernels.

### 6.4. Inference Optimizations

Beyond the serving framework, specific techniques further enhance inference performance:

- **Batching:** Processing multiple requests simultaneously to leverage GPU parallelism.
  - *Static Batching:* Group requests into fixed-size batches. Simple but can lead to idle time if batches don't fill quickly.
  - *Dynamic/Continuous Batching:* Adaptively group incoming requests or manage iterations to maximize GPU utilization and reduce latency, as implemented in vLLM and TGI.[240]
- **KV Cache Optimization:** The Key-Value (KV) cache stores attention keys and values for previously processed tokens, avoiding redundant computation during autoregressive generation.[327] This is crucial for performance but consumes significant GPU memory, especially for long sequences or large batches.[327]
  - *PagedAttention (vLLM):* Manages KV cache memory more efficiently using virtual memory concepts.[240]
  - *Quantization:* Applying quantization (e.g., INT8) specifically to the KV cache can reduce its memory footprint.[327]
  - *Sharing/

**Works cited**


1. Guide to Fine-Tuning LLMs: Definition, Benefits, and How-To - AIM Consulting, accessed May 5, 2025, https://aimconsulting.com/insights/guide-to-fine-tuning-llms-definition-benefits-and-how-to/
2. Fine-tuning large language models (LLMs) in 2025 - SuperAnnotate, accessed May 5, 2025, https://www.superannotate.com/blog/llm-fine-tuning
3. LLM Fine-Tuning: What It Is, Common Techniques, And More - Multimodal.dev, accessed May 5, 2025, https://www.multimodal.dev/post/llm-fine-tuning
4. What is Fine-Tuning LLM? Methods & Step-by-Step Guide in 2025 - Turing, accessed May 5, 2025, https://www.turing.com/resources/finetuning-large-language-models
5. Scalable and cost-effective fine-tuning for LLMs - Red Hat, accessed May 5, 2025, https://www.redhat.com/en/blog/how-to-achieve-scalable-cost-effective-fine-tuning-llm
6. Building Domain-Specific LLMs: A Comprehensive Guide for Enterprise Leaders - Arya.ai, accessed May 5, 2025, https://arya.ai/blog/building-domain-specific-llms-for-enterprise-leaders
7. Fine tuning LLMs for Enterprise: Practical Guidelines and Recommendations -

arXiv, accessed May 5, 2025, https://arxiv.org/html/2404.10779v1
8. RAG vs. fine-tuning: Choosing the right method for your LLM | SuperAnnotate, accessed May 5, 2025, https://www.superannotate.com/blog/rag-vs-fine-tuning
9. Large Language Models: To Fine-tune or not to Fine-tune? - ML6, accessed May 5, 2025, https://www.ml6.eu/blogpost/fine-tuning-large-language-models
10. Parameter-Efficient Fine-Tuning (PEFT): Optimizing LLMs - Kanerika, accessed May 5, 2025, https://kanerika.com/blogs/parameter-efficient-fine-tuning/
11. Pre-Training vs Fine Tuning: Choosing the Right Approach - Label Your Data, accessed May 5, 2025, https://labelyourdata.com/articles/llm-fine-tuning/pre-training-vs-fine-tuning
12. RAG Vs Fine Tuning: How To Choose The Right Method, accessed May 5, 2025, https://www.montecarlodata.com/blog-rag-vs-fine-tuning/
13. LLM Fine-Tuning: Use Cases, Best Practices, and Top 8 PEFT Methods - Kolena, accessed May 5, 2025, https://www.kolena.com/guides/llm-fine-tuning-use-cases-best-practices-and-top-8-peft-methods/
14. Comparing LLM fine-tuning methods - SignalFire, accessed May 5, 2025, https://www.signalfire.com/blog/comparing-llm-fine-tuning-methods
15. Compliance and AI: Finetuning LLMs for your Compliance Needs - Grand Blog, accessed May 5, 2025, https://blog.grand.io/content/files/2025/02/Compliance-and-AI--Finetuning-LLMs-for-your-Compliance-Needs-_removed.pdf
16. H2O.ai Launches Enterprise LLM Studio: Fine-Tuning-as-a-Service for Domain-Specific Models on Private Data - Business Wire, accessed May 5, 2025, https://www.businesswire.com/news/home/20250313244480/en/H2O.ai-Launches-Enterprise-LLM-Studio-Fine-Tuning-as-a-Service-for-Domain-Specific-Models-on-Private-Data
17. From Base to Instruct: Fine-tuning LLMs Using PEFT Techniques - Founding Minds, accessed May 5, 2025, https://www.foundingminds.com/from-base-to-instruct-fine-tuning-llms-using-peft-techniques/
18. Parameter-efficient Fine-tuning (PEFT): Overview, benefits, techniques and model training, accessed May 5, 2025, https://www.leewayhertz.com/parameter-efficient-fine-tuning/
19. Enterprise Guide: Choosing Between On-premise and Cloud LLM and Agentic AI Deployment Models - Allganize, accessed May 5, 2025, https://www.allganize.ai/en/blog/enterprise-guide-choosing-between-on-premise-and-cloud-llm-and-agentic-ai-deployment-models?source=Blog%20Post&medium=Linkedin%20&campaign=Blog_Post
20. Cloud vs. On-Prem LLMs: Strategic Considerations - Radicalbit MLOps Platform, accessed May 5, 2025, https://radicalbit.ai/resources/blog/cloud-onprem-llm/
21. Cloud vs. On-Premises: Choosing the Best Deployment Option for LLMs - MonsterAPI Blog, accessed May 5, 2025, https://blog.monsterapi.ai/cloud-vs-on-premises-hosting/
22. Efficient Fine-Tuning of LLMs: LoRA and QLoRA in Enterprise AI LangGraph

Workflows, accessed May 5, 2025, https://towardsai.net/p/artificial-intelligence/efficient-fine-tuning-of-llms-lora-and-qlora-in-enterprise-ai-langgraph-workflows
23. LLMs: Fine-tuning, distillation, and prompt engineering | Machine Learning, accessed May 5, 2025, https://developers.google.com/machine-learning/crash-course/llm/tuning
24. Fine-Tuning LLMs: Top 6 Methods, Challenges & Best Practices - Acorn Labs, accessed May 5, 2025, https://www.acorn.io/resources/learning-center/fine-tuning-llm/
25. What are common LLM fine-tuning techniques? - Deepchecks, accessed May 5, 2025, https://www.deepchecks.com/question/common-llm-fine-tuning-techniques/
26. The Ultimate Guide to LLM Fine Tuning: Best Practices & Tools - Lakera AI, accessed May 5, 2025, https://www.lakera.ai/blog/llm-fine-tuning-guide
27. Fine-Tuning LLMs: A Guide With Examples - DataCamp, accessed May 5, 2025, https://www.datacamp.com/tutorial/fine-tuning-large-language-models
28. llm-research-summaries/training/ultimate-guide-fine-tuning-llm_parthasarathy-2408.13296.md at main - GitHub, accessed May 5, 2025, https://github.com/cognitivetech/llm-research-summaries/blob/main/training/ultimate-guide-fine-tuning-llm_parthasarathy-2408.13296.md
29. How to Train an LLM: Complete Workflow Guide - Label Your Data, accessed May 5, 2025, https://labelyourdata.com/articles/how-to-train-an-llm/
30. Fine-Tuning Llama 2 for Advanced Chatbot Development | ml-articles - Wandb, accessed May 5, 2025, https://wandb.ai/mostafaibrahim17/ml-articles/reports/Fine-Tuning-Llama-2-for-Advanced-Chatbot-Development--Vmlldzo2NTY3ODUw
31. Fine-Tuning LLMs Breaks Their Safety and Security Alignment - Cisco Blogs, accessed May 5, 2025, https://blogs.cisco.com/security/fine-tuning-llms-breaks-their-safety-and-security-alignment
32. SLM vs LoRA LLM: Edge Deployment and Fine-Tuning Compared - Prem AI Blog, accessed May 5, 2025, https://blog.premai.io/slm-vs-lora-llm-edge-deployment-and-fine-tuning-compared/
33. LoRA Learns Less and Forgets Less - arXiv, accessed May 5, 2025, https://arxiv.org/html/2405.09673v2
34. Is LoRA Fine-Tuning Sometimes Less Effective Than Full Fine-Tuning of Smaller Models?, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1eg0cap/is_lora_finetuning_sometimes_less_effective_than/
35. What is parameter-efficient fine-tuning (PEFT)? - IBM, accessed May 5, 2025, https://www.ibm.com/think/topics/parameter-efficient-fine-tuning
36. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey - arXiv, accessed May 5, 2025, https://arxiv.org/pdf/2403.14608
37. Parameter-Efficient Fine-Tuning using PEFT - Hugging Face, accessed May 5,

