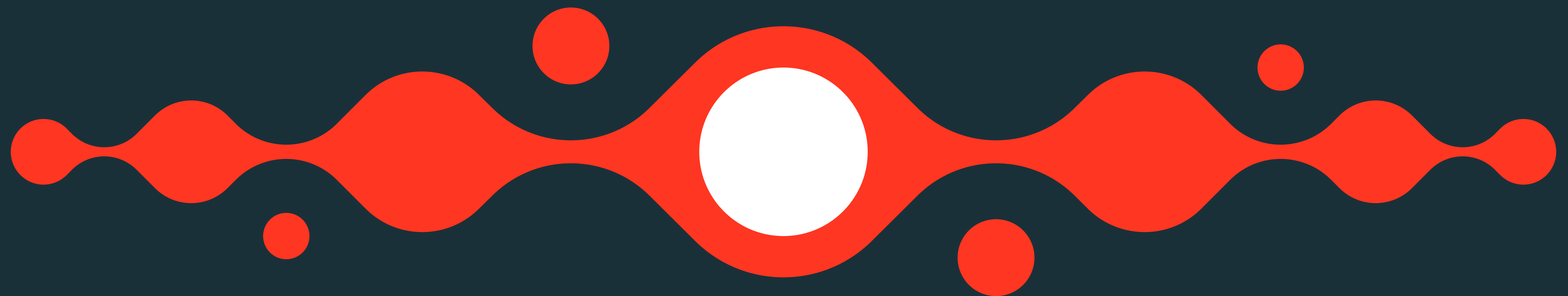
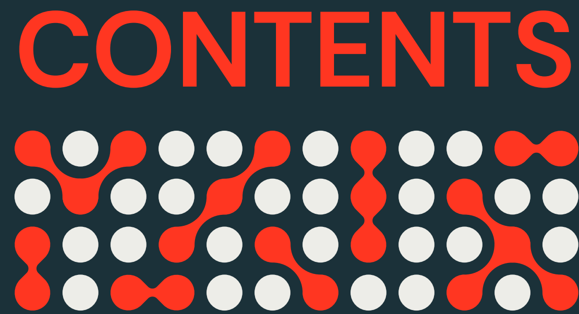
The Big Book of GenAI



5 ways to leverage your data to build production-quality GenAI applications

CONTENTS



Introduction 3

The Path to Deploying Production-Quality GenAI Applications 5

- **Stage 0: Foundation Models** 5
  - Use Case: Introducing DBRX: A New State-of-the-Art Open LLM 5
- **Stage 1: Prompt Engineering** 19
  - Use Case: Automated Analysis of Product Reviews Using Large Language Models 20
- **Stage 2: Retrieval Augmented Generation** 25
  - Use Case: Improve Your RAG Application Response Quality With Real-Time Structured Data 27
- **Stage 3: Fine-Tuning a Foundation Model** 33
  - Use Case: Creating a Bespoke LLM for AI-Generated Documentation 34
  - Use Case: Efficient Fine-Tuning With LoRA: A Guide to Optimal Parameter Selection for Large Language Models 43
- **Stage 4: Pretraining** 60
  - Use Case: Training Stable Diffusion From Scratch for <$50K With MosaicML 62
  - Use Case: Deep Dive: How We Trained Stable Diffusion for Less Than $50K 68
- **Stage 5: LLM Evaluation** 81
  - Use Case: Best Practices for LLM Evaluation of RAG Application 82
  - Use Case: Offline LLM Evaluation: Step-by-Step GenAI Application Assessment on Databricks 98

Summary 117

- **GenAI Training** 117
- **Additional Resources** 117

databricks

Introduction

Achieving Production-Quality GenAI Requires New Tools and Skills

Generative AI has opened new worlds of possibilities for businesses and is being emphatically embraced across organizations. According to a recent MIT Tech Review report, all 600 CIOs surveyed stated they are increasing their investment in AI, and 71% are planning to build their own custom large language models (LLMs) or other GenAI models. However, many organizations have found it challenging to deploy these applications at production quality. To meet the standard of quality required for customer-facing applications, AI output must be accurate, governed and safe.

