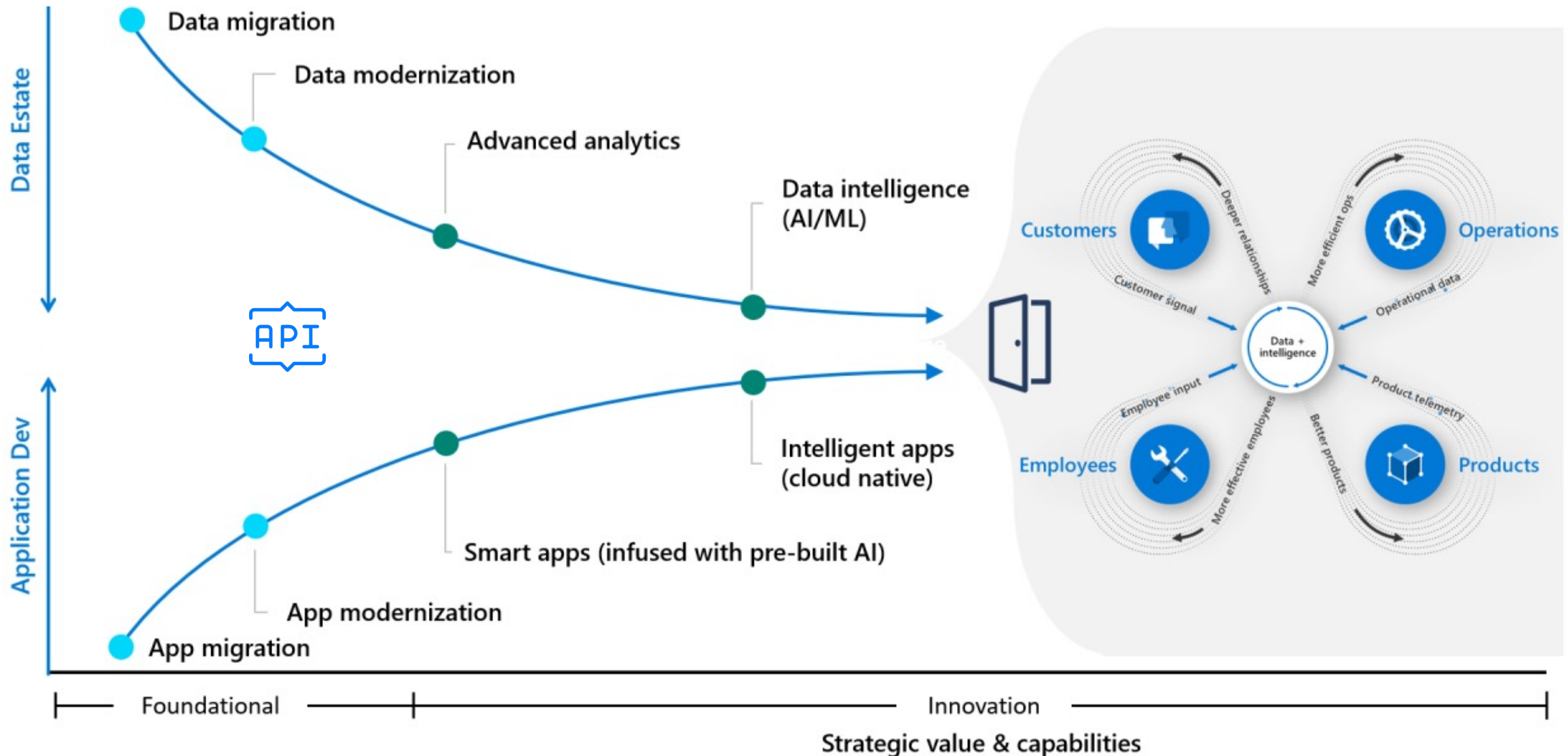
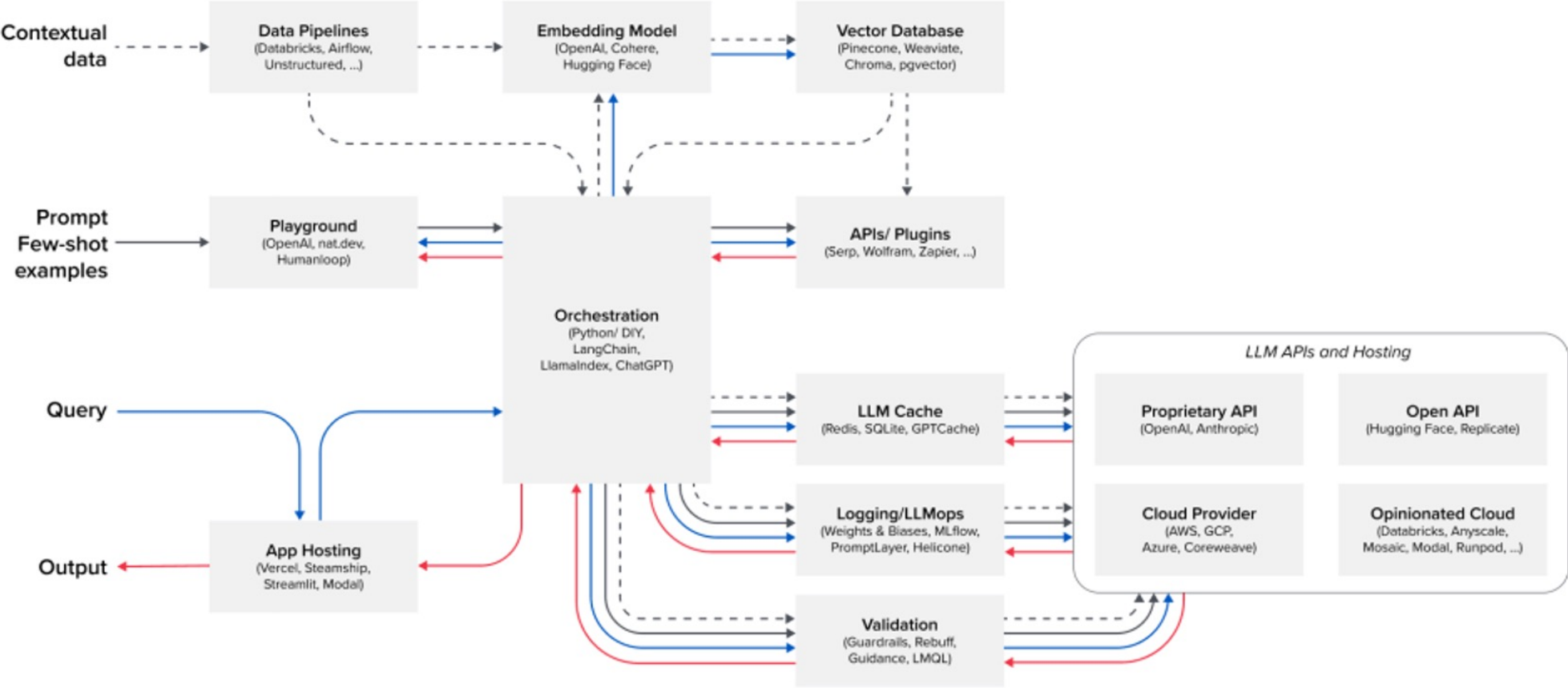


