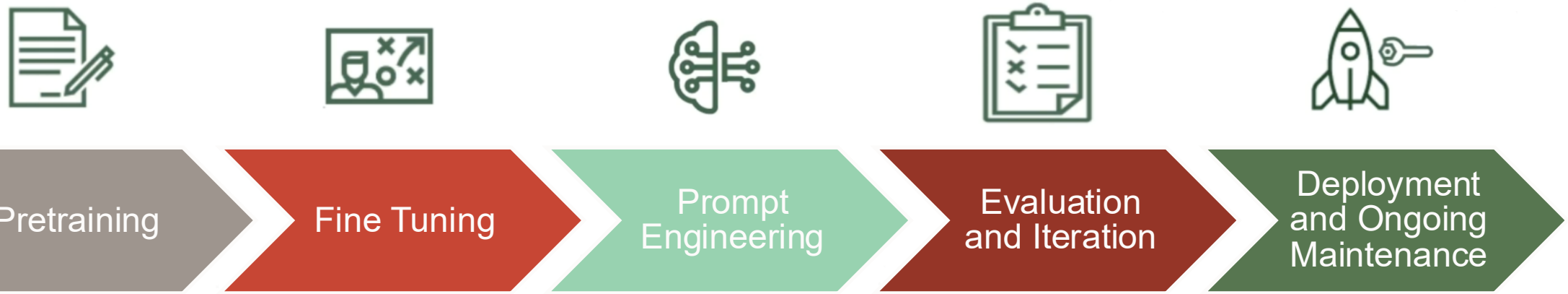
LLMs Fine Tuning and Deployment

Ram N Sangwan

- Large Language Model Lifecycle
- Fine Tuning
- RAG
- Deployment on OCI Dedicated AI Cluster

LLM Lifecycle



General Procedure for Fine-Tuning

Data Preparation

Collect a dataset that represents the task you want the model to perform.

Format the data in a JSONL format where each line is a JSON object.

Model Selection

Choose a pre-trained model that aligns with your task

Fine-Tuning

Use the API or Dashboard.

Provide the dataset and configure the parameters.

Evaluation

Evaluate the model's performance using a separate validation dataset.

Make adjustments to the fine-tuning parameters if necessary, and retrain if needed.

Deployment

Deploy the fine-tuned model for inference.

Monitor the performance and collect feedback for potential further fine-tuning.

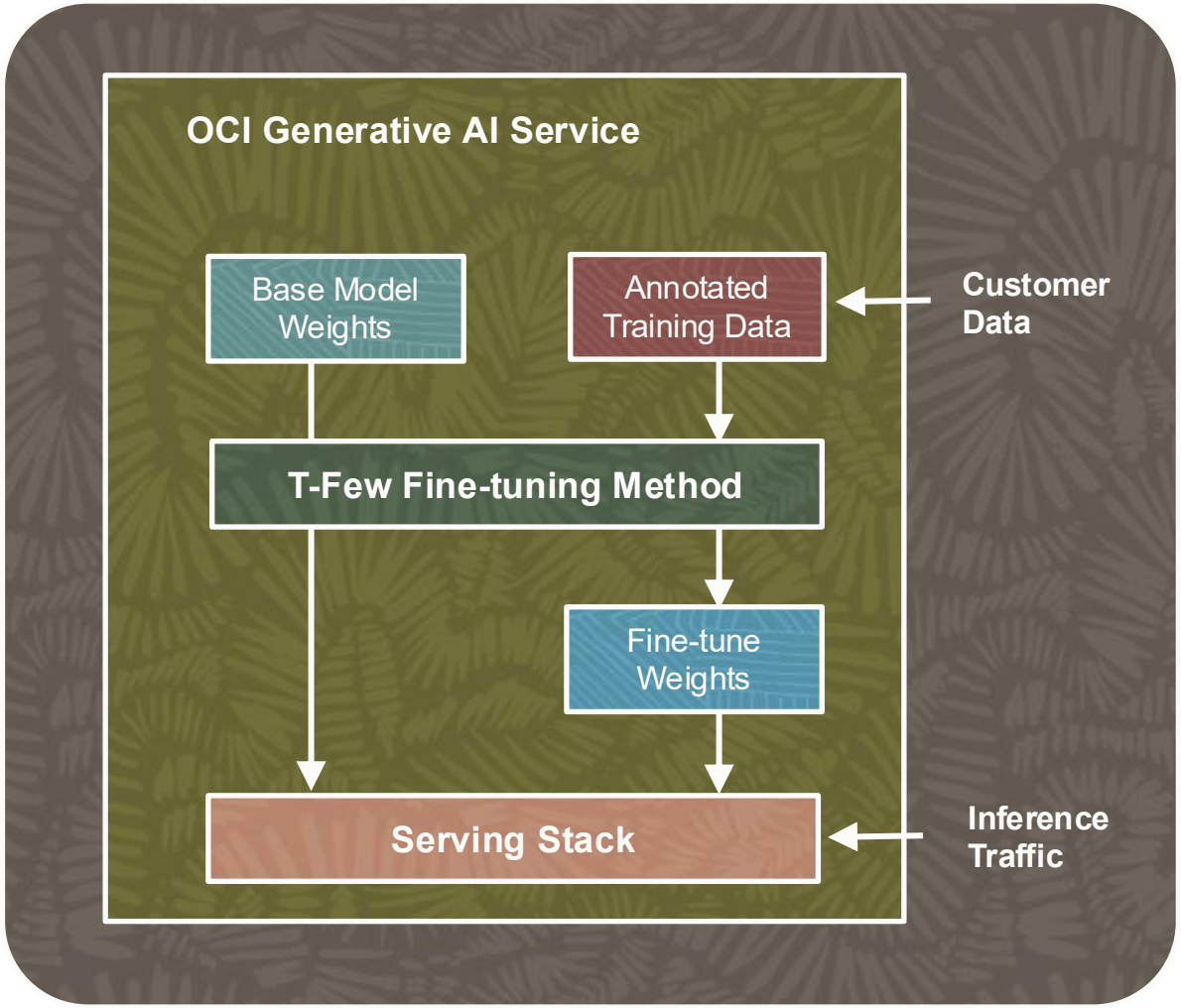
Considerations

- **Ethical Use**
- **Data Privacy**
- **Resource Allocation**

T-Few Fine-tuning

- Traditionally, Vanilla fine-tuning involves updating the weights of all (most) the layers in the model, requiring longer training time and higher serving (inference) costs.
- T-Few fine-tuning selectively updates ***only a fraction of the model's weights***.
  - T-Few fine tuning is an additive Few-Shot Parameter Efficient Fine Tuning (PEFT) technique that inserts additional layers, comprising ~0.01% of the baseline model's size.
  - The weight updates are localized to the T-Few layers during the fine-tuning process.
  - Isolating the weight updates to these T-Few layers significantly reduces the overall training time and cost compared to updating all layers.