Data Infrastructure Must Evolve to Support GenAI-Powered Applications

Making the leap to generative AI is not just about deploying a chatbot; it requires a reshaping of the foundational aspects of data management. Central to this transformation is the emergence of data lakehouses as the new "modern data stack." These advanced data architectures are essential to harnessing the full potential of GenAI, driving faster, more cost-effective and wider democratization of data and AI technologies. As businesses increasingly rely on GenAI-powered tools and applications for competitive advantage, the underlying data infrastructure must evolve to support these advanced technologies effectively and securely.

No Matter Where You Are on Your Path to Deploying GenAI Applications, the Quality of Your Data Matters

Businesses need to achieve production quality with their GenAI applications. Developers need rich tools for understanding the quality of their data and model outputs, along with an underlying platform that lets them combine and optimize all aspects of the GenAI process. GenAI has many components such as data preparation, retrieval models, language models (either SaaS or open source), ranking and post-processing pipelines, prompt engineering, and training models on custom enterprise data.

To help you overcome common enterprise challenges with building GenAI, we've compiled a collection of technical content and code samples. We'll start each section with a brief overview and then provide use cases and example code for reference.

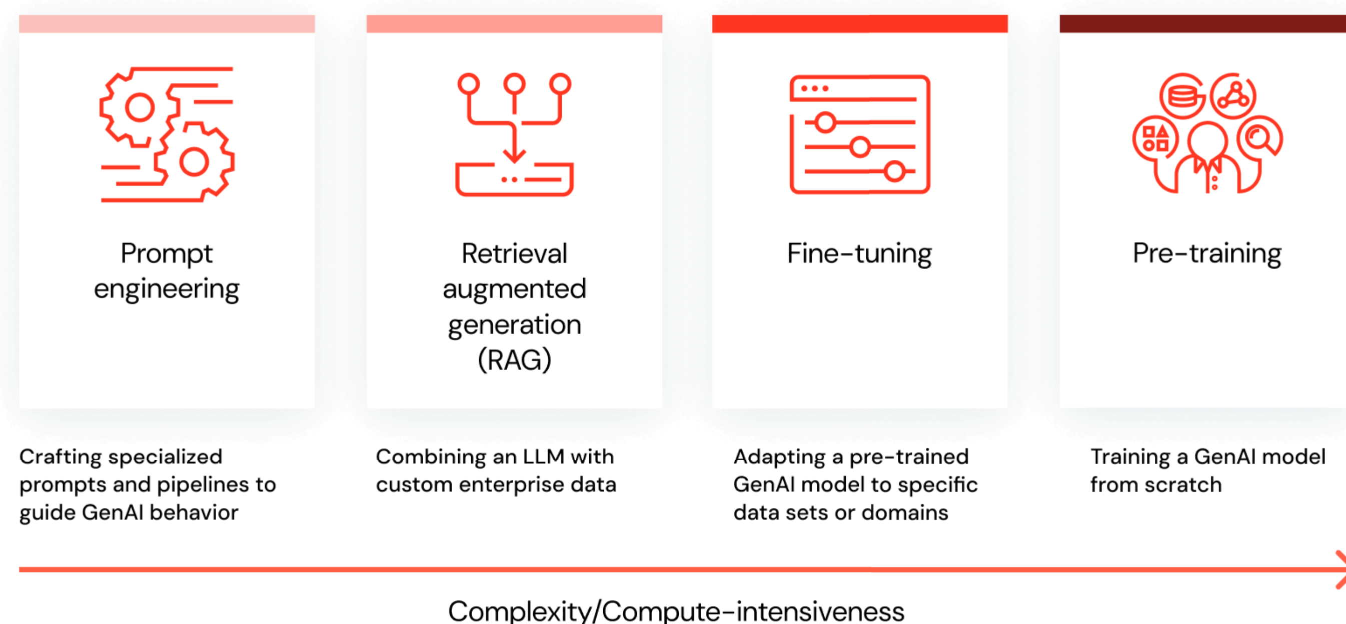
databricks

In this eBook, you'll learn:

- How to plan a path from basic to advanced GenAI applications, leveraging your organization's data
- How to use retrieval augmented generation (RAG) to make an off-the-shelf AI system smarter
- How to evaluate LLMs and where you want to invest in more powerful AI tools and systems that drive more significant operational gain
- How to build a custom LLM that may be better, faster and cheaper for your organization
- When it might be worth it to pretrain your own model — and more