2025, https://huggingface.co/blog/peft
38. Evaluating Privacy Risks of Parameter-Efficient Fine-Tuning | OpenReview, accessed May 5, 2025, https://openreview.net/forum?id=i2Ul8WIQm7
39. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment - arXiv, accessed May 5, 2025, https://arxiv.org/html/2312.12148v1
40. arXiv:2308.15982v1 [cs.CL] 30 Aug 2023, accessed May 5, 2025, https://arxiv.org/pdf/2308.15982
41. What is Parameter-Efficient Fine-Tuning (PEFT) of LLMs? - Hopsworks, accessed May 5, 2025, https://www.hopsworks.ai/dictionary/parameter-efficient-fine-tuning-of-llms
42. Parameter-efficient fine-tuning (PEFT): benefits and techniques - Software Mind, accessed May 5, 2025, https://softwaremind.com/blog/parameter-efficient-fine-tuning-peft-benefits-and-techniques/
43. LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models - arXiv, accessed May 5, 2025, https://arxiv.org/html/2309.12307v2
44. Fine-Tuning LLMs With Adapters - Restack, accessed May 5, 2025, https://www.restack.io/p/fine-tuning-answer-llms-adapters-cat-ai
45. PEFT for LLMs - ἐντελέχεια.άι, accessed May 5, 2025, https://lecture.jeju.ai/lectures/llms/peft/peft-llms.html
46. Load adapters with PEFT - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.43.2/peft
47. Fine-Tuning of Large Language Models with LoRA and QLoRA - Analytics Vidhya, accessed May 5, 2025, https://www.analyticsvidhya.com/blog/2023/08/lora-and-qlora/
48. PEFT - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/peft
49. Efficient Fine-Tuning with LoRA for LLMs | Databricks Blog, accessed May 5, 2025, https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms
50. Parameter-Efficient Fine-Tuning (PEFT) Basics & Tutorial - Acorn Labs, accessed May 5, 2025, https://www.acorn.io/resources/learning-center/parameter-efficient-fine-tuning-peft/
51. Towards Federated Low-Rank Adaptation of Language Models with Rank Heterogeneity - arXiv, accessed May 5, 2025, https://arxiv.org/html/2406.17477v2
52. Parameter-Efficient Fine-Tuning of Large Language Models via Deconvolution in Subspace, accessed May 5, 2025, https://arxiv.org/html/2503.01419v1
53. What is Alpaca-LoRA? Features & Getting Started - Deepchecks, accessed May 5, 2025, https://www.deepchecks.com/llm-tools/alpaca-lora/
54. huggingface/peft: PEFT: State-of-the-art Parameter-Efficient Fine-Tuning. - GitHub, accessed May 5, 2025, https://github.com/huggingface/peft
55. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey - arXiv, accessed May 5, 2025, https://arxiv.org/html/2403.14608v1
56. LoRA - Hugging Face, accessed May 5, 2025,

https://huggingface.co/docs/peft/main/conceptual_guides/lora
57. Fine-Tuning Your First Large Language Model (LLM) with PyTorch and Hugging Face, accessed May 5, 2025, https://huggingface.co/blog/dvgodoy/fine-tuning-llm-hugging-face
58. LoRA-Null: Low-Rank Adaptation via Null Space for Large Language Models - arXiv, accessed May 5, 2025, https://arxiv.org/html/2503.02659v1
59. LoRA vs Full Fine-tuning: An Illusion of Equivalence - arXiv, accessed May 5, 2025, https://arxiv.org/html/2410.21228v1
60. arxiv.org, accessed May 5, 2025, https://arxiv.org/html/2304.01933
61. LoRA vs Full Fine-tuning: An Illusion of Equivalence | AI Research Paper Details, accessed May 5, 2025, https://www.aimodels.fyi/papers/arxiv/lora-vs-full-fine-tuning-illusion-equivalence
62. LoRA vs Full Fine-tuning: An Illusion of Equivalence - ResearchGate, accessed May 5, 2025, https://www.researchgate.net/publication/385318527_LoRA_vs_Full_Fine-tuning_An_Illusion_of_Equivalence
63. [2106.09685] LoRA: Low-Rank Adaptation of Large Language Models - ar5iv - arXiv, accessed May 5, 2025, https://ar5iv.labs.arxiv.org/html/2106.09685
64. qazcdek/peft_cuda: PEFT: State-of-the-art Parameter-Efficient Fine-Tuning. - GitHub, accessed May 5, 2025, https://github.com/qazcdek/peft_cuda
65. Parameter-Efficient Transfer Learning for NLP - arXiv, accessed May 5, 2025, https://arxiv.org/pdf/1902.00751
66. THUDM/Awesome-Parameter-Efficient-Fine-Tuning-for-Foundation-Models - GitHub, accessed May 5, 2025, https://github.com/THUDM/Awesome-Parameter-Efficient-Fine-Tuning-for-Foundation-Models
67. LLaMA-Adapter: Efficient Fine-tuning of Large Language Models with Zero-initialized Attention - arXiv, accessed May 5, 2025, https://arxiv.org/html/2303.16199v3
68. llama-adapter: efficient fine-tuning of large language mod - arXiv, accessed May 5, 2025, http://arxiv.org/pdf/2303.16199
69. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention - arXiv, accessed May 5, 2025, https://arxiv.org/abs/2303.16199
70. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention - ar5iv, accessed May 5, 2025, https://ar5iv.labs.arxiv.org/html/2303.16199
71. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention, accessed May 5, 2025, https://www.researchgate.net/publication/369592739_LLaMA-Adapter_Efficient_Fine-tuning_of_Language_Models_with_Zero-init_Attention
72. Parameter-Efficient Fine-Tuning (PEFT): A Hands-On Guide with LoRA | Towards AI, accessed May 5, 2025, https://towardsai.net/p/machine-learning/parameter-efficient-fine-tuning-peft-a-hands-on-guide-with-lora
73. Fine-tuning Qwen2.5-VL for Document Information Extraction, accessed May 5,