Continuum to Unlock Digital Innovation

- Modernization
- Digital Transformation



MLOPS Emergency App Stack



LEGEND

Gray boxes show key components of the stack, with leading tools/systems listed

Arrows show the flow of data through the stack

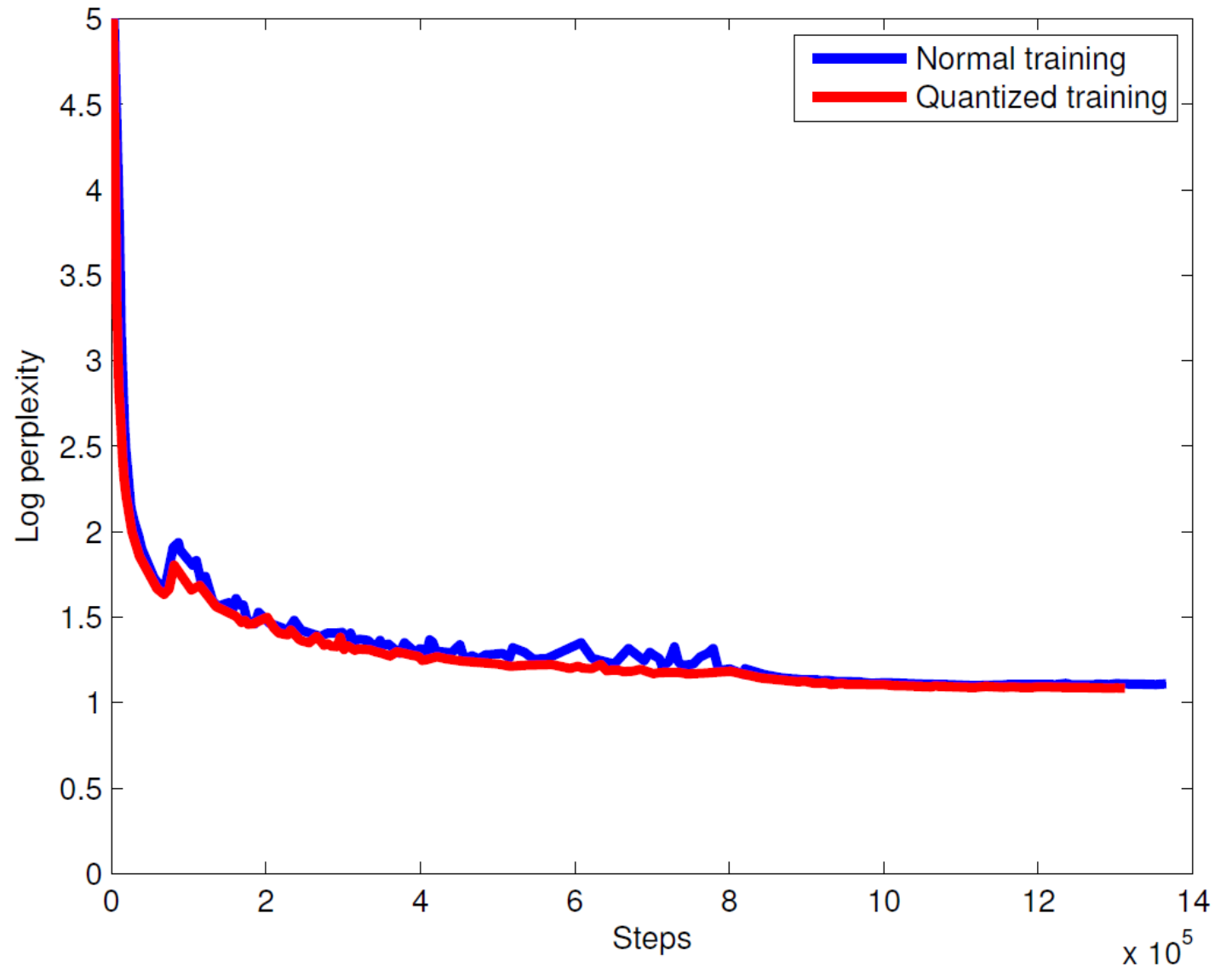
- - - -> Contextual data provided by app developers to condition LLM outputs
- > Prompts and few-shot examples that are sent to the LLM
- > Queries submitted by users
- > Output returned to users



Model Quantization

Model Quantization

- **Model Quantization:** Reducing high-precision floating point arithmetic to low-precision integer arithmetic (approximation). For matrix operations etc.
- **Challenge:** Amplification of quantization (approximation) error as you go deeper into the network.
- **Solution:** Add additional model constraints while training. This ensures the quantization error is small.