Use cases for GenAI covered:

- How to use LLMs to gain actionable insights from product reviews
- How to use RAG for a chatbot to improve the quality of output
- How to train your own generative AI model in a cost-effective manner
- How to monitor and evaluate your deployed LLMs and GenAI applications



databricks

The Path to Deploying Production-Quality GenAI Applications

databricks

Stage 0: Foundation Models

Before setting off to create production-quality GenAI applications, we need to cover the base language models that serve as the foundation for layers of increasingly complex techniques. Foundation models commonly refer to large language models that have been trained over extensive datasets to be generally good at some task (chat, instruction following, code generation, etc.).

We won't cover many models, as it is a constantly shifting landscape, but it is important to note that while underlying architectures may differ drastically, foundation models generally fall under two categories: proprietary (such as GPT-3.5 and Gemini) and open source (such as Llama2-70B and DBRX). The main difference between the two is that while proprietary models historically have an edge on outright performance, users have to send their data out to a third party and don't have control over the underlying model as they're often being updated and changed.

Open source models, on the other hand, offer users full control over the model and the ability to run it on their own terms with their own governance and data privacy. Here's a current list of many open source GenAI models across different domains that are all free for commercial use. Databricks has also created their own state-of-the-art open source foundation model so users can build the highest-quality production GenAI applications.

Foundation Model Use Case

INTRODUCING DBRX: A NEW STATE-OF-THE-ART OPEN LLM

We are excited to introduce DBRX, an open, general-purpose LLM created by Databricks. Across a range of standard benchmarks, DBRX sets a new state-of-the-art for established open LLMs. Moreover, it provides the open community and enterprises building their own LLMs with capabilities that were previously limited to closed model APIs; according to our measurements, it surpasses GPT-3.5, and it is competitive with Gemini 1.0 Pro. It is an especially capable code model, surpassing specialized models like CodeLLaMA-70B on programming, in addition to its strength as a general-purpose LLM.

This state-of-the-art quality comes with marked improvements in training and inference performance. DBRX advances the state-of-the-art in efficiency among open models thanks to its fine-grained mixture-of-experts (MoE) architecture. Inference is up to 2x faster than LLaMA2-70B, and DBRX is about 40% of the size of Grok-1 in terms of both total and active parameter-counts. When hosted on Mosaic AI Model Serving, DBRX can generate text at up to 150 tok/s/user. Our customers will find that training MoEs is also about 2x more FLOP-efficient than training dense models for the same final model quality. End-to-end, our overall recipe for DBRX (including the pretraining data, model architecture, and optimization strategy) can match the quality of our previous-generation MPT models with nearly 4x less compute.

