TECHNICAL WHITE PAPER
September 2025

Domain-Specific Adaptation of Pre-Trained LLMs

A Strategic Guide for Enterprise Technical Knowledge Integration

vmware by Broadcom

Table of contents

Executive Summary

This article presents a comprehensive methodology for transforming open-source language models into specialized domain experts, using Llama 3.1-8B and VMware Cloud Infrastructure documentation as a case study. The approach addresses critical enterprise needs for accurate, context-aware AI assistants while maintaining cost-effectiveness and operational control.

Introduction

The Business Case for Domain Specialization

The business motivation to adapt open-source language models for enterprise-specific domains has moved from experimental to essential.

Picture this: In August 2025, OpenAI quietly released something that would fundamentally change how businesses think about AI costs. Their gpt-oss-20b model has been downloaded more than 3.65 million times in just a few weeks. But the real story isn't in these numbers; it's in what happened next.

A healthcare startup facing $50,000 monthly API costs for their medical AI assistant discovered they could run gtp-oss-20b on a $2,000 consumer computer, processing patient data at 16 tokens per second with complete HIPAA compliance as their data never leaves their premises. Similar adaptations have also occurred with older but effective models such as Llama3 (8b) that has been adapted to low-resource medical settings such as Meditron.

Many successful stories about LLMs powering business applications demonstrate that despite their impressive capabilities of open-source LLMs, it is essential to adapt them to specific knowledge domains where specialization is a must so the model can be used reliably for business applications.

Beyond RAG: Why Domain Adaptation Matters

Many organizations initially turn to Retrieval-Augmented Generation (RAG) as a quick solution for incorporating enterprise knowledge. However, production deployments reveal critical limitations. Research from Microsoft shows that RAG systems suffer from cascading failures where retrieval errors compound into generation errors. Studies indicate that even advanced RAG implementations achieve only 60-75% accuracy on complex multi-hop reasoning tasks. Finally, consider that as vector databases scale, performance degrades significantly, research shows accuracy drops of 10-12% per 100,000 documents (EyeLevel.ai research), while query latency can increase by 10x under load, from 200ms to 2 seconds (Milvus monitoring guide), making real-time applications unfeasible.

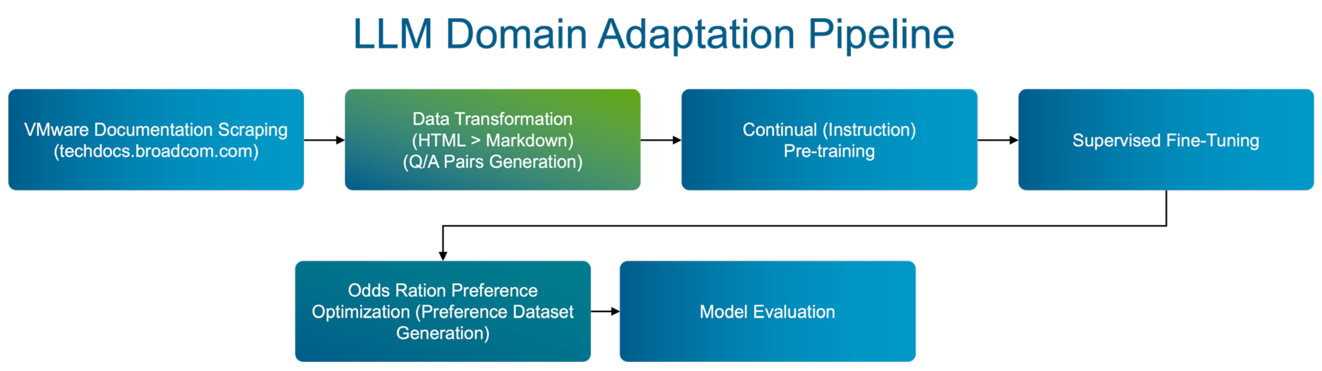
Domain-specific adaptation (continued pre-training followed by fine-tuning) offers a fundamentally different approach. Instead of retrieving and hoping for the right context, the model internalizes the knowledge, developing true expertise. This isn't just about memorizing facts, it's about understanding relationships, dependencies, and the subtle nuances that make the difference between a correct answer and the right answer for your specific context.

The Path Forward: A Proven Methodology

This guide presents a comprehensive methodology for organizations eager to start experimenting with domain adaptation techniques to adapt language models to their business use-cases. Using VMware Cloud Infrastructure's technical documentation as our specialized knowledge domain, we will walk through the stages that transform a vanilla (base) Llama3.1-8b model into a VMware technologies expert. We will also show you how to do it in an environment that is constrained to a single host with 8 x H100 GPUs.

The following picture illustrates the six-stage process to adapt LLMs to a specialized knowledge domain, these are:

1. Document Scraping (recollection). Documents get retrieved from a repository and get sorted in a semantically-ordered manner for the LLM to learn coherent text sequences.
2. Data Transformation. The original document formatting gets converted to a more convenient one for LLM training purposes. In addition, Q/A pairs get generated and appended to these documents as instruction pre-training techniques recommend.
3. Continual Pre-training. We continue the pre-training of the Llama3.1-8b base model so it learns all about the VMware technical documentation knowledge domain.
4. Supervised Fine-Tuning (SFT) to teach the pre-trained skills like human-style chatting, basic chain of thought reasoning and instruction following.
5. Odds Ration Preference Optimization (ORPO) to teach the model to prefer accurate response elaboration vs. inaccurate/ wrong answers that could be also deceiving.
6. Model Evaluation. Once the model has completed its adaptation training, it is necessary to apply rigorous evaluations on it to ensure it has properly adapted and mastered the new specialized knowledge domain.


LLM Domain Adaptation Pipeline
VMware Documentation Scraping (techdocs.broadcom.com)
Data Transformation (HTML > Markdown) (Q/A Pairs Generation)
Continual (Instruction) Pre-training
Supervised Fine-Tuning
Odds Ration Preference Optimization (Preference Dataset Generation)
Model Evaluation

Stage 1: Data Ingestion from Technical Documentation

Process Overview

The foundation of any successful model adaptation lies in comprehensive data acquisition. For technical documentation hosted on platforms like Broadcom's Techdocs for VMware Cloud Infrastructure, the ingestion process involves systematically crawling and downloading the entire documentation corpus while preserving its structural integrity.

Adapting large language models to understand a technical domain begins with something that seems deceptively simple: ingesting documentation. In practice, building a reliable ingestion pipeline is closer to engineering a distributed data system. It must balance coverage, fidelity, scalability, and compliance.