T-Few fine-tuning process



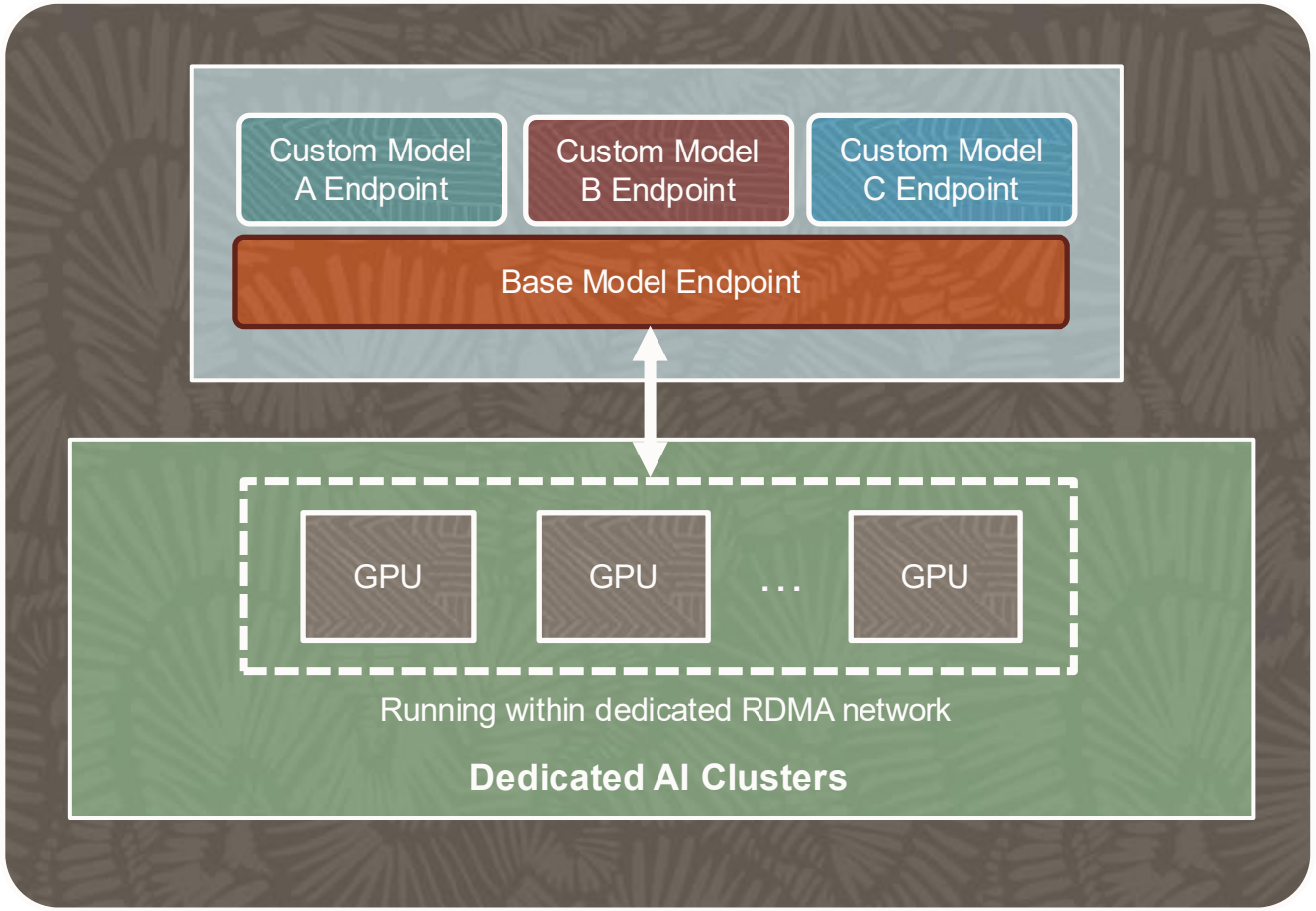
T-Few fine-tuning process begins by utilizing the initial weights of the base model and an annotated training dataset.

Annotated data comprises of input-output pairs employed in supervised training.

Supplementary set of model weights is generated (**~ 0.01% of the baseline model's size**).

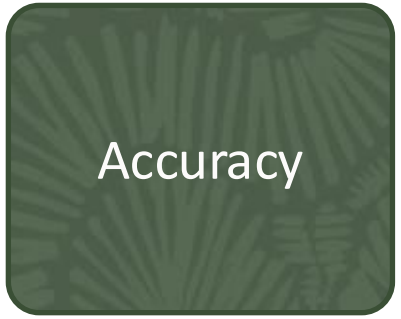
Updates to the weights are confined to a specific group of transformer layers, (T-Few transformer layers), saving substantial training time and cost.

Reducing Inference costs



- **Inference is computationally expensive**.
- Each Hosting cluster can host one Base Model Endpoint and up to N Fine-tuned Custom Model Endpoints serving requests concurrently.
- This approach of models **sharing the same GPU resources** reduces the expenses associated with inference.
- Endpoints can be deactivated to stop serving requests and re-activated later.