2025, https://ubiai.tools/the-most-effective-techniques-for-applying-parameter-efficient-fine-tuning-peft/
74. LoRA - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/peft/package_reference/lora
75. Fine-Tuning Llama 3 with LoRA: Step-by-Step Guide - Neptune.ai, accessed May 5, 2025, https://neptune.ai/blog/fine-tuning-llama-3-with-lora
76. LLM Finetuning - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/autotrain/v0.8.24/tasks/llm_finetuning
77. LoRA - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/peft/main/developer_guides/lora
78. Fine-Tune Gemma using Hugging Face Transformers and QloRA | Google AI for Developers, accessed May 5, 2025, https://ai.google.dev/gemma/docs/core/huggingface_text_finetune_qlora
79. Efficiently fine-tune Llama 3 with PyTorch FSDP and Q-Lora - Philschmid, accessed May 5, 2025, https://www.philschmid.de/fsdp-qlora-llama3
80. A Guide to Quantization in LLMs | Symbl.ai, accessed May 5, 2025, https://symbl.ai/developers/blog/a-guide-to-quantization-in-llms/
81. AQLoRA: An Adaptive Quantization-Based Efficient Fine-Tuning Method for LLMs | Request PDF - ResearchGate, accessed May 5, 2025, https://www.researchgate.net/publication/385468492_AQLoRA_An_Adaptive_Quantization-Based_Efficient_Fine-Tuning_Method_for_LLMs
82. Exploring acquisition of Novel World Knowledge in LLMs Using Prefix-Tuning - arXiv, accessed May 5, 2025, https://arxiv.org/html/2408.17070v1
83. [2503.02875] The First Few Tokens Are All You Need: An Efficient and Effective Unsupervised Prefix Fine-Tuning Method for Reasoning Models - arXiv, accessed May 5, 2025, https://arxiv.org/abs/2503.02875
84. openreview.net, accessed May 5, 2025, https://openreview.net/pdf?id=GYOXIRXI7W
85. synchronized label tuning for prompt and prefix in llms - arXiv, accessed May 5, 2025, https://arxiv.org/pdf/2402.01643
86. Fine Tuning LLMs with Hugging Face - Pluralsight, accessed May 5, 2025, https://www.pluralsight.com/professional-services/data-machine-learning/-fine-tuning-llms-with-hugging-face
87. FINETUNED LANGUAGE MODELS ARE ZERO-SHOT LEARNERS - OpenReview, accessed May 5, 2025, https://openreview.net/pdf?id=gEZrGCozdqR
88. DiffoRA: Enabling Parameter-Efficient LLM Fine-Tuning via Differential Low-Rank Matrix Adaptation - arXiv, accessed May 5, 2025, https://arxiv.org/html/2502.08905v1
89. [2106.10199] BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models - arXiv, accessed May 5, 2025, https://arxiv.org/abs/2106.10199
90. arxiv.org, accessed May 5, 2025, https://arxiv.org/pdf/2106.10199
91. LLMs are machine learning classifiers | LLMs-as-classifiers ... - Wandb, accessed May 5, 2025,

https://wandb.ai/gladiator/LLMs-as-classifiers/reports/LLMs-are-machine-learning-classifiers--VmlldzoxMTEwNzUyNA
92. Fine-tuning a model with the Trainer API - Hugging Face LLM Course, accessed May 5, 2025, https://huggingface.co/learn/llm-course/chapter3/3
93. Fine-tune a pretrained model - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/main/en//training
94. Fine-tuning a pretrained model - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.14.1/training
95. Classification heads - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/setfit/how_to/classification_heads
96. Hugging Face Transformers: Fine-tuning DistilBERT for Binary Classification Tasks, accessed May 5, 2025, https://towardsdatascience.com/hugging-face-transformers-fine-tuning-distilbert-for-binary-classification-tasks-490f1d192379/
97. Fine-tuning - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/training
98. Customizing models - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/custom_models
99. How do I change the classification head of a model? - Transformers, accessed May 5, 2025, https://discuss.huggingface.co/t/how-do-i-change-the-classification-head-of-a-model/4720
100. How do I change the classification head of a model? - Page 2 - Transformers, accessed May 5, 2025, https://discuss.huggingface.co/t/how-do-i-change-the-classification-head-of-a-model/4720?page=2
101. Auto Classes - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/en/model_doc/auto
102. Add a classification head to a fine-tuned language model - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/add-a-classification-head-to-a-fine-tuned-language-model/8176
103. what is the difference between AutoModelForCausalLM and AutoModel? : r/huggingface, accessed May 5, 2025, https://www.reddit.com/r/huggingface/comments/1bv1kfk/what_is_the_difference_between/
104. Question Answering using Transformers Hugging Face Library || BERT QA Python Demo, accessed May 5, 2025, https://www.youtube.com/watch?v=DNRiUGtKlVU&pp=0gcJCdgAo7VqN5tD
105. An Introduction to Using Transformers and Hugging Face | DataCamp, accessed May 5, 2025, https://www.datacamp.com/tutorial/an-introduction-to-using-transformers-and-hugging-face
106. Auto Classes - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.14.1/model_doc/auto

107. AutoModels — transformers 3.0.2 documentation - Hugging Face, accessed May 5, 2025, https://huggingface.co/transformers/v3.0.2/model_doc/auto.html
108. Token classification - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/tasks/token_classification
109. What is the classification head of a hugging face AutoModelForTokenClassification Model, accessed May 5, 2025, https://stackoverflow.com/questions/75890430/what-is-the-classification-head-of-a-hugging-face-automodelfortokenclassificatio
110. What are differences between AutoModelForSequenceClassification vs AutoModel, accessed May 5, 2025, https://stackoverflow.com/questions/69907682/what-are-differences-between-automodelforsequenceclassification-vs-automodel
111. Token classification - Hugging Face NLP Course, accessed May 5, 2025, https://huggingface.co/learn/nlp-course/chapter7/2
112. How is the "Auto Model For Sequence Classification" architecture? - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/how-is-the-auto-model-for-sequence-classification-architecture/8440
113. TransformerEncoderLayer — PyTorch 2.7 documentation, accessed May 5, 2025, https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html
114. Transformer — PyTorch 2.7 documentation, accessed May 5, 2025, https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html
115. Complete Guide to Building a Transformer Model with PyTorch - DataCamp, accessed May 5, 2025, https://www.datacamp.com/tutorial/building-a-transformer-with-py-torch
116. Transformers always only use a single Linear layer for classification head? - Stack Overflow, accessed May 5, 2025, https://stackoverflow.com/questions/75548317/transformers-always-only-use-a-single-linear-layer-for-classification-head
117. Add layers on pretrained model - vision - PyTorch Forums, accessed May 5, 2025, https://discuss.pytorch.org/t/add-layers-on-pretrained-model/88760
118. transformer-heads/README.md at main - GitHub, accessed May 5, 2025, https://github.com/center-for-humans-and-machines/transformer-heads/blob/main/README.md
119. Models - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/trl/models
120. Models - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/main/main_classes/model
121. Models — transformers 3.2.0 documentation - Hugging Face, accessed May 5, 2025, https://huggingface.co/transformers/v3.2.0/main_classes/model.html
122. Fine-tuning a masked language model - Hugging Face LLM Course, accessed May 5, 2025, https://huggingface.co/learn/llm-course/chapter7/3
123. Question answering - Hugging Face LLM Course, accessed May 5, 2025,