The journey can be thought of as a four-layer stack

1. Web Crawling and Acquisition -

The first layer is collecting raw material: navigating documentation portals and APIs to build a comprehensive dataset. A production-ready crawler must:

– Traverse nested navigation structures and cross-references.

– Handle JavaScript-heavy sites using tools like Puppeteer or Playwright.

– Manage authentication and access restrictions without breaking workflows.

Best Practice: Ethical crawling is essential. Guidelines include respecting robots.txt and using rate limits that protect host systems. A useful reference is Introduction to Information Retrieval (Manning et al., 2008): https://nlp.stanford.edu/IR-book/.

2. Storage Infrastructure

Once data is collected, the next challenge is storing at scale. Documentation datasets can easily run into tens or hundreds of gigabytes when multiple versions, images, and code samples are included.

– Robust setups typically involve:

– Durable distributed storage such as S3-type

– On-premises shared filesystems like NFS when using parallel processing clusters.

Version-aware approaches, so updates do not overwrite or duplicate vast amounts of data.

Best Practice: Make sure that the hosts (VMs) that participate in data ingestion, transformation and loading (for training loops) according to their role. For instance, hosts performing scraping and transformations need access to shared filesystems that store raw and transformed data. Conversely, hosts running training and evaluation processes might need access only to transformed data. Shared file systems allow collaborating hosts "see" the same data while proper access controls help keep control on data access points.

3. Network and Bandwidth Considerations

Large-scale documentation ingestion stresses both local and remote systems. To address this:

Implement **retry and error recovery mechanisms**, ensuring robustness against timeouts or API throttling.

Balance **throughput and politeness,** a well-tuned crawler avoids overwhelming servers while still progressing at the scale required.

Monitor for **schema drift** in the site structure over time, since navigation links and URL patterns often change.

Best Practice: Treat crawling sessions like distributed jobs. Orchestrators such as Apache Airflow or Luigi provide scheduling, retries, and monitoring.

4. Maintenance and Lifecycle Management

Documentation evolves continuously with each product release. Without active maintenance, the dataset quickly becomes stale. Strong pipelines:

• Schedule incremental updates, whether weekly or aligned with release cycles.

• Use data versioning tools like DVC or LakeFS to track changes across time.

• Adopt CI/CD practices in data engineering, including unit tests for scrapers, monitoring for crawl completeness, and alerts for schema drift.

Best Practice: Think of documentation ingestion as data operations. Just as ML models require retraining, ingestion pipelines require continuous validation to remain effective. There are online tutorials on how to run programmatic scraping pipelines based on robust tools like Airflow.

Stage 2: Data Transformation and Enhancement

Format Conversion Strategy

Converting HTML documentation to Markdown represents more than a technical necessity, it's a strategic optimization that fundamentally improves how models understand and process information. The AI industry has converged on this approach for compelling reasons.

When data is structured and predictable (in markdown format), the LLM responses are much better, as discovered by teams building production AI systems. This isn't just anecdotal, the benefits are rooted in how models are trained. LLMs thrive on clarity. Markdown provides a clear visual and structural distinction between different elements like titles, lists, and paragraphs. When an LLM sees a # Heading, it knows a new section is starting. This simplicity reduces ambiguity and helps the model better understand what's important and how to respond.

The hidden cost of HTML's verbosity

HTML's extensive markup creates a compounding token inefficiency problem that significantly impacts training economics. Research from the OpenAI developer community demonstrates this concretely: converting a 13,869-token JSON file to Markdown reduced it to 11,612 tokens - a **15% improvement that translates to millions of dollars in compute savings** at scale. This efficiency gap widens further with HTML, where opening and closing tags, attributes, and inline styling can double or triple the token count of equivalent content.

The tokenization process itself favors Markdown's design philosophy. Byte Pair Encoding (BPE) tokenizers, used by GPT-4 and similar models, process text by identifying common character sequences. HTML's syntax creates unnatural breaks in this process - a paragraph tag `<p>` splits into separate tokens, consuming valuable context window space without adding semantic value. **HTML can require up to 76% more tokens than Markdown** for identical content, according to comparative testing. This inefficiency forces researchers to either accept higher training costs or implement complex preprocessing pipelines to strip HTML down to its essential content.

Converting HTML to Markdown Can be a Daunting Task

The challenges encountered when converting VMware technical documentation from HTML to Markdown are far from unique. Across the technical documentation landscape, developers and technical writers are discovering that popular Python libraries like `html2text` and `markdownify` struggle with the complexity of modern technical documents, particularly when dealing with intricate table structures and properly formatted code blocks.

The Python Library Limitations

While Python's HTML-to-Markdown converters excel at simple content conversion, they often fail when confronted with:

- **Complex nested tables** with rowspan/colspan attributes common in API documentation
- **Multi-level code blocks** with syntax highlighting and line numbers
- **Mixed content cells** containing both formatted text and code snippets
- **Specialized technical markup** used in enterprise documentation platforms

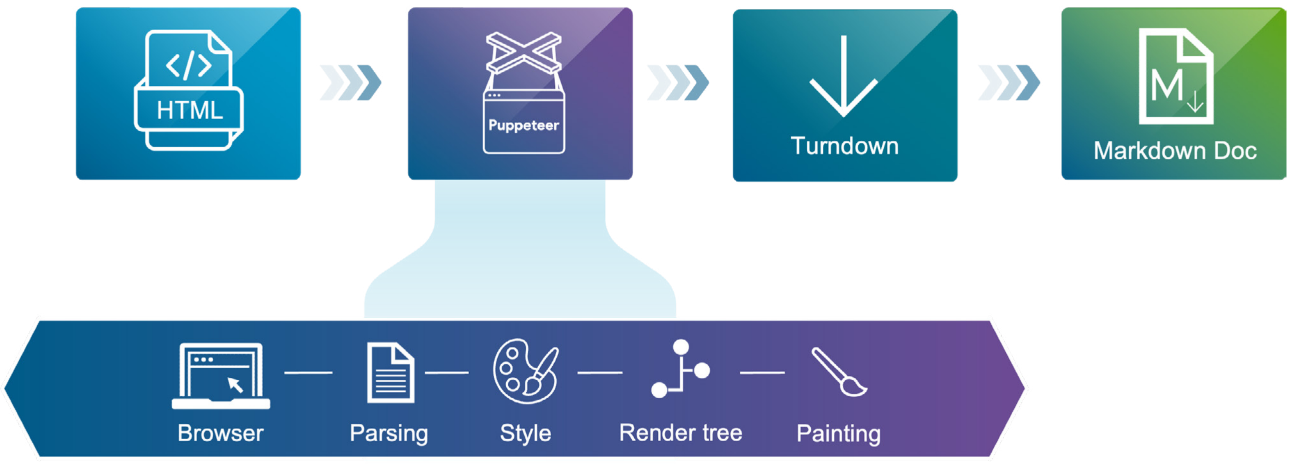
These limitations stem from fundamental architectural decisions in Python libraries. For instance, `html2text` offers options like `bypass_tables` to render tables as HTML rather than Markdown, essentially admitting defeat when faced with complex table structures. The `markdownify` library, while more customizable, still struggles with deeply nested structures and often produces malformed Markdown when processing technical documentation tables.