Understanding Fine-tuning Results

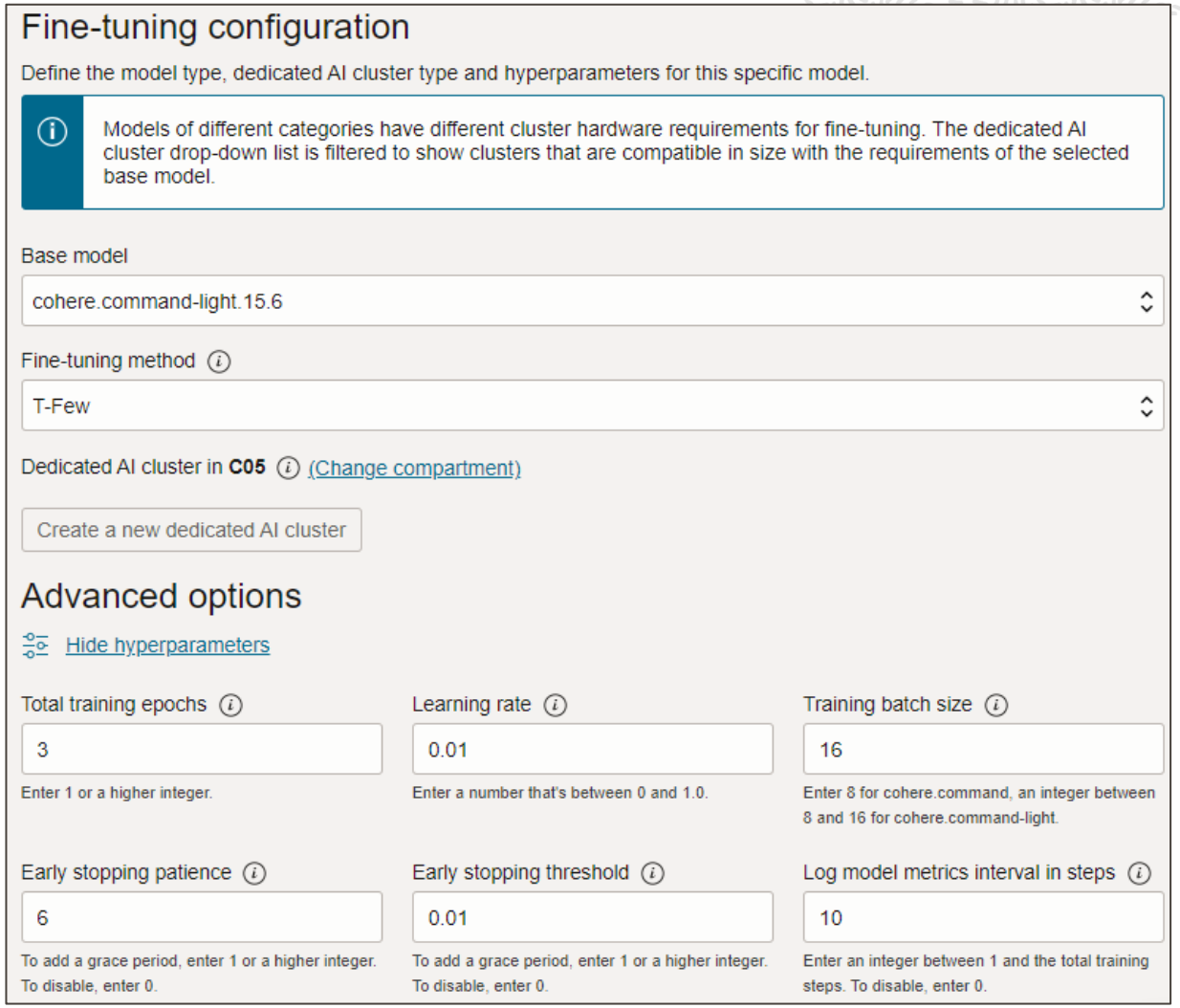


- Accuracy is a measure of how many predictions the model made correctly out of all the predictions in an evaluation.
- To evaluate LLMs for accuracy, we ask it to predict certain words in the user-uploaded data.

- Loss is a measure that describes how bad or wrong a prediction is.
- Accuracy may tell you how many predictions the model got wrong, but it will not describe how incorrect the wrong predictions are.
- To evaluate LLMs for loss, we ask the model to predict certain words in the user-provided data and evaluate how wrong the incorrect predictions are.
- Loss should decrease as the model improves.

Fine-tuning Configuration

- Training Methods
  - Vanilla: Traditional fine-tuning method
  - T-Few: Efficient fine-tuning method
- Hyperparameters
  - Total Training Epochs
  - Learning Rate
  - Training Batch Size
  - Early Stopping Patience
  - Early Stopping Threshold
  - Log Model metrics interval in steps



Fine-tuning Parameters (T-Few)

Hyperparameter	Description	Default value
Total Training Epochs	The number of iterations through the entire training dataset; for example, 1 epoch means that the model is trained using the entire training dataset one time.	Default (3)
Batch Size	The number of samples processed before updating model parameters	8 (cohere.command), an integer between 8 -16 for cohere.command-light
Learning Rate	The rate at which model parameters are updated after each batch	Default (0.1)
Early stopping threshold	The minimum improvement in loss required to prevent premature termination of the training process	Default (0.01)
Early stopping patience	The tolerance for stagnation in the loss metric before stopping the training process	Default (6)
Log model metrics interval in steps	Determines how frequently to log model metrics. Every step is logged for the first 20 steps and then follows this parameter for log frequency.	Default (10)

RAG Framework

Retriever

Function: Sources relevant information from a large corpus or database

Working: Uses retrieval techniques

Purpose: Provides the LLM with contextually relevant, accurate, and up-to-date information that might not be present in the model’s pre-trained knowledge

Ranker

Function: Evaluates and prioritizes the information retrieved by the retriever

Working: Uses various algorithms to evaluate the quality of the retrieved content

Purpose: Ensures that the LLM receives the most pertinent and high-quality input

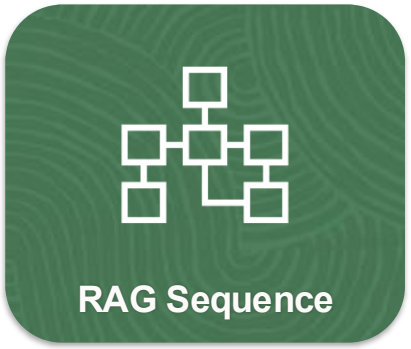
LLM

Function: Generates human-like text based on the retrieved and ranked information, along with the user's original query it receives

Working: Uses LLM to craft human-like text that is contextually relevant, coherent, and informative

Purpose: Ensures that the final response is factually accurate, relevant, and also coherent, fluent, and styled typically like human language

RAG Techniques



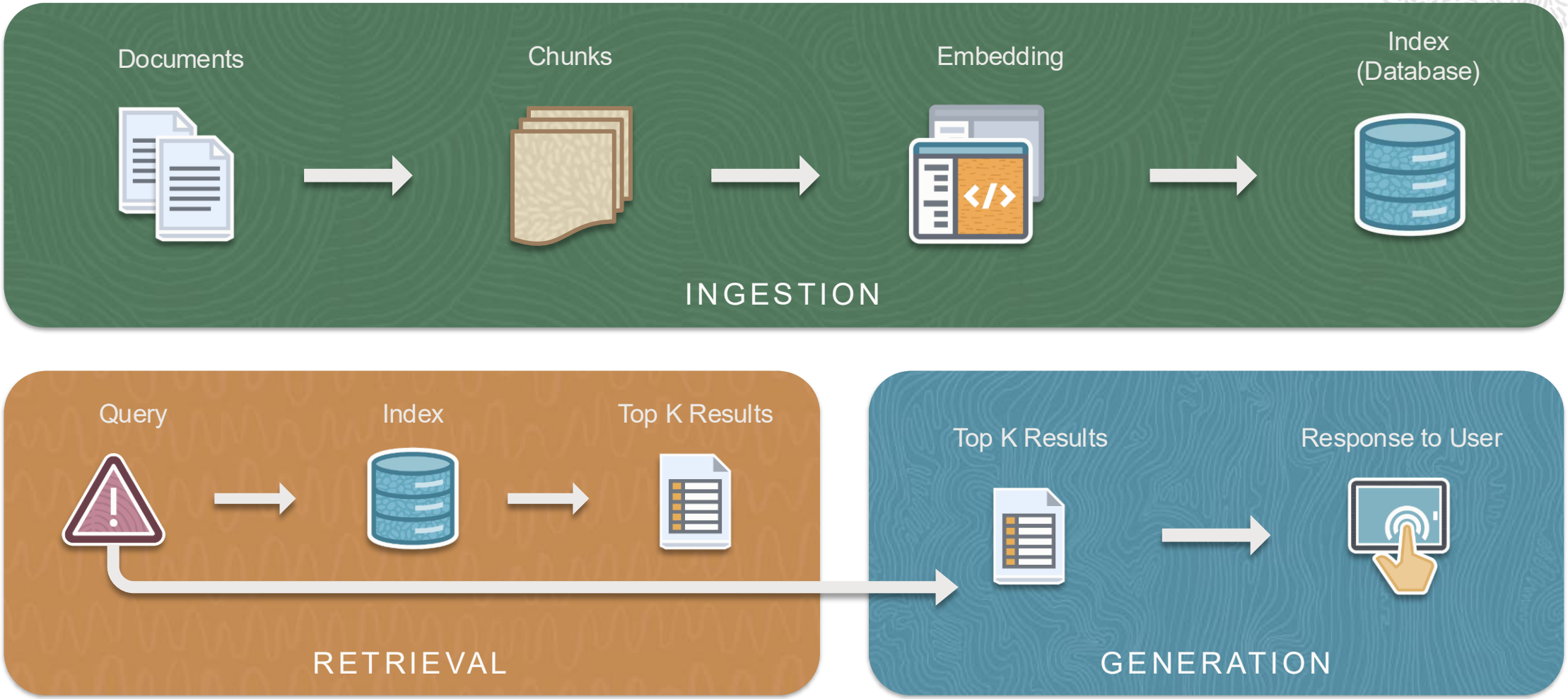
For each input query (like a chapter topic), the model retrieves a set of relevant documents or information.