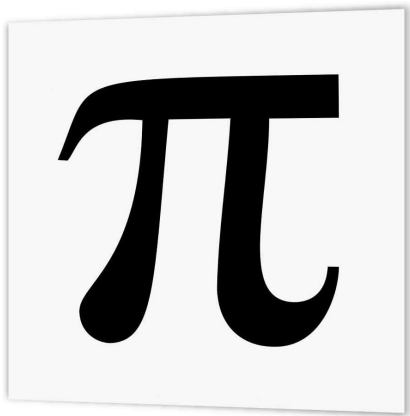
Model quantization is a technique used in Large Language Models (LLMs) to reduce the size of the model while maintaining its accuracy.

This is achieved by representing the model's weights and activations with smaller data types, **such as integers or fixed-point numbers, instead of floating-point numbers.**

Why Quantize?

Quantizing models can bring several benefits:

- 1.Reduced memory usage:** By using smaller data types, the model requires less memory to store its weights and activations.
- 2.Faster inference:** Quantized models can be executed more efficiently on hardware accelerators, such as GPUs or TPUs, which are optimized for integer arithmetic.
- 3.Improved deployment:** Smaller models can be deployed on devices with limited memory or processing power, making them suitable for edge computing applications.



Llama 3.1 Model Sizes

8B, 70B, and 405B parameter size

FP16

3.1415926535 8979323846 2643383279
5028841971 6939937510 5820974944
5923078164 0628620899 8628034825
3421170679

or

INT 8

3.14

Types of Quantization

There are several types of quantization techniques used in LLMs:

- 1.Integer Quantization (IQ):** Represents weights and activations as integers using a fixed number of bits.
- 2.Fixed-Point Quantization (FPQ):** Similar to IQ, but allows for fractional parts by representing numbers with a specific binary point.
- 3.Dynamic Quantization:** Adjusts the quantization scale based on the input data or model behavior.
- 4.Knowledge Distillation-based Quantization:** Uses a smaller student model trained to mimic the behavior of a larger teacher model.

Quantization Approaches

There are various methods for applying quantization to LLMs:

- 1.Weight-only quantization:** Only quantizes the model's weights, leaving activations as floating-point numbers.
- 2.Activation-aware quantization:** Quantizes both weights and activations using techniques like histogram-based or statistical quantization.
- 3.Quantization-aware training (QAT):** Trains a model with quantized weights and activations during training, rather than applying quantization after training.

Challenges and Future Directions

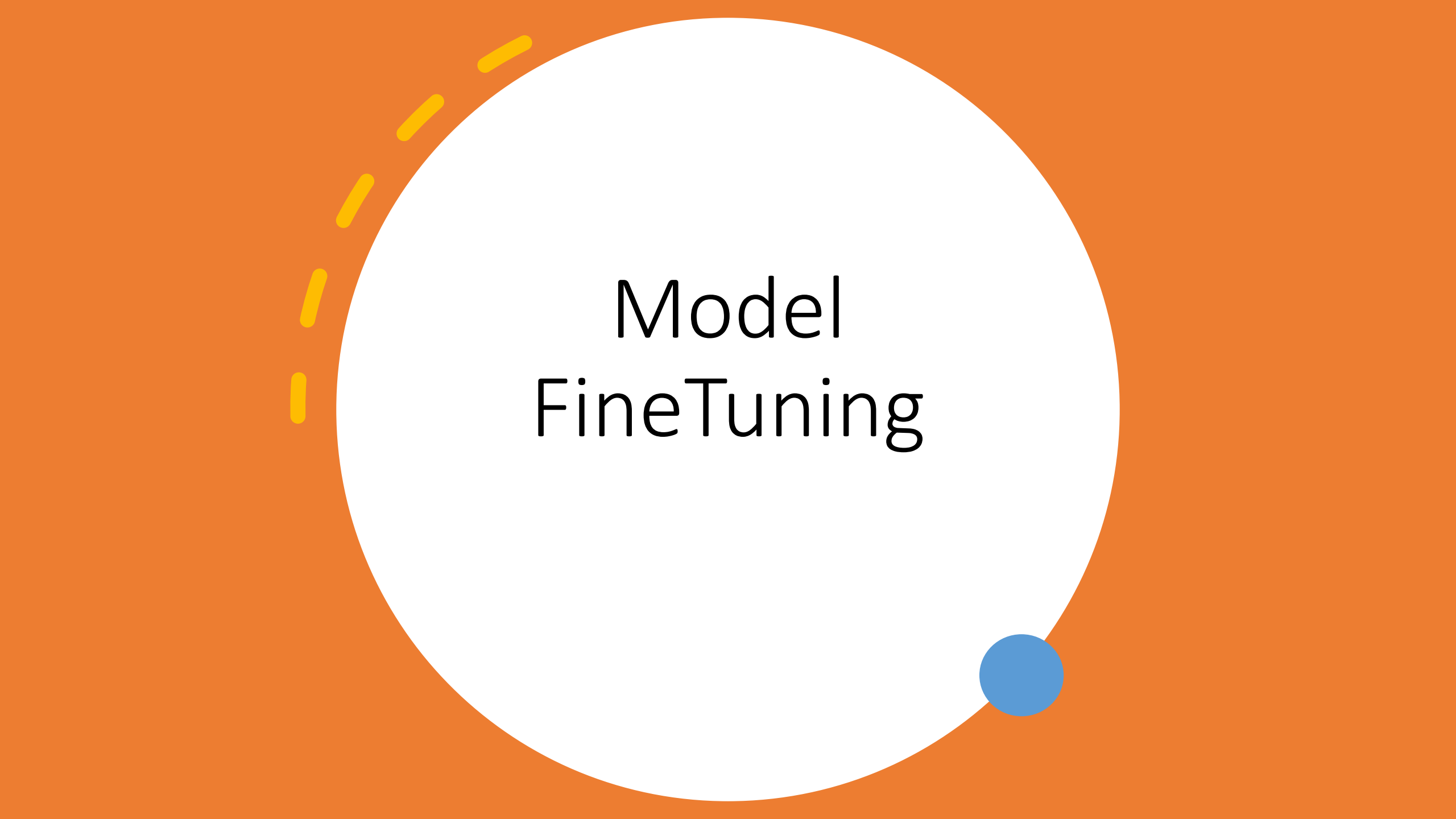
While model quantization is an active area of research, there are still challenges to overcome:

- 1.Accuracy degradation:** Quantizing models can lead to accuracy loss if not done carefully.
- 2.Quantization-aware training:** Requiring significant computational resources and expertise.
- 3.Mixed-precision training:** Balancing between using smaller precision for inference and maintaining larger precision during training.

To address these challenges, researchers are exploring new techniques, such as:

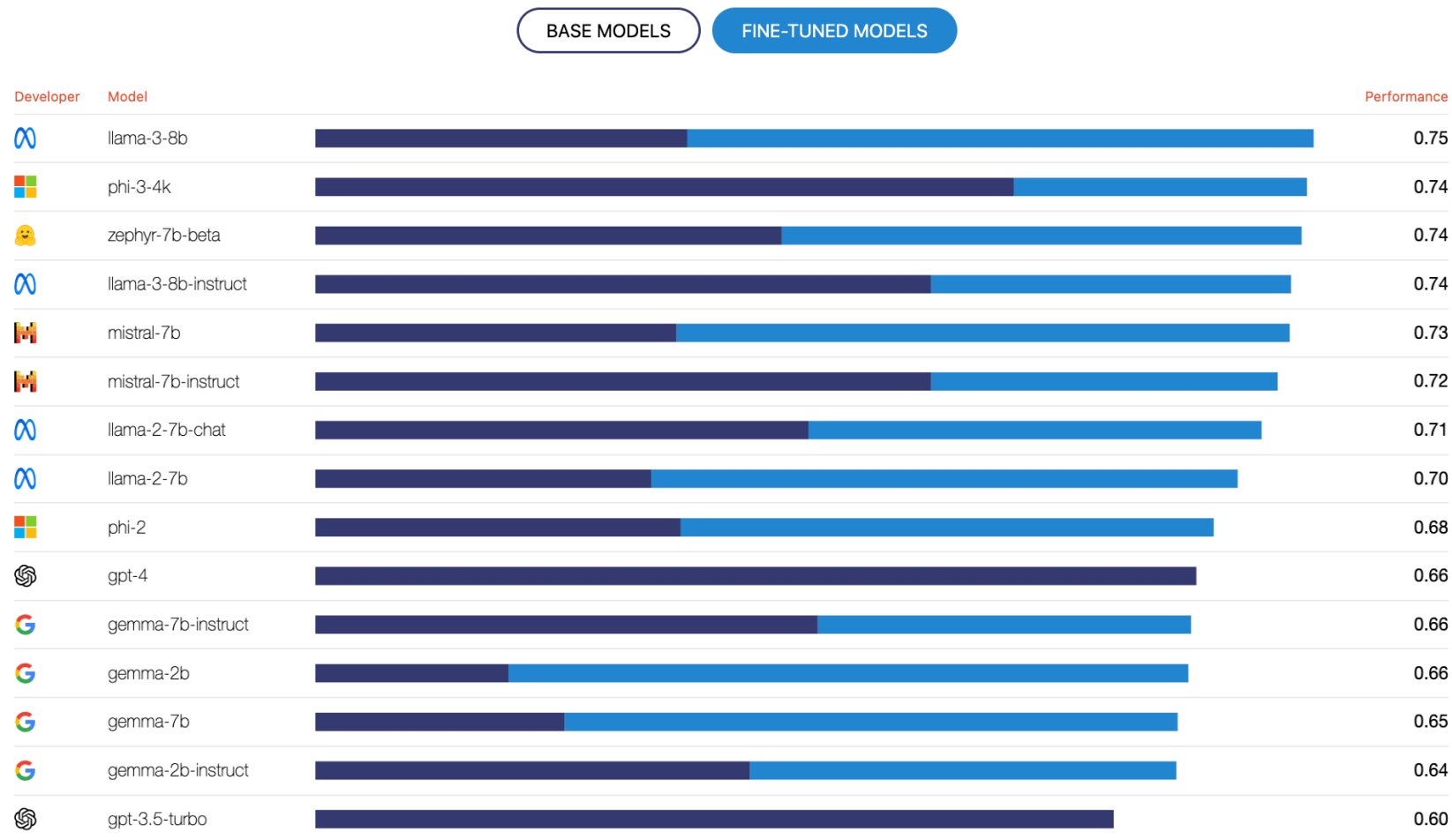
- 1.Efficient neural architecture search (NAS)**
- 2.Quantization-aware pruning**
- 3.Knowledge distillation-based quantization**

These advancements will help improve the performance of LLMs while reducing their computational and memory requirements.



Model FineTuning

The Fine-tuning Leaderboard shows the performance of each model aggregated across 31 distinct tasks. You can evaluate the performance pre and post fine-tuning by selecting the base or fine-tuned model button at the top. Remarkably, most of the fine-tuned open-source models surpass GPT-4 with Llama-3, Phi-3 and Zephyr demonstrating the strongest performance.