Why JavaScript Tools Excel

The shift toward JavaScript-based solutions like Puppeteer with Turndown isn't merely a trend, it's a response to real technical advantages:

1. **Full DOM Rendering:** Puppeteer launches an actual Chrome/Chromium instance, ensuring that dynamically generated content, JavaScript-rendered tables, and complex CSS layouts are fully processed before conversion. This is crucial for documentation platforms that rely heavily on client-side rendering.
2. **Robust HTML Parsing:** JavaScript's native DOM manipulation capabilities, combined with libraries like Turndown, provide more accurate parsing of complex HTML structures. Turndown was specifically designed with a deep understanding of how browsers interpret HTML, making it more reliable for edge cases.
3. **Customizable Conversion Rules:** Turndown's architecture allows for granular control over how specific HTML elements are converted. This is particularly valuable for technical documentation where custom elements or specific formatting patterns need special handling.

The following picture illustrates how Puppeteer and Turndown can be used to achieve high quality HTML to Markdown conversions.



A Pragmatic Approach

The evolution from Python to JavaScript tools for HTML-to-Markdown conversion reflects a broader pattern in web technologies. While Python remains excellent for many data processing tasks, the web-native capabilities of JavaScript make it better suited for tasks that require a deep understanding of HTML rendering and DOM structures.

For technical teams facing similar challenges, the Puppeteer + Turndown combination offers several advantages:

• Accuracy: Better preservation of table structures and code formatting

• Flexibility: Extensive customization options for handling edge cases

• Reliability: Consistent results across different documentation platforms

• Maintenance: Active development and community support

The extra complexity of setting up a Node.js environment and managing headless browser instances is often justified by the significant improvement in conversion quality, especially for mission-critical technical documentation where accuracy and formatting preservation are paramount.

Instruction Pre-training Methodology

The paradigm shift from traditional pre-training to instruction pre-training represents a fundamental rethinking of how models learn. Microsoft Research's work on **"Instruction Pre-Training: Language Models are Supervised Multitask Learners"** demonstrates that, instead of directly pre-training on raw corpora, Instruction Pre-Training augments each text from the raw corpora with a set of instruction-response pairs generated by an instruction synthesizer.

Considerable Performance Gains:
Researchers found that their 500M model pre-trained on 100B tokens reaches performance of the 1B model pre-trained on 300B tokens. This 2x efficiency in parameter usage fundamentally changes the economics of model deployment. Even more impressively, in continual pre-training, Instruction Pre-Training enables Llama3-8B to be comparable to or even outperform Llama3-70B, effectively bridging a nearly 10x parameter gap through superior training methodology.

The Science Behind the Success:
Why does instruction pre-training work so effectively? The research reveals that supervised multitask learning still holds significant promise, as scaling it in the post-training stage trends towards better generalization. By synthesizing 200M instruction-response pairs covering 40+ task categories, the approach ensures models learn not just facts but how to apply knowledge across diverse contexts.

The methodology transforms passive documentation into active learning through:

- Converting a wide range of context-based task completion datasets, which require models to perform tasks based on a given context
- Creating multi-turn synthesis where in each round, we prepend the texts and instruction-response pairs from previous rounds to the current text
- Ensuring diversity across various domains such as encyclopedias, social media, and academic tests
- Maintaining quality with over 85% relevance to the context and 70% response accuracy

Synthetic Pre-training Dataset Generation

The Instruction Synthesizer Architecture:

The key breakthrough lies in using an instruction synthesizer LLM, capable of generating diverse instruction-response pairs based on various raw corpora. The synthesizer operates on the principle that raw corpora contains numerous intrinsic tasks, which enables efficient scaling of task synthesis along with the upscale of raw corpora.

The synthesis process achieves remarkable quality metrics:

- Over 85% relevance to the original context
- 70% response accuracy across diverse domains
- Coverage of 49 different task categories in general pre-training
- Successful generation across technical, biomedical, and financial domains

It is important to highlight that the cited paper comes with lightweight instruction synthesizers which can generate good quality Q/A pairs at high inference speeds, however, if you'd like to generate very high quality pairs (but at lower paces), you could use more powerful models such as OpenAI gpt-oss-20b or Mistral AI Mistral-Small-3.1-24B-Instruct-2503.

Here is a prompt example for question/answer pair generation to augment technical documents.

```
prompt = f"""
- You are an expert in multiple VMware products, including VCF, vSphere, vSAN, and NSX, among
others.
- You must generate exactly {number_of_qa_pairs} question-answer pairs from the following
VMware documentation text.
- The question/answer pairs should be specific, technical, comprehensive, and MUST be based
solely on content from the document you are given.
- At the top of the document, you will find the source (the document's URL), product name,
and product version that the document discusses. You must use this information to the
citation at the end of each answer.
- The answers should be accurate, sufficiently extensive, and directly derived from the text.
- Focus on questions that would help a system administrator understand how to set up, manage,
or troubleshoot VMware products.

Here is the document content you must use to build the question/answer pairs:
{content}

Format your response as a sequence of markdown paragraphs started by 'Question' and 'Answer'
fields, according to the following example format:

<QUE> [your question here] <ANS> [your answer here] <END>

FORMATTING EXAMPLE:

<QUE> What are the key new features and updates in VMware Aria Automation 8.18.1?

<ANS> VMware Aria Automation 8.18.1 introduces several new features and updates:
**Full Username Display**: Full usernames are now displayed alongside user IDs in the UI
for Resources, Deployments, and Policy. This can be enabled via the Manage Columns button.
Similar support is available for Catalog, Content Source, and Content through the API.
**Removed Migration Support**: Migration from vRealize Automation 7.x to VMware Aria
Automation 8.x and NSX-V to NSX-T is no longer supported as the migration assistant has been
removed.
**DCGM Exporter Integration**: The Deep Learning VM (DLVM) image now includes DCGM-Exporter
by default, allowing Prometheus to monitor GPU metrics via an HTTP endpoint (/metrics).

**Source:** VMware Aria Operations 8.18 documentation
https://techdocs.broadcom.com/us/en/vmware-cis/aria/aria-operations/8-18/Chunk1327600040.html
<END>
"""
```