It then considers all these documents together to generate a single, cohesive response (the entire chapter) that reflects the combined information.

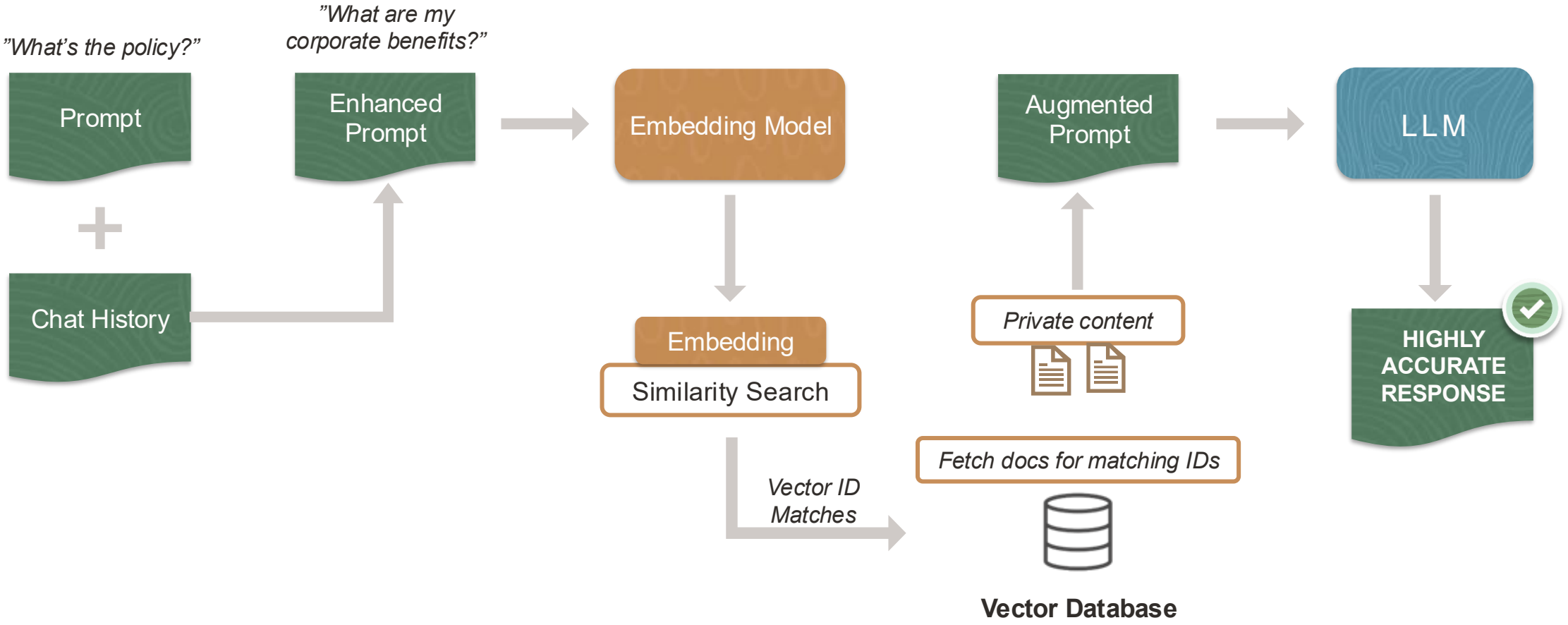
For each part of the response (like each sentence or even each word), the model retrieves relevant documents.

The response is constructed incrementally, with each part reflecting the information from the documents retrieved for that specific part.