https://huggingface.co/learn/llm-course/en/chapter7/7
124. Question Answering - Stanford University, accessed May 5, 2025, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6853494.pdf
125. Adapters - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/peft/conceptual_guides/adapter
126. Text Classification With LLMs: A Roundup of the Best Methods - Striveworks, accessed May 5, 2025, https://www.striveworks.com/blog/text-classification-with-llms-a-roundup-of-the-best-methods
127. 7 Things You Need to Know About Fine-tuning LLMs - Predibase, accessed May 5, 2025, https://predibase.com/blog/7-things-you-need-to-know-about-fine-tuning-llms
128. RAG vs Fine-Tuning: Choosing the Right Approach for Building LLM-Powered Chatbots, accessed May 5, 2025, https://www.techaheadcorp.com/blog/rag-vs-fine-tuning-difference-for-chatbots/
129. RAG vs. Fine-Tuning: How to Choose - Oracle, accessed May 5, 2025, https://www.oracle.com/artificial-intelligence/generative-ai/retrieval-augmented-generation-rag/rag-fine-tuning/
130. How RAG and Fine-Tuning Enhance LLM Performance: Case Studies - TiDB, accessed May 5, 2025, https://www.pingcap.com/article/how-rag-and-fine-tuning-enhance-llm-performance-case-studies/
131. How to Make Your LLM More Accurate with RAG & Fine-Tuning | Towards Data Science, accessed May 5, 2025, https://towardsdatascience.com/how-to-make-your-llm-more-accurate-with-rag-fine-tuning/
132. RAFT: Adapting Language Model to Domain Specific RAG - arXiv, accessed May 5, 2025, https://arxiv.org/html/2403.10131v1
133. [2403.10131] RAFT: Adapting Language Model to Domain Specific RAG - arXiv, accessed May 5, 2025, https://arxiv.org/abs/2403.10131
134. arXiv:2403.10131v2 [cs.CL] 5 Jun 2024, accessed May 5, 2025, https://arxiv.org/pdf/2403.10131
135. How to fine-tune LLMs for better RAG performance - TechTalks, accessed May 5, 2025, https://bdtechtalks.com/2024/03/25/raft-llm-fine-tuning-for-rag/
136. On Premise vs. Cloud: Key Differences, Benefits and Risks | Cleo, accessed May 5, 2025, https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud
137. PEFT-as-an-Attack! Jailbreaking Language Models during Federated Parameter-Efficient Fine-Tuning - arXiv, accessed May 5, 2025, https://arxiv.org/html/2411.19335v1
138. Synthesizing Healthcare Data for AI, With HIPAA Expert Determination | Tonic.ai, accessed May 5, 2025, https://www.tonic.ai/guides/hipaa-ai-compliance
139. LLM Tuning and Training in Healthcare and Research - - Fred Hutch SciWiki, accessed May 5, 2025, https://sciwiki.fredhutch.org/datascience/llm_tuning/

140. Towards a HIPAA Compliant Agentic AI System in Healthcare - arXiv, accessed May 5, 2025, https://arxiv.org/html/2504.17669v1
141. Development of secure infrastructure for advancing generative artificial intelligence research in healthcare at an academic medical center - PubMed Central, accessed May 5, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11833461/
142. The EDPB Opinion on training AI models using personal data and recent Garante fine – lawful deployment of LLMs, accessed May 5, 2025, https://www.dataprotectionreport.com/2025/01/the-edpb-opinion-on-training-ai-models-using-personal-data-and-recent-garante-fine-lawful-deployment-of-llms/
143. Large language models (LLM) | European Data Protection Supervisor, accessed May 5, 2025, https://www.edps.europa.eu/data-protection/technology-monitoring/techsonar/large-language-models-llm_en
144. LegiLM: A Fine-Tuned Legal Language Model for Data Compliance - arXiv, accessed May 5, 2025, https://arxiv.org/html/2409.13721v1
145. Hamburg DPA Launches GDPR Discussion Paper on Personal Data in LLMs, accessed May 5, 2025, https://blog.privacycraft.pro/hamburg-dpa-launches-gdpr-discussion-paper-on-personal-data-in-llms/
146. LoRA vs Full Fine-tuning: An Illusion of Equivalence | OpenReview, accessed May 5, 2025, https://openreview.net/forum?id=PGNdDfsl6C
147. llm-research-summaries/training/LoRA-vs-Full-Fine-tuning-Illusion-Equivalence_2410.21228v1.md at main - GitHub, accessed May 5, 2025, https://github.com/cognitivetech/llm-research-summaries/blob/main/training/LoRA-vs-Full-Fine-tuning-Illusion-Equivalence_2410.21228v1.md
148. BitNet a4.8, LoRA vs Full Fine-tuning, and Mixture of In-context Learner - The AI Timeline, accessed May 5, 2025, https://mail.bycloud.ai/p/bitnet-a4-8-lora-vs-full-fine-tuning-and-mixture-of-in-context-learner
149. Fine-Tuning Large Language Models for Graph RAG-5 Indexing - Dataworkz, accessed May 5, 2025, https://www.dataworkz.com/fine-tuning-a-llm-to-index-graph-rag-5/
150. Guide to Ethical Fine-Tuning of Large Language Models | Tonic.ai, accessed May 5, 2025, https://www.tonic.ai/guides/ethical-fine-tuning-llm-synthetic-data
151. Ethical Considerations in LLM Development - Gaper.io, accessed May 5, 2025, https://gaper.io/ethical-considerations-llm-development/
152. 6 Common LLM Fine-Tuning Mistakes Everyone Should Know - MonsterAPI Blog, accessed May 5, 2025, https://blog.monsterapi.ai/common-llm-fine-tuning-mistakes/
153. Prepare supervised fine-tuning data for Translation LLM models | Generative AI on Vertex AI, accessed May 5, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/models/translation-supervised-tuning-prepare

154. How LLM Web Scraping Transforms Data Extraction and Processing? - PromptCloud, accessed May 5, 2025, https://www.promptcloud.com/blog/llm-web-scraping-for-data-extraction/
155. How to do Web Scraping with LLMs for Your Next AI Project? - ProjectPro, accessed May 5, 2025, https://www.projectpro.io/article/web-scraping-with-llms/1081
156. An introduction to preparing your own dataset for LLM training - AWS - Amazon.com, accessed May 5, 2025, https://aws.amazon.com/blogs/machine-learning/an-introduction-to-preparing-your-own-dataset-for-llm-training/
157. Step-by-Step Guide on Building a Dataset for LLM Fine-tuning - MonsterAPI Blog, accessed May 5, 2025, https://blog.monsterapi.ai/how-to-build-a-dataset-for-llm-fine-tuning/
158. LLM Evaluation: Benchmarks to Test Model Quality - Label Your Data, accessed May 5, 2025, https://labelyourdata.com/articles/llm-fine-tuning/llm-evaluation
159. Datasets Guide - Unsloth Documentation, accessed May 5, 2025, https://docs.unsloth.ai/basics/datasets-guide
160. tloen/alpaca-lora: Instruct-tune LLaMA on consumer hardware - GitHub, accessed May 5, 2025, https://github.com/tloen/alpaca-lora
161. LLM Dataset Formats 101: A No-BS Guide for Hugging Face Devs, accessed May 5, 2025, https://huggingface.co/blog/tegridydev/llm-dataset-formats-101-hugging-face
162. How to Use JSON for Fine-Tuning Machine Learning Models - DigitalOcean, accessed May 5, 2025, https://www.digitalocean.com/community/tutorials/json-for-finetuning-machine-learning-models
163. How to Fine-Tune an LLM Part 1: Preparing a Dataset for Instruction Tuning - Wandb, accessed May 5, 2025, https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-Tune-an-LLM-Part-1-Preparing-a-Dataset-for-Instruction-Tuning--Vmlldzo1NTcxNzE2
164. What is the correct format for dataset content for fine tuning the models (solved) - API, accessed May 5, 2025, https://community.openai.com/t/what-is-the-correct-format-for-dataset-content-for-fine-tuning-the-models-solved/691954
165. LLM fine tuning for NER using UBIAI annotation tool, accessed May 5, 2025, https://ubiai.tools/llm-fine-tuning-for-ner-using-ubiai-annotation-tool/
166. Fine-Tuning In a Nutshell with a Single Line JSONL File and n_epochs - Documentation, accessed May 5, 2025, https://community.openai.com/t/fine-tuning-in-a-nutshell-with-a-single-line-jsonl-file-and-n-epochs/60806
167. poteminr/instruct-ner: Instruct LLMs for flat and nested NER. Fine-tuning Llama and Mistral models for instruction named entity recognition. (Instruction NER) - GitHub, accessed May 5, 2025, https://github.com/poteminr/instruct-ner
168. Fine-Tuning Mistral 7B for Named Entity Recognition - UBIAI tool, accessed