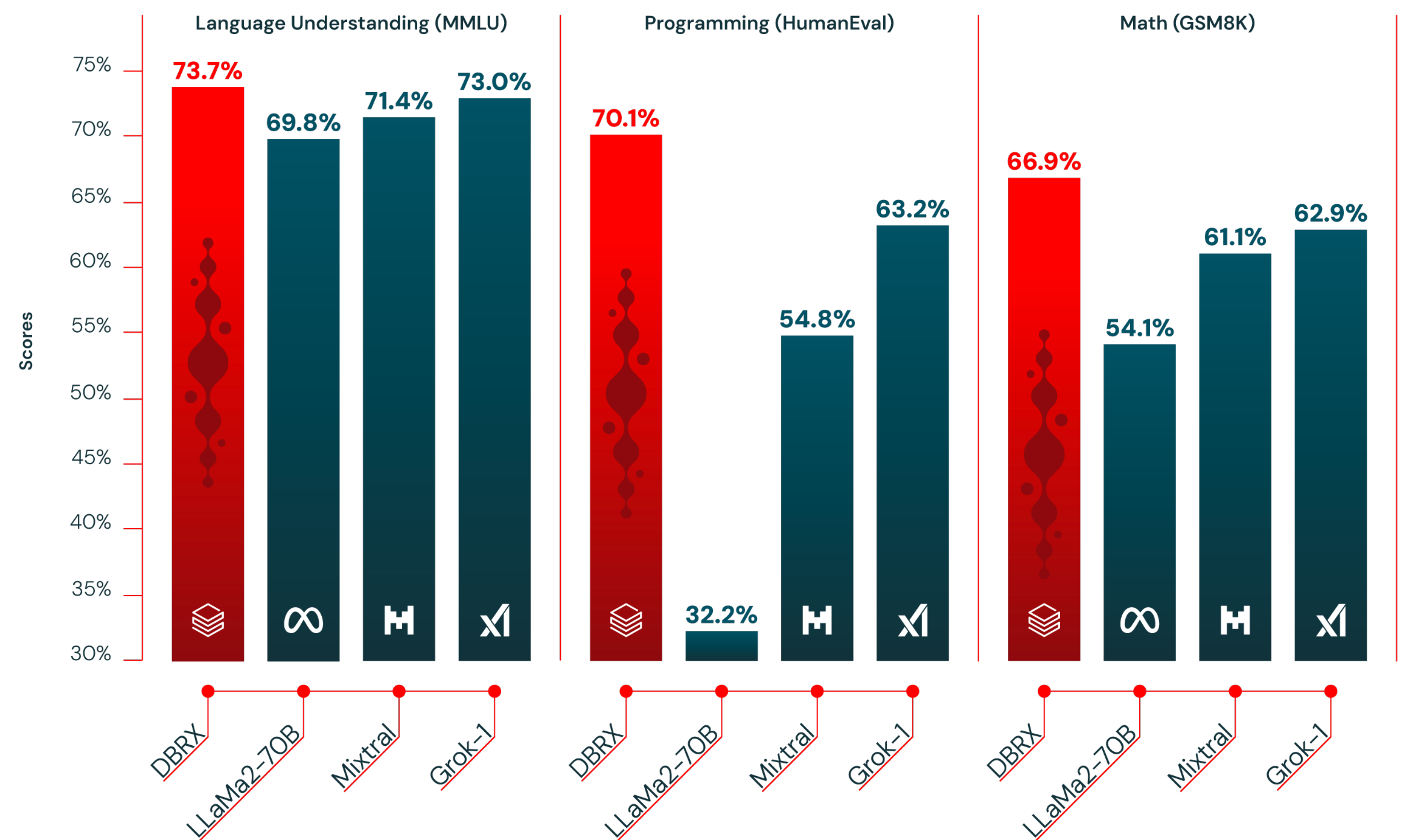


Figure 1: DBRX outperforms established open source models on language understanding (MMLU), Programming (HumanEval), and Math (GSM8K).

The weights of the base model (DBRX Base) and the fine-tuned model (DBRX Instruct) are available on Hugging Face under an open license. Starting today, DBRX is available for Databricks customers to use via APIs, and Databricks customers can pretrain their own DBRX-class models from scratch or continue training on top of one of our checkpoints using the same tools and science we used to build it. DBRX is already being integrated into our GenAI-powered products, where — in applications like SQL — early rollouts have surpassed GPT-3.5 Turbo and are challenging GPT-4 Turbo. It is also a leading model among open models and GPT-3.5 Turbo on RAG tasks.

Training mixture-of-experts models is hard. We had to overcome a variety of scientific and performance challenges to build a pipeline robust enough to repeatedly train DBRX-class models in an efficient manner. Now that we have done so, we have a one-of-a-kind training stack that allows any enterprise to train world-class MoE foundation models from scratch. We look forward to sharing that capability with our customers and sharing our lessons learned with the community.

Download DBRX today from Hugging Face (DBRX Base, DBRX Instruct), or try out DBRX Instruct in our HF Space, or see our model repository on github: databricks/dbrx.

What Is DBRX?

DBRX is a transformer-based decoder-only large language model (LLM) that was trained using next-token prediction. It uses a fine-grained mixture-of-experts (MoE) architecture with 132B total parameters of which 36B parameters are active on any input. It was pre-trained on 12T tokens of text and code data. Compared to other open MoE models like Mixtral and Grok-1, DBRX is fine-grained, meaning it uses a larger number of smaller experts. DBRX has 16 experts and chooses 4, while Mixtral and Grok-1 have 8 experts and choose 2. This provides 65x more possible combinations of experts and we found that this improves model quality. DBRX uses rotary position encodings (RoPE), gated linear units (GLU), and grouped query attention (GQA). It uses the GPT-4 tokenizer as provided in the tiktoken repository. We made these choices based on exhaustive evaluation and scaling experiments.