By using this prompt to augment source documentation, it is possible to get enhanced documents as the one shown below:

```
# VMware Aria Operations 8.17.1

[8.18](https://techdocs.broadcom.com/us/en/vmware-cis/aria/aria-operations/8-18

# User Scenario: Adding an Application

As the system administrator of an online training system, you must monitor components in the
Web, application, and database tiers of your environment that can affect the performance of
the system. You build an application that groups related objects together in each tier. If a
problem occurs with one of the objects, it is reflected in the application display and you
can open a summary to investigate the source of the problem further.

In your application, you add the DB-related objects that store data for the training system
in a tier, Web-related objects that run the user interface in a tier, and application-related
objects that process the data for the training system in a tier. The network tier might not
be needed. Use this model to develop your application.

-   From the left menu, click `Environment`, then click `Applications`in the left pane.
-   Click `ADD`.
-   Click `Basic n-tier Web App`and click `OK`.

    The Application Management page that appears has two rows. Select objects from the bottom
row to populate the tiers in the top row.

-   Type a meaningful name such as Online Training Application in the Application text box.
-   For each of the Web, application and database tiers listed, add the objects to the Tier
Objects section.
    -   Select a tier name. This is the tier that you populate.
    -   To the left of the object row, select object tags to filter for objects that have
that tag value. Click the tag name once to select the tag from the list and click the tag
name again to deselect the tag from the list. If you select multiple tags, objects displayed
depend on the values that you select.

        You can also search for the object by name.

    -   To the right of the object row, select the objects to add to the tier.
    -   Drag the objects to the Tier Objects section.

-   Click Save to save the application.

The new application appears in the list of applications on the Environment Overview
Applications page. If any of the components in any of the tiers develops a problem, the
application displays a yellow or red status.

To investigate the source of the problem, click the application name and see Evaluate Object
Information Using Badge Alerts and Summary Tab .


[Managing Application Groups](https://techdocs.broadcom.com/us/en/vmware-cis/aria/aria-
operations/8-17-1/vmware-aria-operations-configuration-guide-8-17-1/configuring-objects/
object-discovery/managing-application-groups.html)
</CTX>
---

<QUE> How do you add an application in VMware Aria Operations 8.17.1 to monitor components
across different tiers?

<ANS> To add an application in VMware Aria Operations 8.17.1 to monitor components across
different tiers, follow these steps:

1. From the left menu, click `Environment`, then click `Applications` in the left pane.
2. Click `ADD`.
```

```
3. Select `Basic n-tier Web App` and click `OK`. This opens the Application Management page
with two rows, allowing you to select objects from the bottom row to populate the tiers in
the top row.
4. Type a meaningful name for the application in the Application text box, such as “Online
Training Application.”
5. For each of the Web, application, and database tiers listed, add the relevant objects to
the Tier Objects section:
    - Select a tier name to specify which tier you are populating.
    - To the left of the object row, select object tags to filter for objects that have that
tag value. Click the tag name once to select it and again to deselect it. If multiple tags
are selected, objects displayed depend on the values chosen.
    - You can also search for the object by name.
    - To the right of the object row, select the objects to add to the tier.
    - Drag the selected objects to the Tier Objects section.
6. Click `Save` to save the application.

The new application will appear in the list of applications on the Environment Overview
Applications page. If any of the components in any of the tiers develop a problem, the
application will display a yellow or red status. To investigate the source of the problem,
click the application name and evaluate the object summary information.

**Source:** VMware Aria Operations 8.17.1 Configuration Guide
https://techdocs.broadcom.com/us/en/vmware-cis/aria/aria-operations/8-17-1/vmware-aria-
operations-configuration-guide-8-17-1/configuring-objects/object-discovery/managing-
application-groups/user-scenario-add-an-application.html
<END>

<QUE> How can you filter and select objects to add to specific tiers when creating an
application in VMware Aria Operations 8.17.1?

<ANS> When creating an application in VMware Aria Operations 8.17.1, you can filter and
select objects to add to specific tiers by following these steps:

1. Select a tier name to specify which tier you are populating (e.g., Web, application, or
database tier).
2. To the left of the object row, select object tags to filter for objects that have that tag
value. Click the tag name once to select it and again to deselect it. If multiple tags are
selected, the objects displayed will depend on the values chosen.
3. You can also search for the object by name using the search functionality.
4. To the right of the object row, select the objects you want to add to the tier.
5. Drag the selected objects to the Tier Objects section for the specific tier.

By using these filtering and selection methods, you can ensure that the correct objects
are added to each tier, allowing for effective monitoring and troubleshooting of the
application’s performance.

**Source:** VMware Aria Operations 8.17.1 Configuration Guide
https://techdocs.broadcom.com/us/en/vmware-cis/aria/aria-operations/8-17-1/vmware-aria-
operations-configuration-guide-8-17-1/configuring-objects/object-discovery/managing-
application-groups/user-scenario-add-an-application.html
<END>
```

Stage 3: Continual Pre-training on Extended Contexts

Long Sequence Processing Architecture

As previously mentioned, our experimentation leverages Llama3.1-8b as the base model to be adapted to the VMware Cloud Infrastructure technical documentation domain. The reasons for this design decision are:

- Llama3.1-8b has a context window of 128k tokens, which is big enough to be used as an assistant for multiple types of tasks.
- Given its size, it is possible to continue the Llama3.1-8b pretraining process and then fine-tune it on an environment constrained to a single node with 8 x H100 GPUs.
- Once fully trained, the final model can be executed on GPUs as small as the Nvidia A30 GPU that only has 24GB of memory.

When working with input sequences as long as 128k tokens, a key bottleneck is GPU memory (VRAM). In standard Transformer architectures, self-attention requires computing an interaction matrix between every pair of tokens, leading to quadratic growth in memory usage. Practically, this means that doubling the sequence length leads to about four times the VRAM demand, making extremely long-context training prohibitively expensive unless alternative attention mechanisms or memory-efficient techniques are used.

To overcome VRAM constraints, Zigzag ring attention emerged as a breakthrough solution, enabling efficient processing of documents up to millions of tokens on a single machine.

The Architecture Enables:
- Processing entire technical manuals as single sequence
- Maintaining relationships across distant document sections
- Efficient GPU memory utilization through distributed attention
- Progressive scaling from 8K to 128K token sequences
- The EasyContext GitHub repository provides a working implementation for Llama-type models.