May 5, 2025, https://ubiai.tools/fine-tuning-mistral-for-ner/
169. I will do the fine-tuning for you, or here's my DIY guide : r/LocalLLaMA - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/18n2bwu/i_will_do_the_finetuning_for_you_or_heres_my_diy/
170. Fine-Tuning for retrieval augmented generation (RAG) with Qdrant | OpenAI Cookbook, accessed May 5, 2025, https://cookbook.openai.com/examples/fine-tuned_qa/ft_retrieval_augmented_generation_qdrant
171. Dataset Formats – Axolotl - GitHub Pages, accessed May 5, 2025, https://axolotl-ai-cloud.github.io/axolotl/docs/dataset-formats/
172. Chat Datasets — torchtune 0.3 documentation - PyTorch, accessed May 5, 2025, https://pytorch.org/torchtune/0.3/basics/chat_datasets.html
173. LLM Classification Finetuning | Kaggle, accessed May 5, 2025, https://www.kaggle.com/competitions/llm-classification-finetuning/data
174. Datasets | Open Assistant - laion-ai.github.io, accessed May 5, 2025, https://laion-ai.github.io/Open-Assistant/docs/data/datasets
175. Open-Assistant/data/datasets/README.md at main - GitHub, accessed May 5, 2025, https://github.com/LAION-AI/Open-Assistant/blob/main/data/datasets/README.md
176. Data Schemas | Open Assistant - laion-ai.github.io, accessed May 5, 2025, https://laion-ai.github.io/Open-Assistant/docs/data/schemas
177. Finetuned Language Models Are Zero-Shot Learners - ResearchGate, accessed May 5, 2025, https://www.researchgate.net/publication/354379338_Finetuned_Language_Models_Are_Zero-Shot_Learners
178. README.md · Arun63/sharegpt-structured-output-json at main - Hugging Face, accessed May 5, 2025, https://huggingface.co/datasets/Arun63/sharegpt-structured-output-json/blob/main/README.md
179. ChatGPT Prompt Engineering for Developers - DeepLearning.AI - Learning Platform, accessed May 5, 2025, https://learn.deeplearning.ai/courses/chatgpt-prompt-eng/lesson/jtmdv/chatbot
180. Fine-tuning GPT-J on conversations (format of dataset) : r/LanguageTechnology - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LanguageTechnology/comments/10bmbky/finetuning_gptj_on_conversations_format_of_dataset/
181. Building an LLM fine-tuning Dataset - YouTube, accessed May 5, 2025, https://www.youtube.com/watch?v=pCX_3p40Efc&pp=0gcJCfcAhR29_xXO
182. Supervised Fine-Tuning - Hugging Face LLM Course, accessed May 5, 2025, https://huggingface.co/learn/nlp-course/en/chapter11/1
183. Token Classification Model with Named Entity Recognition (NER) - NVIDIA Docs, accessed May 5, 2025, https://docs.nvidia.com/nemo-framework/user-guide/24.12/nemotoolkit/nlp/token

_classification.html
184. asahi417/tner: Language model fine-tuning on NER with an easy interface and cross-domain evaluation. "T-NER: An All-Round Python Library for Transformer-based Named Entity Recognition, EACL 2021" - GitHub, accessed May 5, 2025, https://github.com/asahi417/tner
185. Exploring the Potential of Large Language Models (LLMs) for Low-resource Languages: A Study on Named-Entity Recognition (NER) an - ACL Anthology, accessed May 5, 2025, https://aclanthology.org/2024.lrec-main.611.pdf
186. HEMANGANI/Fine-Tuning-LLM-for-QA: Fine-Tuning Large Language Models for Question Answering - GitHub, accessed May 5, 2025, https://github.com/HEMANGANI/Fine-Tuning-LLM-for-QA
187. LLM QA Builder - Research Computing Systems - Lehigh Confluence, accessed May 5, 2025, https://lehigh.atlassian.net/wiki/spaces/hpc/pages/153583813/LLM+QA+Builder
188. Fine tuning a model for customer service for our specific app - OpenAI Developer Forum, accessed May 5, 2025, https://community.openai.com/t/fine-tuning-a-model-for-customer-service-for-our-specific-app/29376
189. What is SQuAD (Stanford Question Answering Dataset)? - H2O.ai, accessed May 5, 2025, https://h2o.ai/wiki/squad/
190. Evaluating QA: Metrics, Predictions, and the Null Response | NLP for Question Answering, accessed May 5, 2025, https://qa.fastforwardlabs.com/no%20answer/null%20threshold/bert/distilbert/exact%20match/f1/robust%20predictions/2020/06/09/Evaluating_BERT_on_SQuAD.html
191. evaluate/metrics/squad_v2/README.md at main · huggingface/evaluate - GitHub, accessed May 5, 2025, https://github.com/huggingface/evaluate/blob/main/metrics/squad_v2/README.md
192. Ultimate Guide to LLM Fine-tuning 2025 - Rapid Innovation, accessed May 5, 2025, https://www.rapidinnovation.io/post/fine-tuning-large-language-models-llms
193. Ethical Considerations and Fundamental Principles of Large Language Models in Medical Education: Viewpoint - PMC, accessed May 5, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11327620/
194. pii-anonymizer - PyPI, accessed May 5, 2025, https://pypi.org/project/pii-anonymizer/
195. PII Masker is an open-source tool for protecting sensitive data by automatically detecting and masking PII using advanced AI, powered by DeBERTa-v3. It provides high-precision detection, scalable performance, and a simple Python API for seamless integration into workflows, ensuring privacy compliance in various industries. - GitHub, accessed May 5, 2025, https://github.com/HydroXai/pii-masker
196. PII Detection and Masking in RAG Pipelines - Analytics Vidhya, accessed May 5, 2025,

https://www.analyticsvidhya.com/blog/2024/03/pii-detection-and-masking-in-rag-pipelines/
197. presidio-anonymizer - PyPI, accessed May 5, 2025, https://pypi.org/project/presidio-anonymizer/
198. Data anonymization with Microsoft Presidio | 🦜 LangChain, accessed May 5, 2025, https://python.langchain.com/v0.1/docs/guides/productionization/safety/presidio_data_anonymization/
199. the-mask - PyPI, accessed May 5, 2025, https://pypi.org/project/the-mask/
200. graemeworthy/pii_mask: This is an an experimental implementation of field-level data masking of Personally Identifiable Information (PII) for use in Django. - GitHub, accessed May 5, 2025, https://github.com/graemeworthy/pii_mask
201. Presidio Anonymizer, accessed May 5, 2025, https://microsoft.github.io/presidio/anonymizer/
202. Reversible data anonymization with Microsoft Presidio - LangChain, accessed May 5, 2025, https://python.langchain.com/v0.1/docs/guides/productionization/safety/presidio_data_anonymization/reversible/
203. presidio/docs/samples/python/example_structured.ipynb at main - GitHub, accessed May 5, 2025, https://github.com/microsoft/presidio/blob/main/docs/samples/python/example_structured.ipynb
204. Build RAG with Milvus + PII Masker, accessed May 5, 2025, https://milvus.io/docs/RAG_with_pii_and_milvus.md
205. LLM Fun: Building a Q&A Bot of Myself | Bounded Rationality, accessed May 5, 2025, https://bjlkeng.io/posts/building-a-qa-bot-of-me-with-openai-and-cloudflare/
206. Tokenization in Large Language Models (LLMs) - ingoampt - Artificial Intelligence integration into iOS apps and SaaS + Education, accessed May 5, 2025, https://ingoampt.com/tokenization-in-large-language-models-llms/
207. 5 Approaches to Solve LLM Token Limits - Deepchecks, accessed May 5, 2025, https://www.deepchecks.com/5-approaches-to-solve-llm-token-limits/
208. Calculating LLM Token Counts: A Practical Guide - Winder.AI, accessed May 5, 2025, https://winder.ai/calculating-token-counts-llm-context-windows-practical-guide/
209. Fine-Tuning LLaMa 2 for Text Summarization | ml-articles – Weights & Biases - Wandb, accessed May 5, 2025, https://wandb.ai/mostafaibrahim17/ml-articles/reports/Fine-Tuning-LLaMa-2-for-Text-Summarization--Vmlldzo2NjA1OTAy
210. Getting the most out of your tokenizer for pre-training and domain adaptation - arXiv, accessed May 5, 2025, https://arxiv.org/html/2402.01035v2
211. Tokenizers — torchtune 0.4 documentation - PyTorch, accessed May 5, 2025, https://pytorch.org/torchtune/0.4/basics/tokenizers.html
212. Fine-Tune Mistral-7B with LoRA A Quickstart Guide - DigitalOcean, accessed