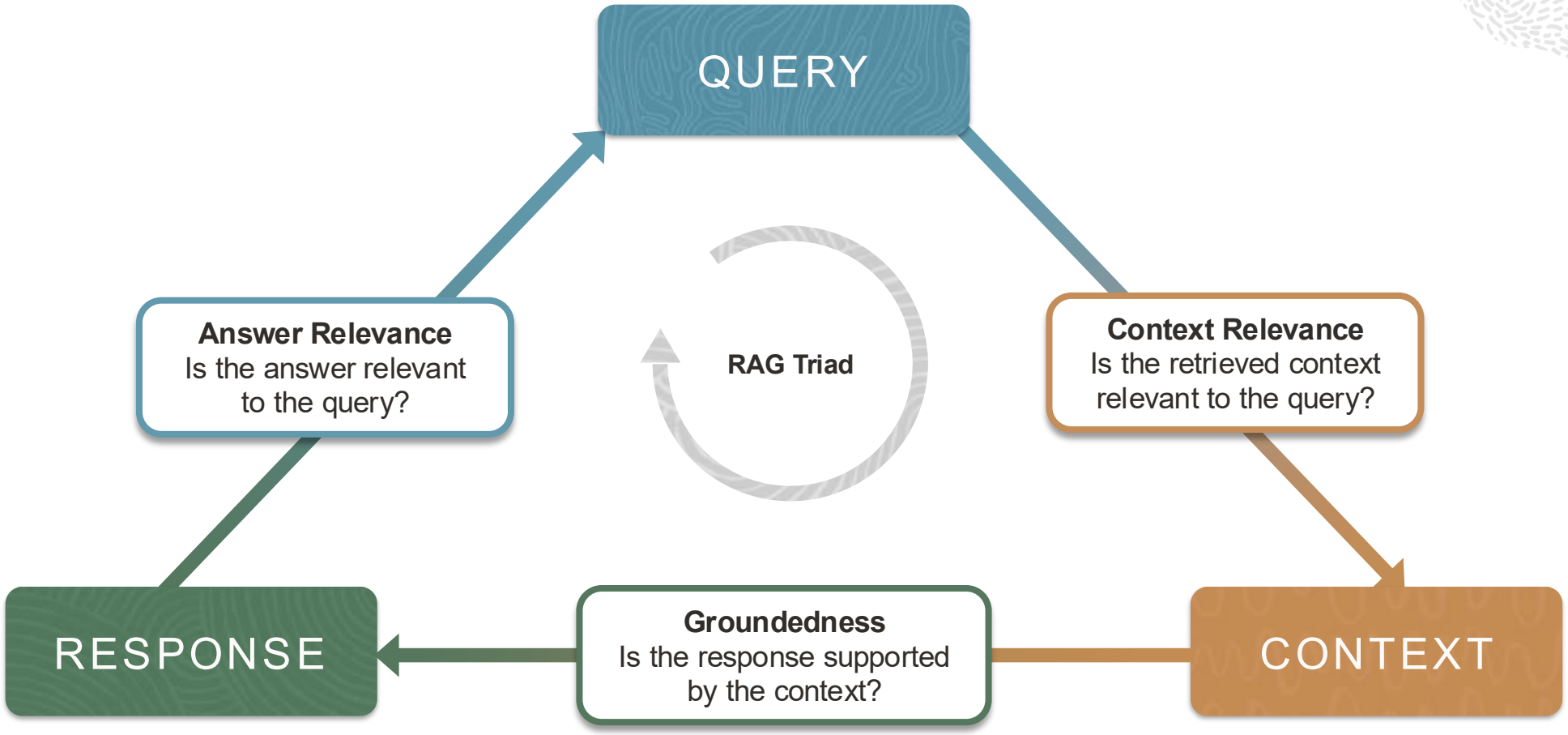
RAG Pipeline



RAG Application



RAG Evaluation



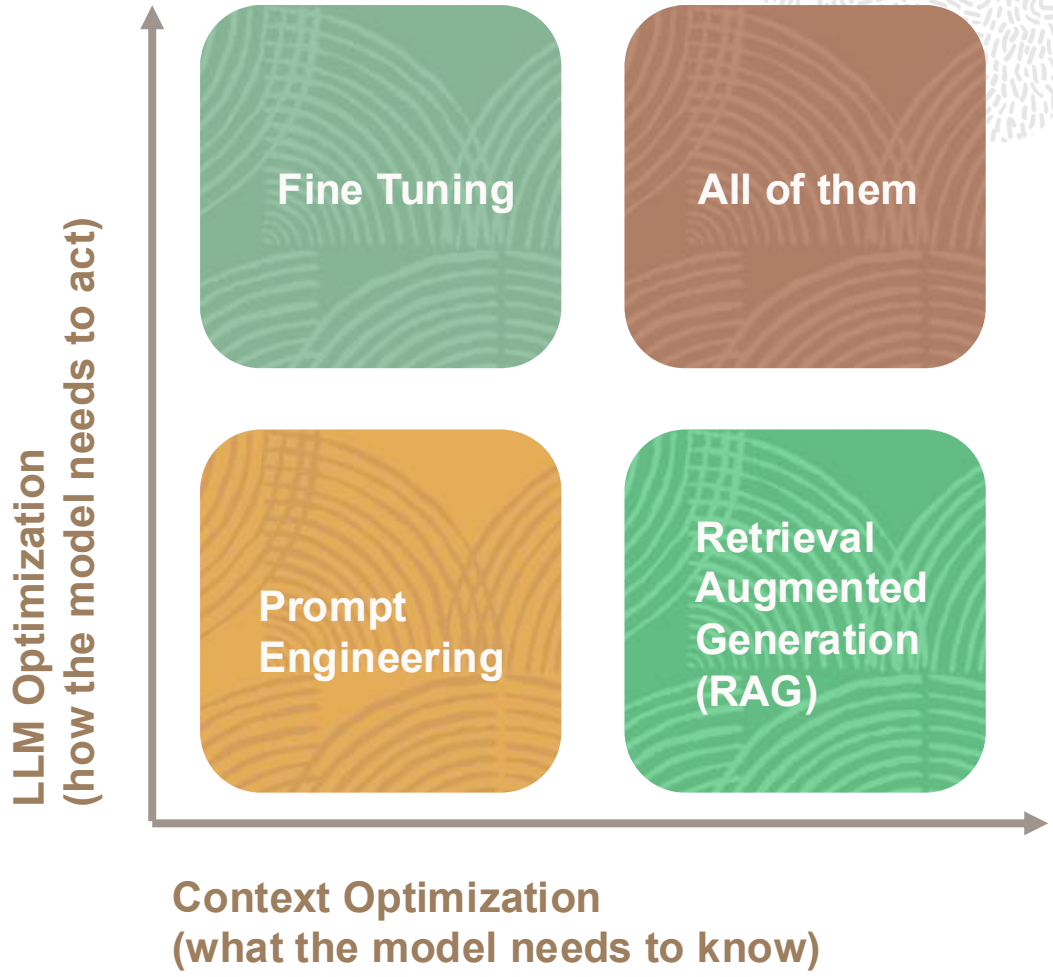
Source: truera; https://truera.com/ai-quality-education/generative-ai-rags/what-is-the-rag-triad/

Customize LLMs with your data

Method	Description	When to use?	Pros	Cons
Few shot Prompting	Provide examples in the prompt to steer the model to better performance	LLM already understands topics that are necessary for the text generation	Very simple No training cost	Adds latency to each model request
Fine-tuning	Adapt a pretrained LLM to perform a specific task on private data	LLM does not perform well on a particular task Data required to adapt the LLM is too large for prompt engineering Latency with the current LLM is too high	Increase in model performance on a specific task No impact on model latency	Requires a labeled dataset which can be expensive and time-consuming to acquire
RAG	Optimize the output of a LLM with targeted information without modifying the underlying model itself	When the data changes rapidly When you want to mitigate hallucinations by grounding answers in enterprise data (improve auditing)	Access the latest data Grounds the result Does not require fine-tuning jobs	More complex to setup Requires a compatible data source