Infrastructure Requirements and Optimization

The beauty of zigzag ring attention lies in its hardware efficiency. While traditional long-context training might require multiple nodes with dozens of GPUs, this approach achieves comparable results with:

Single Node Configuration:
- 8 x NVIDIA H100 GPUs (or equivalent)
- 1TB+ system RAM for data loading
- High-speed NVMe storage for checkpoint management
- NVLink (or equivalent) high-speed interconnect for GPU communication

Scaling Considerations
Our training process follows a curriculum approach, progressively increasing context lengths and their semantic payload:

Phase 1 (8K tokens):

- Foundation training on individual documentation sections
- Establishes basic technical vocabulary and concepts

Phase 2 (16K-32K tokens):

- Cross-section relationship learning
- Develops understanding of component interactions

Phase 3 (64K-128K tokens):

• Learns troubleshooting documents based on KBs, design guides and operations manuals.

• Masters complex troubleshooting workflows and architectural understanding

Why This Matters

Extended context processing fundamentally changes the model's capabilities. Instead of providing fragmented answers based on partial information, the model can consider entire technical specifications, understanding prerequisites, dependencies, and complex multi-step procedures. This capability proves invaluable for enterprise scenarios where solutions often require synthesizing information from multiple documentation sources.

Considerations

• **In our case**, we had to **implement training scripts** that use DeepSpeed and an adaptation of the Zigzag Ring Attention algorithms.

• **Comprehensive monitoring**. It is essential to monitor the training process via tools like MLflow or Weights and Biases so you can keep an eye on training and evaluation losses, GPU usage, and the overall state of the training process.

• **We also had to implement checkpointing routines** to be able to resume training from an interruption. This included saving of the optimizer's state in addition to the model's weights.

• **Numeric stability**. We had to run multiple experiments to find the right parameters for DeepSpeed and Zigzag ring attention to ensure numerical stability which might not be obtained at the first attempt (we were getting NaN values in loss calculations, for example).

Stage 4: SFT with off-the-shelf datasets

Supervised Fine-Tuning: Building on Strong Foundations

Following instruction pre-training, Supervised Fine-Tuning (SFT) serves as a critical refinement step that transforms a broadly capable model into one that excels at following instructions. This phase leverages high-quality instruction datasets to reinforce and enhance the model's ability to understand and respond to user queries effectively.

The Value of High-Quality Instruction Datasets

The success of SFT hinges on the quality of instruction data used. Datasets like OpenHermes 2.5 have emerged as gold standards for this purpose, offering:

• Over 1 million diverse instruction-response pairs covering reasoning, coding, analysis, and conversation

• Carefully curated examples that demonstrate clear, helpful, and accurate responses

• Balanced representation of different task types and complexity levels

• Consistent formatting that reinforces proper instruction-following patterns

For domain-specific applications, organizations typically blend these general instruction datasets with their specialized examples, creating a training mixture that maintains broad capabilities while emphasizing domain expertise.

Why LlamaFactory Excels for SFT Implementation

LlamaFactory has been very successful in the fine-tuning landscape by democratizing access to advanced training techniques while maintaining production-grade reliability. This unified framework represents a paradigm shift in how organizations approach model customization, offering an unprecedented combination of simplicity and power.

Core Benefits of LlamaFactory:

1. **Zero-to-Production Speed**: Unlike traditional training frameworks that require weeks of setup and configuration, LlamaFactory enables teams to go from raw model to fine-tuned deployment in hours. The framework's intelligent defaults and pre-configured templates eliminate the typical trial-and-error phase that consumes valuable engineering time.

2. **Unified Training Interface**: LlamaFactory consolidates multiple training paradigms, including SFT, RLHF, DPO, PPO, and ORPO, under a single, consistent interface. This means teams can experiment with different training strategies without learning new tools or rewriting infrastructure code. The same configuration format works across all training methods, dramatically reducing the learning curve.

3. **Production-Ready Optimizations**: The framework automatically implements cutting-edge optimization techniques that would typically require deep expertise to configure:

   – **LoRA and QLoRA**: Reduce memory requirements by up to 90% while maintaining performance

   – **FlashAttention-2**: Accelerates training and saves VRAM through optimized attention computation

   – **Gradient Checkpointing**: Also helps saving VRAM

   – **DeepSpeed Integration**: Seamlessly scales to multi-node training without code changes and provides different degrees of VRAM sayings by uploading components like the training optimizer to RAM or even disk

   – **Automatic Mixed Precision**: Speeds up training by 30-50% with minimal accuracy loss

4. **Comprehensive Model Support:** LlamaFactory supports over 100+ open-source models out of the box, including:

   – All Llama variants (1, 2, 3, 3.1, 3.2)

   – Mistral and Mixtral models

   – Custom architectures through simple configuration

5. Enterprise-Grade Features:

– **Web UI and API:** Non-technical stakeholders can monitor and control training through an intuitive interface

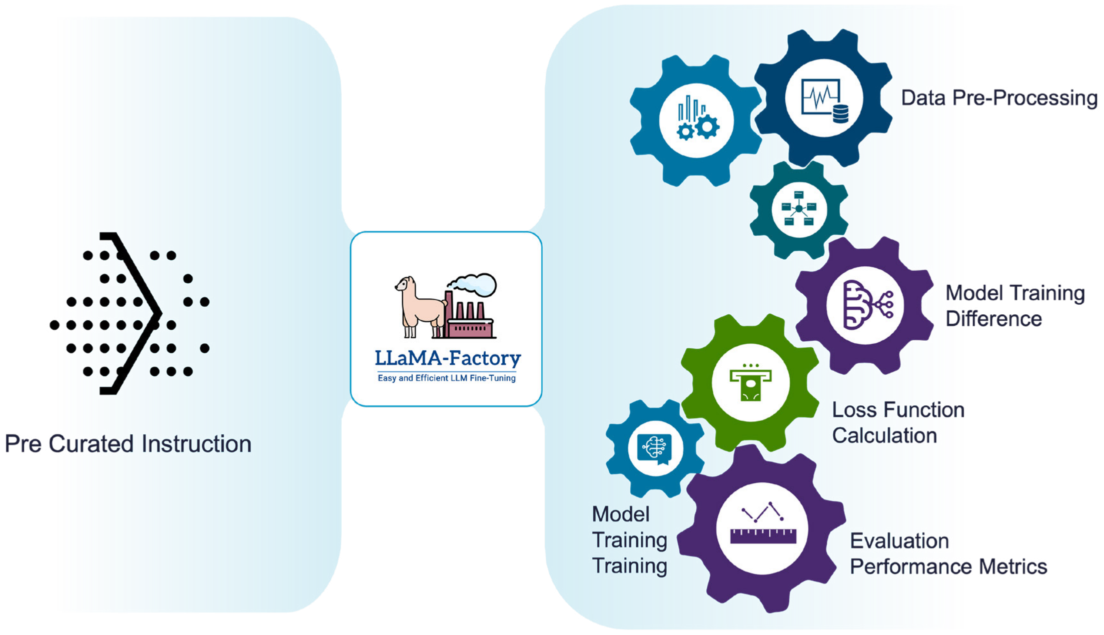
– **Experiment Tracking:** Built-in integration with Weights & Biases, TensorBoard, and MLflow