DBRX was pretrained on 12T tokens of carefully curated data and a maximum context length of 32k tokens. We estimate that this data is at least 2x better token-for-token than the data we used to pretrain the MPT family of models. This new dataset was developed using the full suite of Databricks tools, including Apache Spark™ and Databricks notebooks for data processing, Unity Catalog for data management and governance, and MLflow for experiment tracking. We used curriculum learning for pretraining, changing the data mix during training in ways we found to substantially improve model quality.

Quality on Benchmarks vs. Leading Open Models

Table 1 shows the quality of DBRX Instruct and leading established, open models. DBRX Instruct is the leading model on composite benchmarks, programming and mathematics benchmarks, and MMLU. It surpasses all chat or instruction fine-tuned models on standard benchmarks.

Composite benchmarks. We evaluated DBRX Instruct and peers on two composite benchmarks: the Hugging Face Open LLM Leaderboard (the average of ARC-Challenge, HellaSwag, MMLU, TruthfulQA, WinoGrande, and GSM8k) and the Databricks Model Gauntlet (a suite of over 30 tasks spanning six categories: world knowledge, commonsense reasoning, language understanding, reading comprehension, symbolic problem solving, and programming).

Among the models we evaluated, DBRX Instruct scores the highest on two composite benchmarks: the Hugging Face Open LLM Leaderboard (74.5% vs. 72.7% for the next highest model, Mixtral Instruct) and the Databricks Gauntlet (66.8% vs. 60.7% for the next highest model, Mixtral Instruct).

Programming and mathematics. DBRX Instruct is especially strong at programming and mathematics. It scores higher than the other open models we evaluated on HumanEval (70.1% vs. 63.2% for Grok-1, 54.8% for Mixtral Instruct, and 32.2% for the best-performing LLaMA2-70B variant) and GSM8k (66.9% vs. 62.9% for Grok-1, 61.1% for Mixtral Instruct, and 54.1% for the best-performing LLaMA2-70B variant). DBRX outperforms Grok-1, the next best model on these benchmarks, despite the fact that Grok-1 has 2.4x as many parameters. On HumanEval, DBRX Instruct even surpasses CodeLLaMA-70B Instruct, a model built explicitly for programming, despite the fact that DBRX Instruct is designed for general-purpose use (70.1% vs. 67.8% on HumanEval as reported by Meta in the CodeLLaMA blog).

MMLU. DBRX Instruct scores higher than all other models we consider on MMLU, reaching 73.7%.

MODEL	DBRX INSTRUCT	MIXTRAL INSTRUCT	MIXTRAL BASE	LLAMA2-70 B CHAT	LLAMA2-70 B BASE	GROK-11
Open LLM Leaderboard2 (Avg of next 6 rows)	**74.5%**	72.7%	68.4%	62.4%	67.9%	—
ARC-challenge 25-shot	68.9%	**70.1%**	66.4%	64.6%	67.3%	—
HellaSwag 10-shot	**89.0%**	87.6%	86.5%	85.9%	87.3%	—
MMLU 5-shot	**73.7%**	71.4%	71.9%	63.9%	69.8%	73.0%
Truthful QA 0-shot	**66.9%**	65.0%	46.8%	52.8%	44.9%	—
WinoGrande 5-shot	81.8%	81.1%	81.7%	80.5%	**83.7%**	—
GSM8k CoT 5-shot maj@13	**66.9%**	61.1%	57.6%	26.7%	54.1%	62.9% (8-shot)
Gauntlet v0.34 (Avg of 30+ diverse tasks)	**66.8%**	60.7%	56.8%	52.8%	56.4%	—
HumanEval5 0-Shot, pass@1 (Programming)	**70.1%**	54.8%	40.2%	32.2%	31.0%	63.2%

Table 1: Quality of DBRX Instruct and leading open models. See footnotes for details on how numbers were collected. Bolded and underlined is the highest score.

databricks

Quality on Benchmarks vs. Leading Closed Models

Table 2 shows the quality of DBRX Instruct and leading closed models. According to the scores reported by each model creator, DBRX Instruct surpasses GPT-3.5 (as described in the GPT-4 paper), and it is competitive with Gemini 1.0 Pro and Mistral Medium.