May 5, 2025, https://www.digitalocean.com/community/tutorials/mistral-7b-fine-tuning
213. imoneoi/mistral-tokenizer - GitHub, accessed May 5, 2025, https://github.com/imoneoi/mistral-tokenizer
214. Vocabulary expansion for non-SentencePiece based BPE tokeniser, accessed May 5, 2025, https://gucci-j.github.io/post/en/vocab-expansion/
215. What is the maximum input length an LLM can handle? - Milvus, accessed May 5, 2025, https://milvus.io/ai-quick-reference/what-is-the-maximum-input-length-an-llm-can-handle
216. LLM Prompt Best Practices for Large Context Windows - Winder.AI, accessed May 5, 2025, https://winder.ai/llm-prompt-best-practices-large-context-windows/
217. Finetuning Large Language Models for Longer Context Lengths - Esperanto Technologies, accessed May 5, 2025, https://www.esperanto.ai/blog/finetuning-large-language-models-for-longer-context-lengths/
218. Strategies for Preserving Long-Term Context in LLMs? : r/LocalLLaMA - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1jxiz2y/strategies_for_preserving_longterm_context_in_llms/
219. Long Context Fine-Tuning: A Technical Deep Dive - Together AI, accessed May 5, 2025, https://www.together.ai/blog/long-context-fine-tuning-a-technical-deep-dive
220. LLM Maybe LongLM: Self-Extend LLM Context Window Without Tuning - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/18x8g6c/llm_maybe_longlm_selfextend_llm_context_window/
221. Huggingface Model Max Length Explained | Restackio, accessed May 5, 2025, https://www.restack.io/p/transformer-models-answer-huggingface-max-length-cat-ai
222. LLM Fine Tuning Parameters - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/autotrain/llm_finetuning_params
223. Question About the Practicality of the Context Length - Models - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/question-about-the-practicality-of-the-context-length/73569
224. Everything About Long Context Fine-tuning - Hugging Face, accessed May 5, 2025, https://huggingface.co/blog/wenbopan/long-context-fine-tuning
225. Fine-tuning LLMs for longer context and better RAG systems - Anyscale, accessed May 5, 2025, https://www.anyscale.com/blog/fine-tuning-llms-for-longer-context-and-better-rag-systems
226. Long Context Training/Finetuning through Reinforcement-Learning Bootstrapping. A (probably stupid) Idea : r/LocalLLaMA - Reddit, accessed May 5,

2025, https://www.reddit.com/r/LocalLLaMA/comments/1iquigb/long_context_trainingfinetuning_through/
227. [2309.12307] LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models, accessed May 5, 2025, https://arxiv.org/abs/2309.12307
228. How can I fine-tune a LLM to increase *effective context*? : r/LocalLLaMA - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1f9hprp/how_can_i_finetune_a_llm_to_increase_effective/
229. Fine tuning LLM for exhaustive summary of a possible long document : r/ollama - Reddit, accessed May 5, 2025, https://www.reddit.com/r/ollama/comments/1hl3l7o/fine_tuning_llm_for_exhaustive_summary_of_a/
230. Finetuning summarization model using long text data - Beginners - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/finetuning-summarization-model-using-long-text-data/77010
231. Understanding context length and memory usage : r/LocalLLaMA - Reddit, accessed May 5, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1j7r1sm/understanding_context_length_and_memory_usage/
232. Flash Attention 2 | Continuum Labs, accessed May 5, 2025, https://training.continuumlabs.ai/inference/why-is-inference-important/flash-attention-2
233. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning, accessed May 5, 2025, https://openreview.net/forum?id=mZn2Xyh9Ec
234. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning, accessed May 5, 2025, https://princeton-nlp.github.io/flash-atttention-2/
235. Exploring Longformer - Scaler Topics, accessed May 5, 2025, https://www.scaler.com/topics/nlp/longformer/
236. Longformer: The Long-Document Transformer, accessed May 5, 2025, https://ysu1989.github.io/courses/au20/cse5539/Longformer.pdf
237. Longformer Explained | Papers With Code, accessed May 5, 2025, https://paperswithcode.com/method/longformer
238. Longformer: Long Sequence Transformer | Ultralytics, accessed May 5, 2025, https://www.ultralytics.com/glossary/longformer
239. 9 LLM Summarization Strategies to Maximize AI Output Quality - Galileo AI, accessed May 5, 2025, https://www.galileo.ai/blog/llm-summarization-strategies
240. Understanding LLM Batch Inference - Adaline, accessed May 5, 2025, https://www.adaline.ai/blog/llm-batch-inference
241. LLMOps Architecture : A Detailed Explanation - TrueFoundry, accessed May 5, 2025, https://www.truefoundry.com/blog/llmops-architecture
242. Supervised Fine-tuning Trainer - Hugging Face, accessed May 5, 2025,

https://huggingface.co/docs/trl/sft_trainer
243. How to Fine-tune an LLM Part 3: The HuggingFace Trainer | alpaca_ft - Wandb, accessed May 5, 2025, https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-tune-an-LLM-Part-3-The-HuggingFace-Trainer--Vmlldzo1OTEyNjMy
244. Trainer - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/main/main_classes/trainer
245. IST-DASLab/peft-rosa: A fork of the PEFT library, supporting Robust Adaptation (RoSA), accessed May 5, 2025, https://github.com/IST-DASLab/peft-rosa
246. PEFT - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/peft/index
247. Fine tune a finetuned model - Beginners - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/fine-tune-a-finetuned-model/131732
248. Correct way to save/load adapters and checkpoints in PEFT - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/correct-way-to-save-load-adapters-and-checkpoints-in-peft/77836
249. I wonder how to merge my PEFT adapter with the base model and finally get a new whole model? - #2 by John6666 - Transformers - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/i-wonder-how-to-merge-my-peft-adapter-with-the-base-model-and-finally-get-a-new-whole-model/139138/2
250. Using, saving, and loading an encapsulated PEFT-tuned model inside a regular pytorch model - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/using-saving-and-loading-an-encapsulated-peft-tuned-model-inside-a-regular-pytorch-model/133577
251. Incorrect Saving Peft Models using HuggingFace Trainer · Issue #96 - GitHub, accessed May 5, 2025, https://github.com/huggingface/peft/issues/96
252. Difference between AutoModelForCausalLM and peft_model.merge_and_unload() for a LoRA model during inference - Transformers - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/difference-between-automodelforcausallm-and-peft-model-merge-and-unload-for-a-lora-model-during-inference/72455
253. I wonder how to merge my PEFT adapter with the base model and finally get a new whole model? - Transformers - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/i-wonder-how-to-merge-my-peft-adapter-with-the-base-model-and-finally-get-a-new-whole-model/139138
254. wrong model loading after merge_and_unload + save_pretrained · Issue #1381 · huggingface/peft - GitHub, accessed May 5, 2025, https://github.com/huggingface/peft/issues/1381
255. deep-learning-pytorch-huggingface/training/fine-tune-llms-in-2024-with-trl.ipynb at main, accessed May 5, 2025, https://github.com/philschmid/deep-learning-pytorch-huggingface/blob/main/trai