– **Checkpoint Management:** Automatic saving, resumption, and model merging capabilities

– **Multi-Dataset Training:** Simultaneous training on multiple datasets with automatic balancing

– **Dynamic Quantization:** Deploy models at various precision levels without retraining

The following figure illustrates the different tasks that LlamaFactory performs when training a model using SFT techniques


Pre Curated Instruction
LLaMA-Factory
Easy and Efficient LLM Fine-Tuning
Data Pre-Processing
Model Training Difference
Loss Function Calculation
Model Training Training
Evaluation Performance Metrics

Real-World Implementation Advantages:

The practical benefits of LlamaFactory become apparent in production scenarios:

• **Reduced Engineering Overhead:** Teams report 70-80% reduction in code required for fine-tuning pipelines

• **Faster Iteration Cycles:** Experiment turnaround time decreases from weeks to days

• **Lower Barrier to Entry:** Junior engineers can successfully fine-tune models with minimal supervision

• **Cost Optimization:** Automatic selection of optimal training strategies reduces GPU hours by 40-60%

• **Reproducibility:** Built-in versioning and configuration management ensure consistent results

Configuration Simplicity:

A complete SFT configuration in LlamaFactory requires just a simple YAML file:

```yaml
model_name_or_path: meta-llama/Llama-3.1-8B
dataset: openhermes25,your_custom_data
finetuning_type: lora
lora_rank: 32
learning_rate: 5e-5
num_train_epochs: 3
per_device_train_batch_size: 4
gradient_accumulation_steps: 4
output_dir: ./models/domain_expert
```

This configuration automatically handles:

• Dataset loading and preprocessing

• Hyperparameter selection

• Distributed training setup

• Memory optimization

• Checkpoint management

• Metric logging

In our use case, we have used LlamaFactory to run SFT training after running the continued pre-training stage on the Llama3.1-8b base model. The result is now a model that has expert knowledge of VMware Cloud Infrastructure products and that also has basic abilities to follow instructions and have conversations with humans

Stage 5: Preference-Based Fine-tuning

ORPO Methodology and Benefits

Once the base model has been further pre-trained and then fine-tuned via SFT, it is necessary to give it a good amount of preference training via techniques like Direct Preference Optimization (DPO) or ORPO, which we discuss in more detail next.

The Odds Ratio Preference Optimization (ORPO) represents a significant advancement over traditional fine-tuning approaches. Rather than simply teaching the model to mimic responses, ORPO trains it to consistently choose high-quality answers over inferior alternatives.

Key Advantages:

- More stable training compared to RLHF approaches
- Better alignment with human preferences
- Reduced tendency for hallucination
- Improved consistency in response quality

Dataset Design Principles

One of the distinguishing aspects of **ORPO (Online Rank Preference Optimization)** training is that it blends instruction tuning with preference learning in the same process. Instead of just teaching a model to predict the next token, ORPO directly optimizes for “better” responses over “worse” ones, using training triples in the form: `<user input, preferred (“good”) response, dispreferred (“bad”) response>`

The effectiveness of ORPO depends critically on how these preference datasets are designed. In technical contexts, the boundary between a strong response and a weak response often hinges on precision, clarity, and practical usability rather than just fluency.

What Defines a High-Quality Response?

A “good” response in technical documentation adaptation is not only factually correct but also contextually helpful. High-quality examples typically:

- **Accuracy and Precision** – Use correct terminology, consistent with established standards (e.g., “token limit” vs. “context window” are not interchangeable). References to specific versions, APIs, or configurations prevent ambiguity.
- **Logical Structure** – Present information in a structured, step-by-step manner that reflects how practitioners actually troubleshoot or configure systems.
- **Context Awareness** – Include caveats, edge cases, or compatibility warnings when relevant (e.g., “This setting works in v2.1+, but has changed in v3.0”).
- **Actionability** – Provide clear next steps or examples, such as code snippets, configuration commands, or references to related system components.

What Defines a Low-Quality Response?

A “bad” response in this training context is not just incorrect—it is also **unhelpful for real users of technical documentation**. Low-quality examples often:

- **Lack Depth** – Offer vague, generic answers that could apply to almost any problem (“Check your settings” instead of explaining which ones).
- **Omit Key Details** – Fail to mention parameters, flags, or prerequisites essential for successful execution.
- **Mislead Through Structure** – Provide steps in the wrong order (e.g., applying a patch after restarting a service). This reflects poorly in preference optimization, as ordering is crucial in workflows.
- **Ignore Context** – Recommend approaches that are deprecated, version-incompatible, or unsafe without acknowledging those caveats.

Why These Distinctions Matter for ORPO

The goal of ORPO is to **rank-shift the model distribution** so that even when multiple plausible answers exist, the model consistently favors the one that is both **technically correct** and **practically useful**. In technical documentation adaptation, that often means preferring a slightly longer but precise and actionable answer over a short generic one.

Training triples crafted with these criteria reinforce the model's ability to:

• Avoid hallucinations by rewarding ground-truth alignment.

• Reflect expert problem-solving patterns.

• Produce outputs closer to what a real technical professional would write.

Synthetic Dataset Generation Strategy (Example)

The following is an example approach to generating preference tuning training data from an unstructured corpus with the goal of (essentially) 100% content coverage, i.e. there's a question that covers everything your documentation talks about.

```
For each page in your dataset:
    Generate the list of “main points” covered by the page
    For each main point:
            Generate a “good” question about the main point
            Generate a “good” answer to the “good” question
            Generate a “wrong” answer to the “good” question
            Generate a “false premise” question about the main point
                    - a question based on an intentional misunderstanding in it
            Generate a “corrective” answer to the “false premise” question
                    - an answer that politely but firmly corrects the false premise
            Generate a “sycophantic” answer to the “false premise” question
                    - an answer that assumes the false premise is true
```

This strategy will result in two training triples per main point. The point of the "false premise" question is that LLMs tend to be sycophantic by nature, i.e. if someone asks a question with a false premise in it. For instance, the user may ask "Why is the newer version of your software slower than the older version?", the model will (ever so helpfully) explain why the newer version is slower than the older version, even though it isn't. By specifically training the model to (politely but firmly) correct a false premise every time it sees one, the occurrence rate for sycophantic answers drops dramatically.