<https://predibase.com/fine-tuning-index>

PEFT (parameter-efficient fine-tuning) is a method for training large language models (LLMs) by updating only a small number of parameters during training.

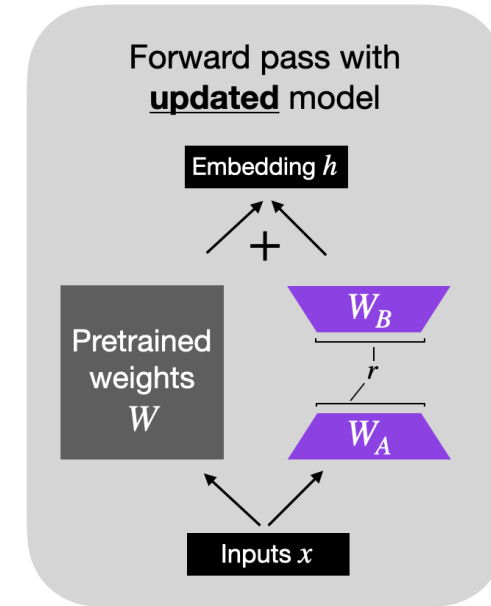
LoRA (Low-rank Adaptation) is a popular PEFT technique that uses low-rank decomposition to **reduce the number of trainable parameters in LLMs**. This reduction in parameters makes fine-tuning more efficient and practical, with lower memory consumption and reduced computational and storage costs

LoRA's approach is to freeze the original weights and introduce new parameters into the model to train through. It does this by decomposing weight matrices into two smaller matrices, which can be trained to adapt to new data while keeping the overall number of changes low.

For example, a weight update matrix of 200×3 and 3×500 can be decomposed into 2100 trainable parameters, which is only 2.1% of the total number of parameters.

LoRA can also be applied to specific layers only, further reducing the number of parameters. As a result, the files can be as small as 8MB, making it much easier and faster to load, apply, and transfer the learned models.

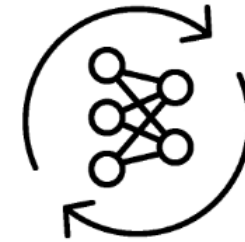
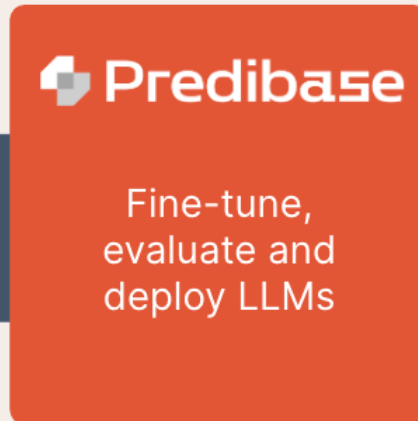
LoRA weights, W_A and W_B , represent ΔW



Smaller, Faster LLMs with Predibase + Gretel



0110
1001
1010
Synthetic
Data



**High-quality models
for your use case**
at a fraction of the cost

From data through deployment

<https://predibase.com/blog/how-to-create-an-sql-copilot-by-fine-tuning-llms-with-synthetic-data>

<https://gretel.ai/videos/how-to-generate-synthetic-data-with-gretel>

<https://github.com/mlabonne/llm-datasets>

High-quality datasets, tools, and concepts for LLM fine-tuning.