ning/fine-tune-llms-in-2024-with-trl.ipynb
256. Trainer - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/trainer
257. Trainer - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.35.2/main_classes/trainer
258. Opinion: Training Argument Fine Tuning MLM RoBERTa - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/opinion-training-argument-fine-tuning-mlm-roberta/135013
259. How to fix the learning-rate for Huggingface´s Trainer? - Stack Overflow, accessed May 5, 2025, https://stackoverflow.com/questions/77792137/how-to-fix-the-learning-rate-for-huggingface%C2%B4s-trainer
260. load and train with bf16, saved torch_dtype is float32 · Issue #30305 · huggingface/transformers - GitHub, accessed May 5, 2025, https://github.com/huggingface/transformers/issues/30305
261. Transformers for Natural Language Processing[Book] - O'Reilly Media, accessed May 5, 2025, https://www.oreilly.com/library/view/transformers-for-natural/9781800565791/
262. Transformers for Natural Language Processing - Second Edition[Book] - O'Reilly Media, accessed May 5, 2025, https://www.oreilly.com/library/view/transformers-for-natural/9781803247335/
263. Alpaca-lora for huggingface implementation using Deepspeed and FullyShardedDataParallel - GitHub, accessed May 5, 2025, https://github.com/naem1023/alpaca-lora-for-huggingface
264. trl/docs/source/sft_trainer.md at main · huggingface/trl - GitHub, accessed May 5, 2025, https://github.com/huggingface/trl/blob/main/docs/source/sft_trainer.md
265. Accelerate - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/accelerate
266. DeepSpeed vs FSDP: A Comprehensive Comparison - BytePlus, accessed May 5, 2025, https://www.byteplus.com/en/topic/499106
267. Accelerate vs. DeepSpeed vs. FSDP - Ben Gubler, accessed May 5, 2025, https://www.bengubler.com/posts/2023-08-29-accelerate-deepspeed-fsdp
268. FSDP vs DeepSpeed - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/accelerate/concept_guides/fsdp_and_deepspeed
269. D65-GPTQ/AWQ/Bitsandbytes in LLM Quantization - Kaggle, accessed May 5, 2025, https://www.kaggle.com/code/simranjeetsingh1430/d65-gptq-awq-bitsandbytes-in-llm-quantization
270. LLM Quantization: Quantize Model with GPTQ, AWQ, and Bitsandbytes | Towards AI, accessed May 5, 2025, https://towardsai.net/p/artificial-intelligence/llm-quantization-quantize-model-with-gptq-awq-and-bitsandbytes
271. FullyShardedDataParallel — PyTorch 2.6 documentation, accessed May 5,

2025, https://pytorch.org/docs/stable/fsdp.html
272. Trainer - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/trl/main/trainer
273. LLM Quantization Techniques- GPTQ | Towards AI, accessed May 5, 2025, https://towardsai.net/p/l/llm-quantization-techniques-gptq
274. How to get out of stagnant loss - Intermediate - Hugging Face Forums, accessed May 5, 2025, https://discuss.huggingface.co/t/how-to-get-out-of-stagnant-loss/141629
275. Trainer - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.22.2/en/main_classes/trainer
276. Optimization - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/main_classes/optimizer_schedules
277. GPU - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/main/perf_train_gpu_one
278. Trainer only saves model in FP16 when using mixed precision together with DeepSpeed · Issue #28921 · huggingface/transformers - GitHub, accessed May 5, 2025, https://github.com/huggingface/transformers/issues/28921
279. Methods and tools for efficient training on a single GPU - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/transformers/v4.39.2/perf_train_gpu_one
280. How to save the merged model trained with peft? · Issue #3246 · UKPLab/sentence-transformers - GitHub, accessed May 5, 2025, https://github.com/UKPLab/sentence-transformers/issues/3246
281. Perplexity In NLP: Understand How To Evaluate LLMs [Practical Guide] - Spot Intelligence, accessed May 5, 2025, https://spotintelligence.com/2024/08/19/perplexity-in-nlp/
282. Classification: Accuracy, recall, precision, and related metrics | Machine Learning, accessed May 5, 2025, https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall
283. Understanding F1 Score, Accuracy, ROC-AUC & PR-AUC Metrics - Deepchecks, accessed May 5, 2025, https://www.deepchecks.com/f1-score-accuracy-roc-auc-and-pr-auc-metrics-for-models/
284. F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose? - Neptune.ai, accessed May 5, 2025, https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc
285. 9 Accuracy Metrics to Evaluate AI Model Performance - Galileo AI, accessed May 5, 2025, https://www.galileo.ai/blog/accuracy-metrics-ai-evaluation
286. LLM Evaluation Metrics: The Ultimate LLM Evaluation Guide - Confident AI, accessed May 5, 2025, https://www.confident-ai.com/blog/llm-evaluation-metrics-everything-you-need-for-llm-evaluation
287. How to Evaluate LLMs Using Hugging Face Evaluate - Analytics Vidhya, accessed May 5, 2025,

https://www.analyticsvidhya.com/blog/2025/04/hugging-face-evaluate/
288. How to Use Hugging Face's New Evaluate Library - Vennify.ai, accessed May 5, 2025, https://www.vennify.ai/hugging-face-evaluate-library/
289. Task-Specific LLM Evals that Do & Don't Work - Eugene Yan, accessed May 5, 2025, https://eugeneyan.com/writing/evals/
290. Understanding and Applying F1 Score: AI Evaluation Essentials with Hands-On Coding Example, accessed May 5, 2025, https://arize.com/blog-course/f1-score/
291. Evaluating NLP Models: A Comprehensive Guide to ROUGE, BLEU, METEOR, and BERTScore Metrics | In Plain English, accessed May 5, 2025, https://plainenglish.io/community/evaluating-nlp-models-a-comprehensive-guide-to-rouge-bleu-meteor-and-bertscore-metrics-d0f1b1
292. How is the F1 score calculated in a question-answering system? - AI Stack Exchange, accessed May 5, 2025, https://ai.stackexchange.com/questions/22676/how-is-the-f1-score-calculated-in-a-question-answering-system
293. Monitoring Text-Based Generative AI Models Using Metrics Like Bleu Score - Arize AI, accessed May 5, 2025, https://arize.com/blog-course/generative-ai-metrics-bleu-score/
294. Understanding BLEU, ROUGE, and Modern NLP Metrics - Adaline, accessed May 5, 2025, https://www.adaline.ai/blog/understanding-bleu-rouge-and-modern-nlp-metrics
295. RAG evaluation metrics: A journey through metrics - Elasticsearch Labs, accessed May 5, 2025, https://www.elastic.co/search-labs/blog/evaluating-rag-metrics
296. We ran over half a million evaluations on quantized LLMs—here's what we found, accessed May 5, 2025, https://developers.redhat.com/articles/2024/10/17/we-ran-over-half-million-evaluations-quantized-llms
297. Teams Contributions to the Books About Modern NLP | deepset, accessed May 5, 2025, https://www.deepset.ai/books
298. Perplexity for LLM Evaluation - Comet.ml, accessed May 5, 2025, https://www.comet.com/site/blog/perplexity-for-llm-evaluation/
299. Decoding Perplexity and its significance in LLMs - UpTrain AI, accessed May 5, 2025, https://blog.uptrain.ai/decoding-perplexity-and-its-significance-in-llms/
300. Perplexity Metric for LLM Evaluation - Analytics Vidhya, accessed May 5, 2025, https://www.analyticsvidhya.com/blog/2025/04/perplexity-metric-for-llm-evaluation/
301. Perplexity for LLM Evaluation | GeeksforGeeks, accessed May 5, 2025, https://www.geeksforgeeks.org/perplexity-for-llm-evaluation/
302. How to Use Hugging Face's New Evaluate Library - YouTube, accessed May 5, 2025, https://www.youtube.com/watch?v=cnp1cGMjSn0
303. Using the `evaluator` - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/evaluate/base_evaluator
304. huggingface/evaluate: Evaluate: A library for easily evaluating machine learning models and datasets. - GitHub, accessed May 5, 2025,