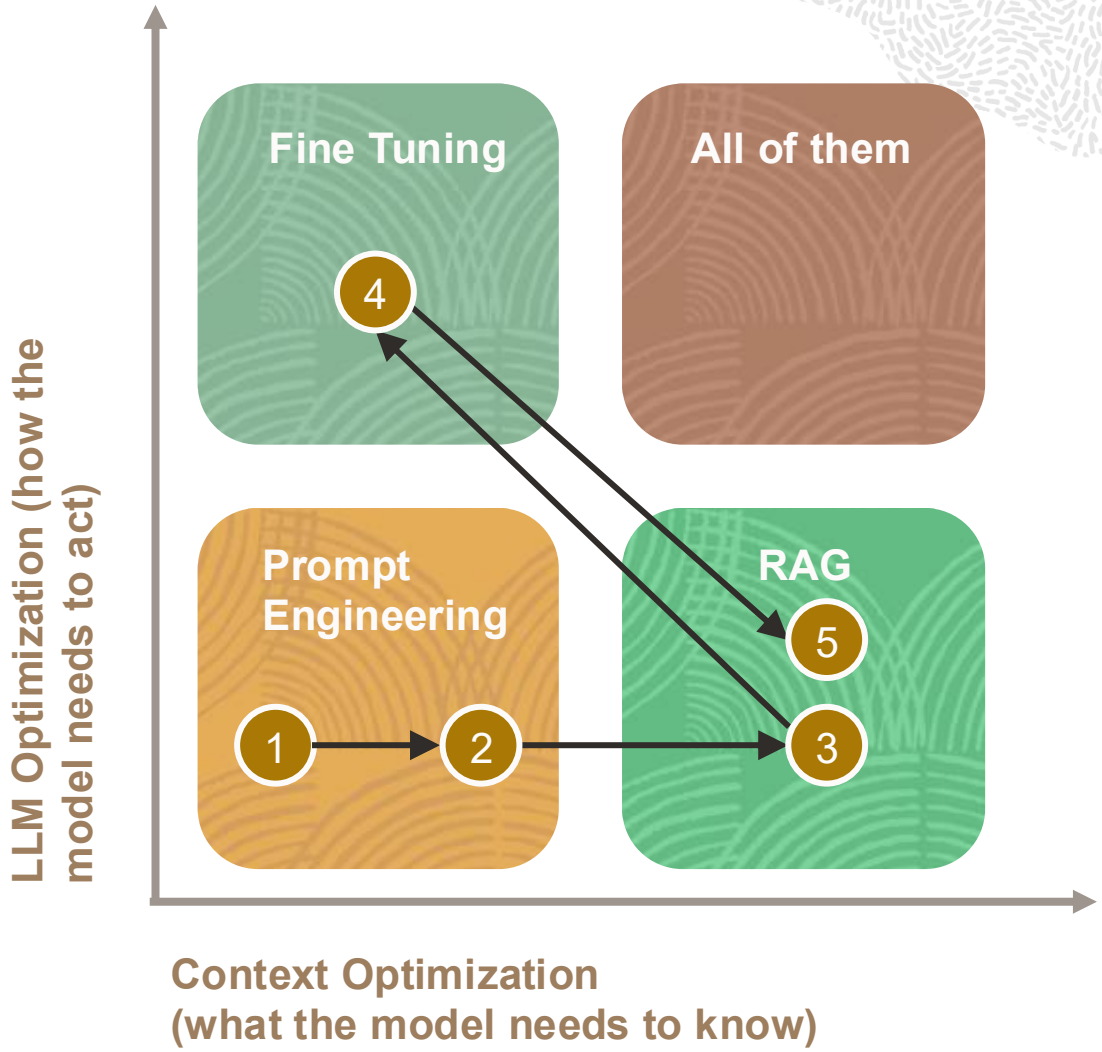
Customize LLMs with your data

- Prompt Engineering is the easiest to start with; test and learn quickly.
- If you need more context, then use Retrieval Augmented Generation (RAG).
- If you need more instruction following, then use Fine-tuning.



Customize LLMs with your data

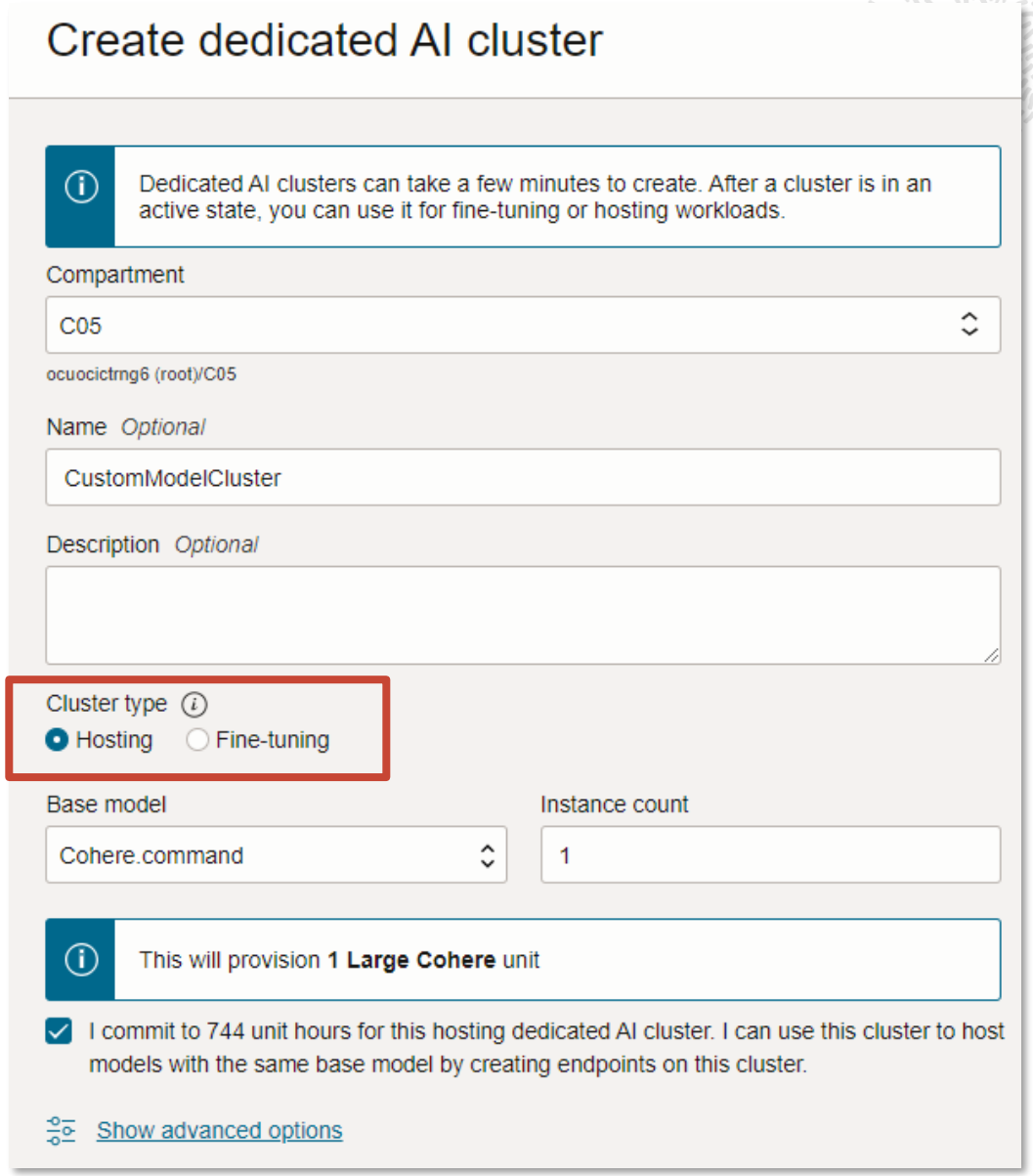
1. Start with a simple Prompt.
2. Add Few shot Prompting.
3. Add simple retrieval using RAG.
4. Fine-tune the model.
5. Optimize the retrieval on fine-tuned model.