Balancing Response Capabilities

The preference dataset must train the model across multiple competencies:

• **Conversational Ability (25%):** Natural interaction with clarifying questions and context awareness

• **Technical Reasoning (30%):** Step-by-step problem decomposition and solution building

• **Factual Accuracy (25%):** Precise technical details and configuration parameters

• **Creative Problem-Solving (20%):** Alternative approaches and workaround strategies

Tool Selection: LlamaFactory

LlamaFactory also emerges as the preferred tool for ORPO implementation due to:

• Simplified configuration management

• Built-in support for various preference optimization algorithms

• Efficient memory usage for large model training

• Comprehensive logging and monitoring capabilities

Why This Matters

Preference-based fine-tuning aligns outputs with accuracy, clarity, safety, and usefulness, typically boosting human satisfaction over instruction-only tuning. In production, that translates to fewer escalations, faster resolution, and better adoption..

Considerations

- **Prerequisite:** The base model has already undergone Supervised Fine-Tuning (SFT). ORPO then ranks preferred responses over dispreferred ones to resolve ambiguity among plausible outputs
- **Diverse, high-quality pairs:** cover ambiguous prompts, false premises, multi-step workflows; programmatic perturbations to scale; validate clean serialization.
- **Balanced coverage:** stratify by domain/task/risk; enforce quotas; track drift; tag slices for per-slice evaluation.
- **Technical accuracy (including synthetic):** ground in versioned specs; embed verifiability (commands/tests); include “catch” items that reward corrective answers.
- **Preserve general capabilities post-SFT:** replay SFT data or alternate SFT/ORPO phases; use canary tasks and early stopping tied to general-cap metrics.
- **Mitigate sycophancy/confident wrongness:** challenge prompts, reward respectful disagreement and caveats; penalize fluent but unverifiable claims; monitor a sycophancy probe set.
- **Avoid style overfitting:** vary format/persona within guardrails; test with paraphrases and shuffled constraints.

Data & Training Best Practices

- **Schema & metadata:** store <prompt, preferred, dispreferred> with domain, difficulty, source, version, rationale, evaluator; preserve provenance.
- **Rubrics:** define “good” (correct, complete, safe, reproducible, version-aware, actionable) and “bad” (omissions, wrong order, unsafe, hallucinated, sycophantic).
- **Curriculum & sampling:** start with high-confidence slices; add harder cases (long context, retrieval, tool use); use active learning to target disagreements.
- **Loss & scheduling:** co-train with SFT; start with higher SFT weight, ramp ORPO; monitor preference margins to avoid saturation.
- **Batching & stability:** mix domains/difficulties; shard by length; gradient clipping, LR schedules; deduplicate and cap lengths; use QLoRA for VRAM limits.

Stage 6: Evaluation Framework Development

Why evaluation matters

Evaluation is the control plane that turns a promising model into a dependable enterprise assistant; without it, production deployments appear capable but fail under constraints like versioning, command correctness, and troubleshooting completeness. Generic benchmarks (e.g., MMLU, HellaSwag) don't test version awareness, executable syntax, or workflow ordering for a specialized model that has internalized new knowledge.

Enterprise-grade evaluation must be grounded in documentation truth and operational reality, measuring not only factuality but whether an answer is usable, safe, and consistent over time. In practice, this means treating evaluation as a first-class system with automated tests, domain rubrics, and targeted expert review.

Custom metrics for specialized domains

A robust harness spans three dimensions that map to real operator needs:

- Technical accuracy
  - Fact verification against documentation ground truth.
  - Configuration parameter correctness and API/CLI validity.
  - Command syntax and flags, including platform/version constraints.
  - Version-specific information handling and deprecation awareness.
- Practical utility
  - Problem-resolution effectiveness and task completion.
  - Troubleshooting completeness and next-step clarity.
  - Safety notices and operational warnings where applicable.
- Consistency
  - Cross-query coherence across paraphrases and near-duplicate queries.
  - Temporal consistency for versioned features and releases.
  - Terminology alignment and tone/style appropriateness.

Quality assurance processes

Effective systems combine automation for coverage with expert review for nuance:

- Automated testing
  - Regression checks on validated Q&A pairs and scenarios.
  - Consistency checks across paraphrases and scenario variants.
  - Performance benchmarking on representative task suites.
  - Continuous monitoring to catch drift and regressions.
- Manual review
  - Expert validation for complex, high-risk responses.
  - Edge-case discovery and annotation for feedback loops.
  - Operator acceptance testing with real workflows.
  - Integration of feedback into data updates and retraining.

Tool selection

DeepEval

In our implementation, bespoke tools and metrics were developed to align tightly with VMware Cloud Infrastructure documentation, achieving fine-grained diagnostics at the cost of significant engineering and domain effort.

For many organizations, that level of custom build-out is unnecessary; frameworks like DeepEval offer a faster, lower-risk path with out-of-the-box metrics and extensibility for domain-specific rules. DeepEval evaluates question–answer pairs using semantic alignment, factual consistency, and relevance scoring, emphasizing whether answers are supported by evidence, which is critical for domain-adapted standalone LLMs that must be correct and usable.

DeepEval: capabilities that matter for standalone domain-adapted LLMs

- **Answer Relevancy:** Scores whether the output directly answers the task without digression; helpful for setup, configuration, and troubleshooting prompts where concision and on-task content are essential.
- **Faithfulness (Hallucination/Contradiction Guard):** Scores whether the answer is supported by authoritative references (gold answers, curated snippets, canonical specs) and flags contradictions; useful to prevent invented commands, flags, or steps. See metric overview and examples.
- **Semantic Similarity:** Rewards paraphrases that preserve meaning vs. a canonical reference answer; useful when a gold answer exists but phrasing may differ. See metric overview and examples.
- **GEval (LLM-as-a-Judge, Custom Rubrics):** Encodes enterprise rules such as version correctness, command validity, workflow ordering, safety checklist inclusion, and terminology standards. See guidance on building custom criteria and judge prompts.
- **Safety/Bias/Toxicity:** Optional gates to enforce policy compliance and professional tone; useful for production deployments in regulated environments.