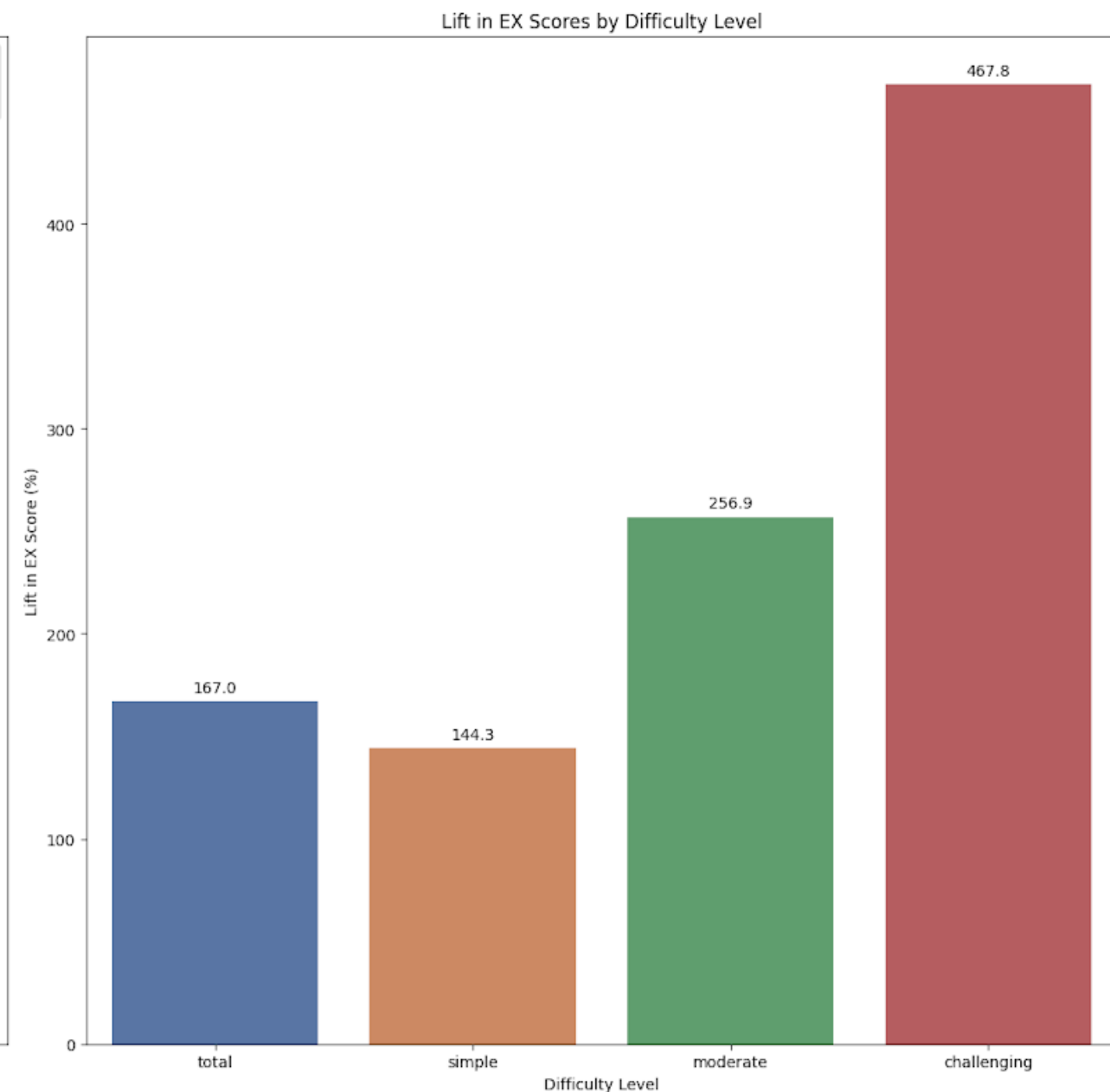
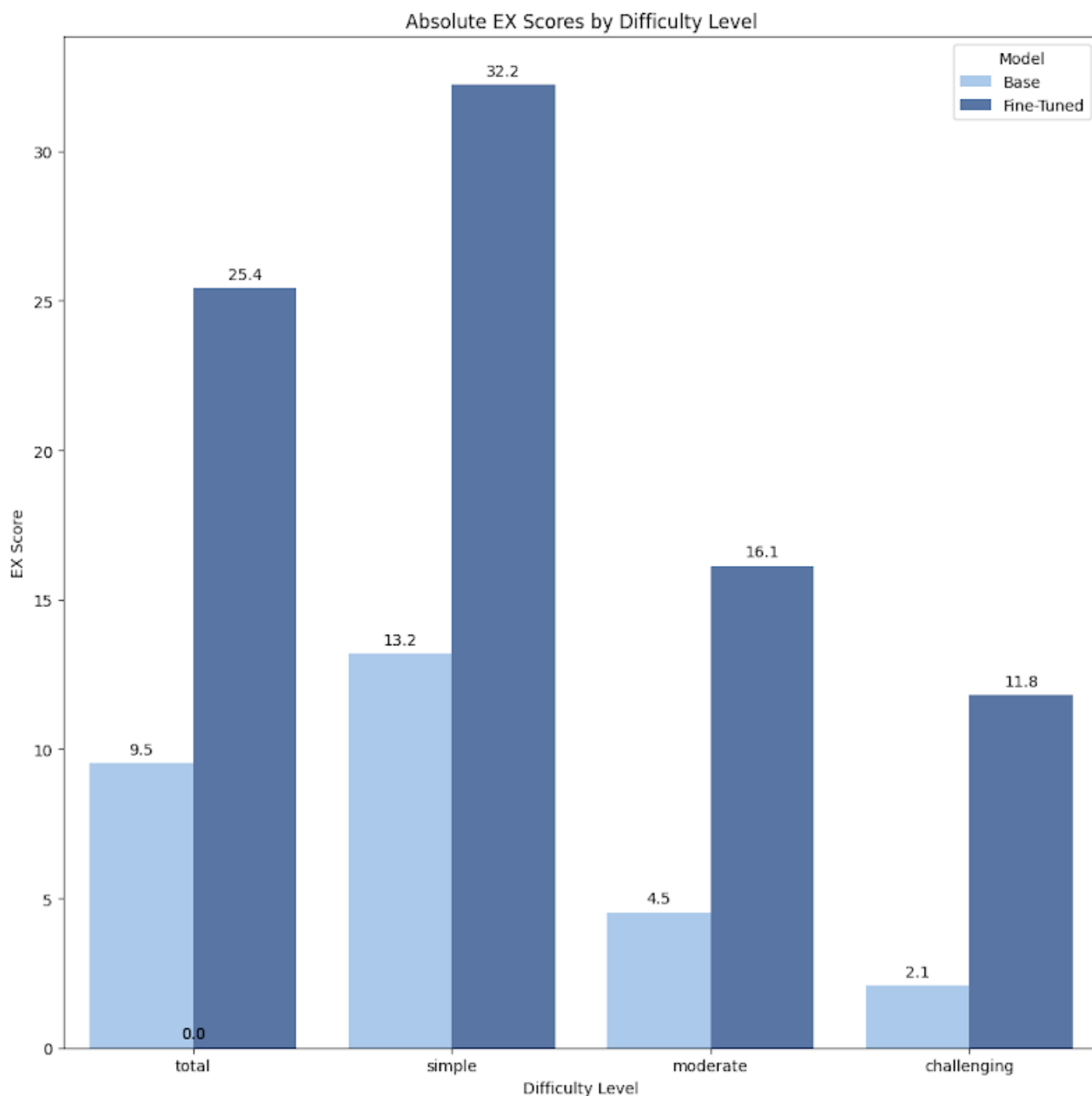
What is a good dataset?

Data is the most valuable asset in LLM development. While datasets can't be directly evaluated like models, high-quality datasets have the following characteristics:

- **Accuracy:** Samples should be factually correct, helpful to users, and well-written. Answers should also be relevant to their corresponding instructions.
- **Diversity:** You want to cover as many use cases as possible to ensure proper instruction-following and relevant answers. This requires a wide range of topics, contexts, lengths, writing styles, etc. sampled in a representative way.
- **Complexity:** Answers should be nontrivial and a/ representative of tasks you expect the model to handle or b/ include complex tasks involving multi-step reasoning, planning, etc.