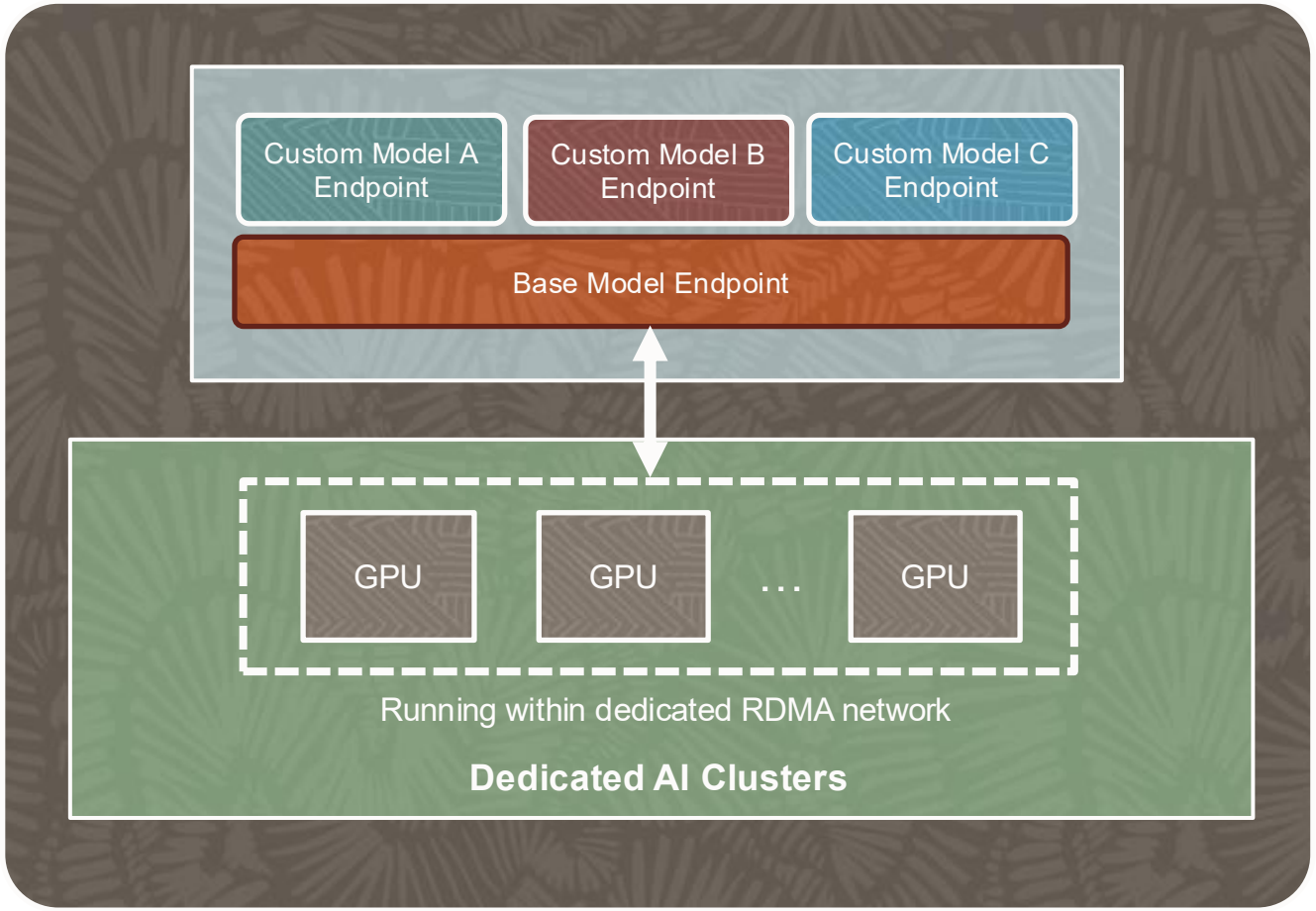
Deployment Options for OCI

Dedicated AI Clusters

- Effectively a single-tenant deployment where the GPUs in the cluster only host your custom models.
- Since the model endpoint isn't shared with other customers, the model throughput is consistent.
- The minimum cluster size is easier to estimate based on the expected throughput.
- Cluster Types
  - **Fine-tuning:** used for *training* a pretrained foundational model
  - **Hosting**: used for hosting a custom model endpoint for *inference*