https://github.com/huggingface/evaluate
305. lm-evaluation-harness/docs/API_guide.md at main - GitHub, accessed May 5, 2025, https://github.com/EleutherAI/lm-evaluation-harness/blob/main/docs/API_guide.md
306. lm-evaluation-harness/docs/model_guide.md at main - GitHub, accessed May 5, 2025, https://github.com/EleutherAI/lm-evaluation-harness/blob/main/docs/model_guide.md
307. lm-evaluation-harness/lm_eval/tasks/README.md at main - GitHub, accessed May 5, 2025, https://github.com/EleutherAI/lm-evaluation-harness/blob/main/lm_eval/tasks/README.md
308. lm-evaluation-harness/lm_eval/tasks/fda/README.md at main - GitHub, accessed May 5, 2025, https://github.com/EleutherAI/lm-evaluation-harness/blob/main/lm_eval/tasks/fda/README.md
309. lm-evaluation-harness/lm_eval/tasks/tinyBenchmarks/README.md at main - GitHub, accessed May 5, 2025, https://github.com/EleutherAI/lm-evaluation-harness/blob/main/lm_eval/tasks/tinyBenchmarks/README.md
310. Holistic Evaluation of Language Models (HELM) - Stanford NLP Group, accessed May 5, 2025, https://nlp.stanford.edu/helm/vhelm_lite/
311. MMLU - Holistic Evaluation of Language Models (HELM) - Stanford CRFM, accessed May 5, 2025, https://crfm.stanford.edu/helm/mmlu/latest/
312. HELM Classic - Holistic Evaluation of Language Models (HELM) - Stanford CRFM, accessed May 5, 2025, https://crfm.stanford.edu/helm/classic/latest/
313. HELM Lite - Holistic Evaluation of Language Models (HELM) - Stanford CRFM, accessed May 5, 2025, https://crfm.stanford.edu/helm/lite/latest/
314. Stanford Researchers Develop HELM Benchmark for Language Models - Datanami, accessed May 5, 2025, https://www.bigdatawire.com/2022/11/22/stanford-researchers-develop-helm-benchmark-for-language-models/
315. Quantization Techniques Demystified: Boosting Efficiency in Large Language Models (LLMs) - Inferless, accessed May 5, 2025, https://www.inferless.com/learn/quantization-techniques-demystified-boosting-efficiency-in-large-language-models-llms
316. Introduction to quantizing ML models | Baseten Blog, accessed May 5, 2025, https://www.baseten.co/blog/introduction-to-quantizing-ml-models/
317. Top LLM Quantization Methods and Their Impact on Model Quality - Deepchecks, accessed May 5, 2025, https://www.deepchecks.com/top-llm-quantization-methods-impact-on-model-quality/
318. Optimizing LLM Inference for Faster Results Using Quantization - A Hands on Guide, accessed May 5, 2025,

https://adasci.org/optimizing-llm-inference-for-faster-results-using-quantization-a-hands-on-guide/
319. [vLLM vs TensorRT-LLM] #6. Weight-Only Quantization - SqueezeBits, accessed May 5, 2025, https://blog.squeezebits.com/vllm-vs-tensorrtllm-6-weightonly-quantization-33728
320. DAQ: Density-Aware Post-Training Weight-Only Quantization For LLMs - arXiv, accessed May 5, 2025, https://arxiv.org/html/2410.12187v2
321. Sustainable LLM Inference for Edge AI: Evaluating Quantized LLMs for Energy Efficiency, Output Accuracy, and Inference Latency - arXiv, accessed May 5, 2025, https://arxiv.org/html/2504.03360v1
322. vLLM serving for text-only and multimodal language models on Cloud GPUs | Generative AI on Vertex AI | Google Cloud, accessed May 5, 2025, https://cloud.google.com/vertex-ai/generative-ai/docs/open-models/vllm/use-vllm
323. What is vLLM? - Red Hat, accessed May 5, 2025, https://www.redhat.com/en/topics/ai/what-is-vllm
324. Vllm Explained: Understanding Its Features | Restackio, accessed May 5, 2025, https://www.restack.io/p/vllm-answer-explained-cat-ai
325. What is vLLM: A Guide to Quick Inference - Hyperstack, accessed May 5, 2025, https://www.hyperstack.cloud/blog/case-study/what-is-vllm-a-guide-to-quick-inference
326. vLLM vs. TGI: Comparing Inference Libraries for Efficient LLM Deployment (2024 Guide), accessed May 5, 2025, https://www.inferless.com/learn/vllm-vs-tgi-the-ultimate-comparison-for-speed-scalability-and-llm-performance
327. Understanding and Implementing KV Cache for Efficient LLM Inference - Kolosal AI, accessed May 5, 2025, https://kolosal.ai/articles/LLM-Optimization/Understanding-and-Implementing-KV-Cache-for-Efficient-LLM-Inference
328. Dynamic Batching in LLM for Enhancing Inferencing - Incubity by Ambilio, accessed May 5, 2025, https://incubity.ambilio.com/dynamic-batching-in-llm-for-enhancing-inferencing/
329. Boost your throughput with dynamic batching | Modal Blog, accessed May 5, 2025, https://modal.com/blog/batching-whisper
330. Vllm Vs Tgi Comparison | Restackio, accessed May 5, 2025, https://www.restack.io/p/vllm-answer-vllm-vs-tgi-cat-ai
331. Comparing vLLM and TGI for hosting LLMs - Tune AI, accessed May 5, 2025, https://tunehq.ai/blog/comparing-vllm-and-tgi
332. Hugging Face's Text Generation Inference Toolkit for LLMs - A Game Changer in AI, accessed May 5, 2025, https://www.datacamp.com/tutorial/hugging-faces-text-generation-inference-toolkit-for-llms
333. Running Hugging Face Text Generation Inference (TGI) with Llama-3.1 8B, accessed May 5, 2025,

https://rocm.docs.amd.com/projects/ai-developer-hub/en/latest/notebooks/inference/2_inference_ver3_HF_TGI.html
334. Text Generation Inference Architecture - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/text-generation-inference/architecture
335. Guidance - Hugging Face, accessed May 5, 2025, https://huggingface.co/docs/text-generation-inference/guidance
336. GenAI Inference Engines: TensorRT-LLM vs vLLM vs Hugging Face TGI vs LMDeploy, accessed May 5, 2025, https://nlpcloud.com/genai-inference-engines-tensorrt-llm-vs-vllm-vs-hugging-face-tgi-vs-lmdeploy.html
337. Triton Inference Server for Every AI Workload - NVIDIA, accessed May 5, 2025, https://www.nvidia.com/en-us/ai-data-science/products/triton-inference-server/
338. Triton Inference Server: The Basics and a Quick Tutorial - Run:ai, accessed May 5, 2025, https://www.run.ai/guides/machine-learning-engineering/triton-inference-server
339. NVIDIA Triton Inference Server, accessed May 5, 2025, https://www.nvidia.com/en-gb/ai-data-science/products/triton-inference-server/
340. The Triton Inference Server provides an optimized cloud and edge inferencing solution. - GitHub, accessed May 5, 2025, https://github.com/triton-inference-server/server
341. Dynamic Batching for LLMs | Wallaroo.AI (Version 2024.4), accessed May 5, 2025, https://docs.wallaroo.ai/wallaroo-llm/wallaroo-llm-optimizations/wallaroo-llm-optimizations-dynamic-batching/
342. Advanced KV Cache Optimization: Strategies for Memory-Efficient LLM Deployment, accessed May 5, 2025, https://www.modular.com/ai-resources/advanced-kv-cache-optimization-strategies-for-memory-efficient-llm-deployment
343. How Caching Helps Improve the LLM Inference? - ADaSci, accessed May 5, 2025, https://adasci.org/how-caching-helps-improve-the-llm-inference/
344. KV Caching in LLMs, explained visually - Daily Dose of Data Science, accessed May 5, 2025, https://www.dailydoseofds.com/p/kv-caching-in-llms-explained-visually/