Measuring accuracy can be easy in the case of mathematical problems using a Python interpreter, or near-impossible with open-ended, subjective questions. On the other hand, clustering datasets by topic is a good way of measuring diversity. Finally, complexity can be assessed using other LLMs acting like judges.

A Llama-3-8B model fine-tuned on a subset of the synthetic text-to-SQL data outperforms the base model by a wide margin on the BIRD-SQL benchmark, delivering an overall lift of 167% and a lift of more than 467% on challenging questions.



LoRAX (LoRA eXchange) is a framework that allows users to serve thousands of fine-tuned models on a single GPU, dramatically reducing the cost of serving without compromising on throughput or latency.



- 🖥️ **Dynamic Adapter Loading:** include any fine-tuned LoRA adapter from [HuggingFace](#), [Predibase](#), or [any filesystem](#) in your request, it will be loaded just-in-time without blocking concurrent requests. [Merge adapters](#) per request to instantly create powerful ensembles.
- 🧑‍🤝‍🧑 **Heterogeneous Continuous Batching:** packs requests for different adapters together into the same batch, keeping latency and throughput nearly constant with the number of concurrent adapters.
- 🍰 **Adapter Exchange Scheduling:** asynchronously prefetches and offloads adapters between GPU and CPU memory, schedules request batching to optimize the aggregate throughput of the system.
- 🧑 **Optimized Inference:** high throughput and low latency optimizations including tensor parallelism, pre-compiled CUDA kernels ([flash-attention](#), [paged attention](#), [SGMV](#)), quantization, token streaming.
- 🚢 **Ready for Production** prebuilt Docker images, Helm charts for Kubernetes, Prometheus metrics, and distributed tracing with Open Telemetry. OpenAI compatible API supporting multi-turn chat conversations. Private adapters through per-request tenant isolation. [Structured Output](#) (JSON mode).
- 🍷 **Free for Commercial Use:** Apache 2.0 License. Enough said



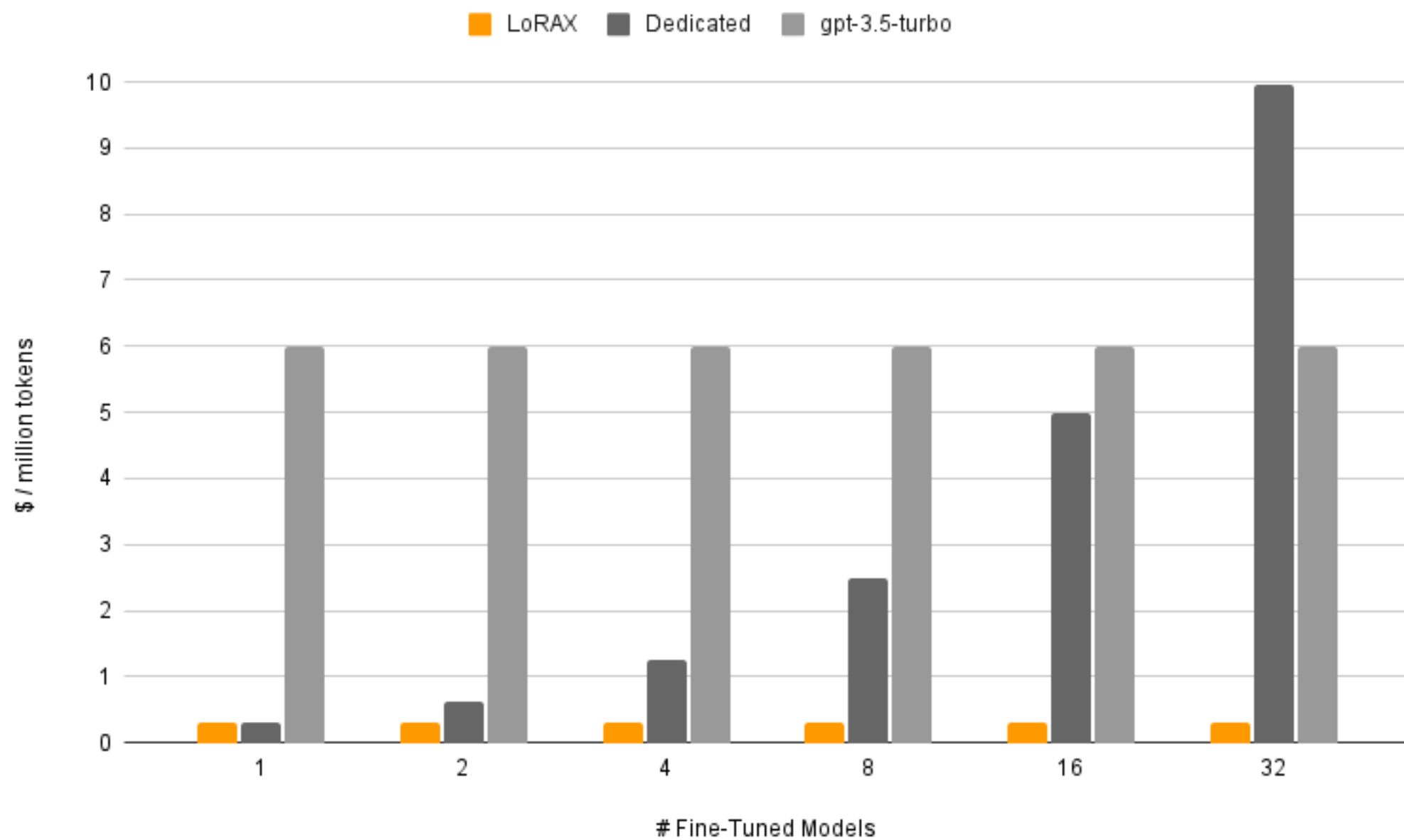
Serving a fine-tuned model with LoRAX consists of two components:

- [Base Model](#): pretrained large model shared across all adapters.
- [Adapter](#): task-specific adapter weights dynamically loaded per request.