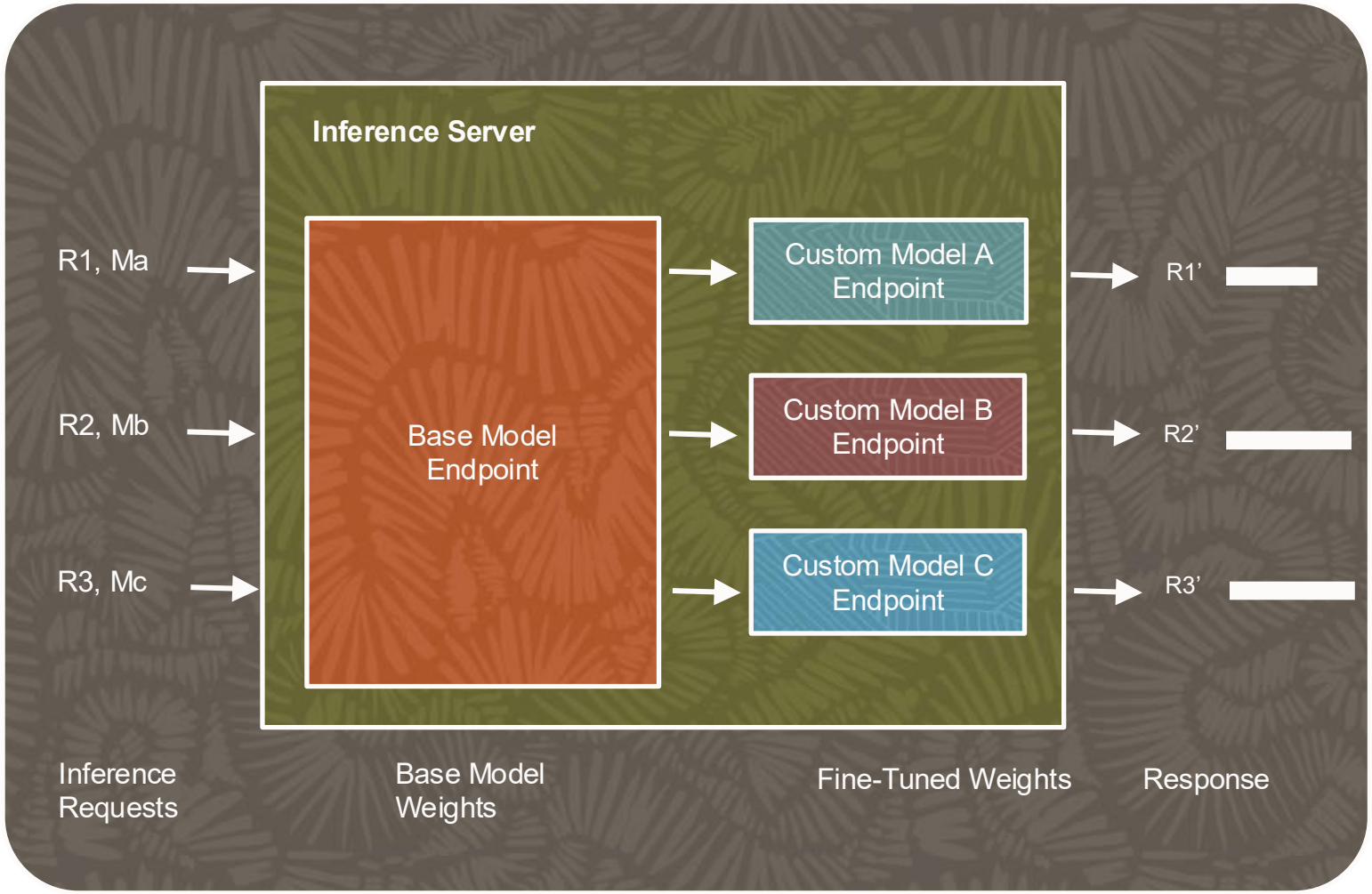


Reducing Inference costs



- **Inference is computationally expensive**.
- Each Hosting cluster can host one Base Model Endpoint and up to N Fine-tuned Custom Model Endpoints serving requests concurrently.
- This approach of models **sharing the same GPU resources** reduces the expenses associated with inference.
- Endpoints can be deactivated to stop serving requests and re-activated later.

Inference serving with minimal overhead



GPU memory is limited, so **switching between models can incur significant overhead** due to reloading the full GPU memory.

These models share the majority of weights, with only slight variations; can be efficiently deployed on the same GPUs in a dedicated AI cluster.

This architecture results in **minimal overhead when switching between models** derived from the same base model.

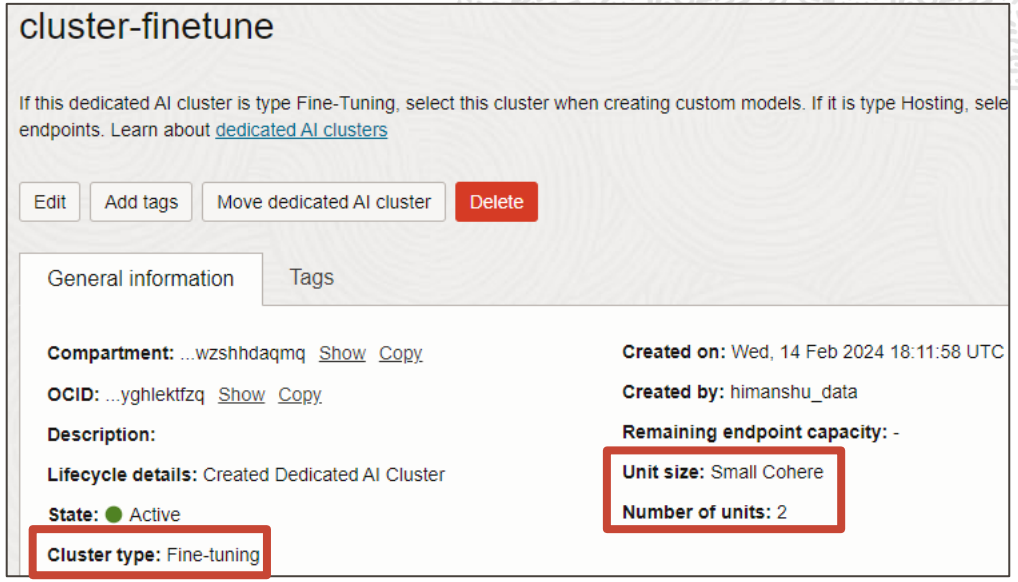
Dedicated AI Clusters Sizing

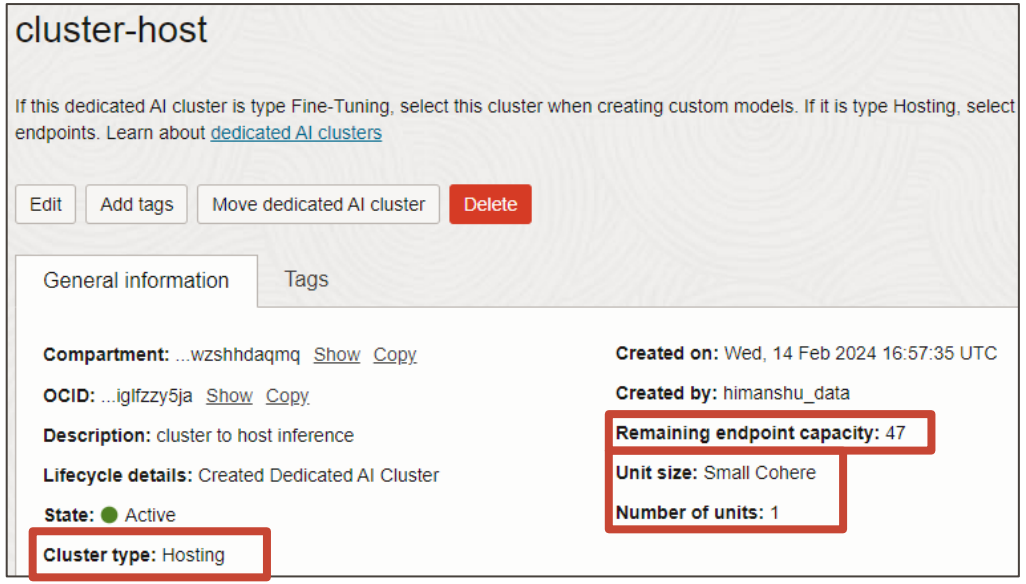
Fine-tuning Dedicated AI Cluster

- Requires **two units** for the base model chosen.
- Fine-tuning a model requires more GPUs than hosting a model (therefore, two units).
- The same fine-tuning cluster can be used to fine-tune several models.

Hosting Dedicated AI Cluster

- Requires **one unit** for the base model chosen.
- Same cluster can host up to 50 different fine-tuned models (using T-Few fine tuning).
- Can create up to 50 endpoints that point to the different models hosted on the same hosting cluster.





Example Pricing

Bob wants to fine-tune a Cohere command (cohere.command) model and after fine-tuning, host the custom models:

- Bob creates a fine-tuning cluster with the preset value of two Large Cohere units.
- The fine-tuning job takes five hours to complete.
- Bob creates a fine-tuning cluster every week.
- Bob creates a hosting cluster with the one Large Cohere unit.

Minimum Commitment

Min Hosting commitment: 744 unit-hours/cluster
Min Fine-tuning commitment: 1 unit-hour/fine-tuning job

Unit Hours for each Fine-tuning

Each fine-tuning cluster requires two units and each cluster is active for five hours
fine-tuning per cluster= 10 unit-hours

Fine-tuning Cost

Fine-tuning cost/month =
(10 unit-hours)/week x (4 weeks) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

Hosting Cost

Hosting cost/month =
(744 unit-hours) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

Total Cost

Total cost/month =
(40 + 744 unit-hours) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

Thank You