Code examples: generator-focused evaluation (no retrieval dependencies)

1. **Task-focused evaluation:** Answer Relevancy + Faithfulness Use when the model must produce a precise, domain-correct answer and you maintain a small authoritative reference (curated snippet or canonical gold answer).

```
# pip install deepeval
from deepeval import evaluate
from deepeval.metrics import AnswerRelevancyMetric, FaithfulnessMetric
from deepeval.test_case import LLMTestCase

test_case = LLMTestCase(
    input="List the required ports for vCenter to ESXi host management on vSphere 8.0.",
    actual_output="TCP 902 and 443 are required; 902 for hostd, 443 for vCenter
connectivity.",
    # Curated snippet or spec excerpt for faithfulness:
    retrieval_context=[
        "For vSphere 8.0, ESXi management uses TCP 902 (hostd/management) and TCP 443 (HTTPS
vCenter communication)."
    ],
    expected_output="TCP 902 and 443 are required for vCenter to ESXi host management on
vSphere 8.0."
)

metrics = [
    AnswerRelevancyMetric(),             # See: metrics overview
    FaithfulnessMetric(include_reason=True)  # See: metrics overview
]

results = evaluate(test_cases=[test_case], metrics=metrics)
print(results)
```

2. **Version-aware correctness via rubric (GEval):** Encodes rules where the “right” answer depends on the requested product/version and on excluding deprecated flags or features.

```
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCase, LLMTestCaseParams
from deepeval import evaluate

version_correctness = GEval(
    name=”VersionAwareCorrectness”,
    evaluation_steps=[
        “Verify the answer explicitly references the requested product and version.”,
        “Penalize mention of deprecated flags/features for the requested version.”,
        “Reward precise terminology and parameter names used in that version.”
    ],
    evaluation_params=[
        LLMTestCaseParams.INPUT,
        LLMTestCaseParams.ACTUAL_OUTPUT,
        LLMTestCaseParams.RETRIEVAL_CONTEXT,  # optional curated spec snippet
    ],
)

case = LLMTestCase(
    input=”Provide the validated upgrade order for VMware Cloud Foundation 5.1.”,
    actual_output=”Upgrade SDDC Manager, then NSX, then vCenter, then ESXi for VCF 5.1.”,
    retrieval_context=[
        “For VCF 5.1, upgrade sequence: SDDC Manager -> NSX -> vCenter -> ESXi.”
    ]
)

results = evaluate(test_cases=[case], metrics=[version_correctness])
print(results)
```

3. Workflow sequencing rubric (order-sensitive tasks): Ensures multi-step procedures are provided in the documented order and not merely listed.

```
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCase, LLMTestCaseParams
from deepeval import evaluate
workflow_order = GEval(
    name=”WorkflowSequencing”,
    evaluation_steps=[
        “Check that the steps are ordered according to the documented procedure.”,
        “Penalize missing prerequisites or skipped validation steps.”,
        “Reward explicit warnings before destructive operations.”
    ],
    evaluation_params=[
        LLMTestCaseParams.ACTUAL_OUTPUT,
        LLMTestCaseParams.RETRIEVAL_CONTEXT,  # curated procedure excerpt
    ],
)
case = LLMTestCase(
    input=”Document the validated upgrade workflow for component X.”,
    actual_output=(
        “1) Backup configuration\n”
        “2) Place hosts in maintenance mode\n”
        “3) Apply upgrade package\n”
        “4) Validate services and exit maintenance mode”
    ),
    retrieval_context=[
        “Validated workflow:\n1) Backup configuration\n2) Maintenance mode\n3) Apply upgrade\
n4) Validation”
    ],
)
print(evaluate(test_cases=[case], metrics=[workflow_order]))
```

4. Command validity and hallucination guard: Flags invented cmdlets/flags and rewards executable, version-appropriate syntax.

```
from deepeval import evaluate
from deepeval.metrics import FaithfulnessMetric, GEval
from deepeval.test_case import LLMTestCase, LLMTestCaseParams

cmd_faithfulness = FaithfulnessMetric(include_reason=True)

cmd_realism = GEval(
    name="CommandValidity",
    evaluation_steps=[
        "Penalize invented cmdlets or flags absent from the canonical reference.",
        "Reward correct quoting, parameter names, and expected output formatting.",
        "Penalize ambiguous placeholders if concrete examples are required."
    ],
    evaluation_params=[
        LLMTestCaseParams.ACTUAL_OUTPUT,
        LLMTestCaseParams.RETRIEVAL_CONTEXT,  # canonical command reference excerpt
    ],
)

case = LLMTestCase(
    input="Show PowerCLI to list vSAN disk groups on vSphere 8.0.",
    actual_output="Get-VsanDiskGroup -Cluster 'ProdCluster'",  # assume invalid cmdlet
    retrieval_context=[
        "Use Get-VsanView or esxcli vsan storage list; 'Get-VsanDiskGroup' is not a valid
cmdlet."
    ],
)

print(evaluate(test_cases=[case], metrics=[cmd_faithfulness, cmd_realism]))
```

5. **CI-style regression harness:** Blocks deployment when scores drop below thresholds for core dimensions.

```
# test_domain_eval.py
import pytest
from deepeval import evaluate
from deepeval.metrics import AnswerRelevancyMetric, FaithfulnessMetric
from deepeval.test_case import LLMTestCase

gold_cases = [
    LLMTestCase(
        input="Ports required for vCenter  ESXi management on vSphere 8.0?",
        actual_output="TCP 902 and 443.",
        retrieval_context=["vSphere 8.0 uses 902 for ESXi management and 443 for HTTPS."],
        expected_output="TCP 902 and 443."
    ),
    # add more canonical tasks for your domain
]

def test_regression():
    metrics = [AnswerRelevancyMetric(), FaithfulnessMetric()]
    results = evaluate(test_cases=gold_cases, metrics=metrics)
    assert all(r.score >= 0.8 for r in results), "Evaluation regression detected"
```

Best practice

Framework-first, then customize. Start with built-in metrics (relevancy, faithfulness, semantic similarity) to establish a reliable baseline for a standalone domain-adapted model; then layer targeted, high-value custom checks tied to enterprise workflows and versioning. This achieves strong coverage rapidly, while reserving bespoke engineering for cases where it most improves production outcomes.

Why this matters

Evaluation is how production risk is managed: it provides confidence in model behavior, reveals gaps for iterative improvement, and sets baselines for future enhancements—turning a capable demo into a dependable system.