LoRAX supports a number of Large Language Models as the base model including [Llama](#) (including [CodeLlama](#)), [Mistral](#) (including [Zephyr](#)), and [Qwen](#). See [Supported Architectures](#) for a complete list of supported base models.

Base models can be loaded in fp16 or quantized with bitsandbytes, [GPT-Q](#), or [AWQ](#).

Supported adapters include LoRA adapters trained using the [PEFT](#) and [Ludwig](#) libraries. Any of the linear layers in the model can be adapted via LoRA and loaded in LoRAX.



```
pip install lorax-client
```



```
from lorax import Client
```

```
client = Client("http://127.0.0.1:8080")
```

```
# Prompt the base LLM
```

```
prompt = "[INST] Natalia sold clips to 48 of her friends in April, and then she sold half as many clips  
in May. How many clips did Natalia sell altogether in April and May? [/INST]"
```

```
print(client.generate(prompt, max_new_tokens=64).generated_text)
```

```
# Prompt a LoRA adapter
```

```
adapter_id = "vineetsharma/qlora-adapter-Mistral-7B-Instruct-v0.1-gsm8k"
```

```
print(client.generate(prompt, max_new_tokens=64, adapter_id=adapter_id).generated_text)
```

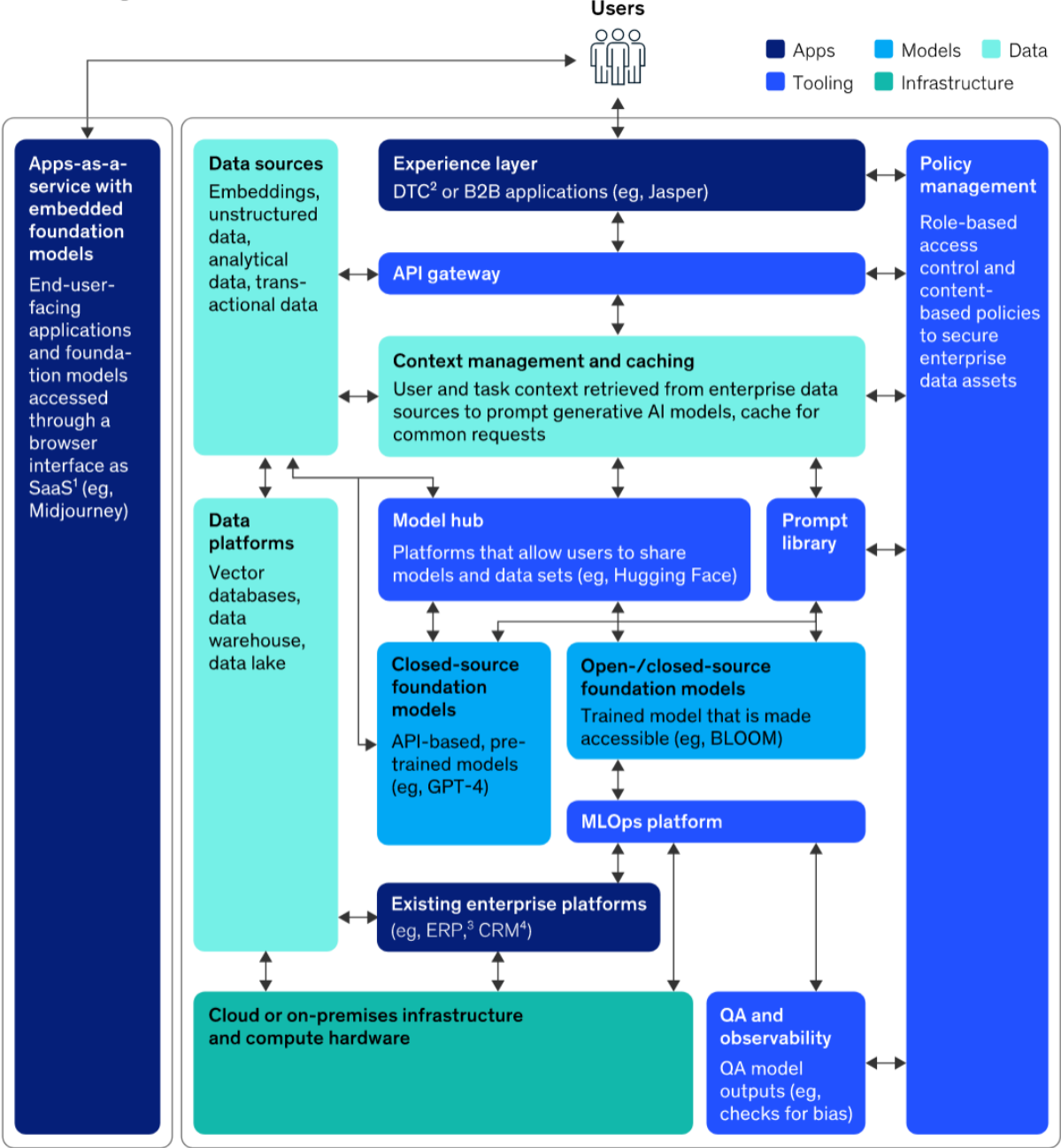
<https://loraexchange.ai/>

AI TechStack

Tech Stack

The tech stack for generative AI is emerging.

Illustrative generative AI tech stack



GEN-AI Service Development