Across nearly all benchmarks we considered, DBRX Instruct surpasses or – at worst – matches GPT-3.5. DBRX Instruct outperforms GPT-3.5 on general knowledge as measured by MMLU (73.7% vs. 70.0%) and commonsense reasoning as measured by HellaSwag (89.0% vs. 85.5%) and WinoGrande (81.8% vs. 81.6%). DBRX Instruct especially shines on programming and mathematical reasoning as measured by HumanEval (70.1% vs. 48.1%) and GSM8k (72.8% vs. 57.1%).

DBRX Instruct is competitive with Gemini 1.0 Pro and Mistral Medium. Scores for DBRX Instruct are higher than Gemini 1.0 Pro on Inflection Corrected MTBench, MMLU, HellaSwag, and HumanEval, while Gemini 1.0 Pro is stronger on GSM8k. Scores for DBRX Instruct and Mistral Medium are similar for HellaSwag, while Mistral Medium is stronger on Winogrande and MMLU and DBRX Instruct is stronger on HumanEval, GSM8k, and Inflection Corrected MTBench.

MODEL	DBRX INSTRUCT	GPT-3.5[7]	GPT-4[8]	CLAUDE 3 HAIKU	CLAUDE 3 SONNET	CLAUDE 3 OPUS	GEMINI 1.0 PRO	GEMINI 1.5 PRO	MISTRAL MEDIUM	MISTRAL LARGE
MT Bench (Inflection corrected, n=5)	8.39 ± 0.08	—	—	8.41 ± 0.04	8.54 ± 0.09	9.03 ± 0.06	8.23 ± 0.08	—	8.05 ± 0.12	8.90 ± 0.06
MMLU 5-shot	73.7%	70.0%	86.4%	75.2%	79.0%	86.8%	71.8%	81.9%	75.3%	81.2%
HellaSwag 10-shot	89.0%	85.5%	95.3%	85.9%	89.0%	95.4%	84.7%	92.5%	88.0%	89.2%
HumanEval 0-Shot pass@1 (Programming)	70.1% temp=0, N=1	48.1%	67.0%	75.9%	73.0%	84.9%	67.7%	71.9%	38.4%	45.1%
GSM8k CoT maj@1	72.8% (5-shot)	57.1% (5-shot)	92.0% (5-shot)	88.9%	92.3%	95.0%	86.5% (maj1@32)	91.7% (11-shot)	66.7% (5-shot)	81.0% (5-shot)
WinoGrande 5-shot	81.8%	81.6%	87.5%	—	—	—	—	—	88.0%	86.7%

Table 2: Quality of DBRX Instruct and leading closed models. Other than Inflection Corrected MTBench (which we measured ourselves on model endpoints), numbers were as reported by the creators of these models in their respective whitepapers. See footnotes for additional details.

Quality on Long-Context Tasks and RAG

DBRX Instruct was trained with up to a 32K token context window. Table 3 compares its performance to that of Mixtral Instruct and the latest versions of the GPT-3.5 Turbo and GPT-4 Turbo APIs on a suite of long-context benchmarks (KV-Pairs from the Lost in the Middle paper and HotpotQAXL, a modified version of HotPotQA that extends the task to longer sequence lengths). GPT-4 Turbo is generally the best model at these tasks. However, with one exception, DBRX Instruct performs better than GPT-3.5 Turbo at all context lengths and all parts of the sequence. Overall performance for DBRX Instruct and Mixtral Instruct are similar.

MODEL	DBRX INSTRUCT	MIXTRAL INSTRUCT	GPT-3.5 TURBO (API)	GPT-4 TURBO (API)
Answer in Beginning Third of Context	45.1%	41.3%	37.3%*	**49.3%**
Answer in Middle Third of Context	45.3%	42.7%	37.3%*	**49.0%**
Answer in Last Third of Context	48.0%	44.4%	37.0%*	**50.9%**
2K Context	59.1%	64.6%	36.3%	**69.3%**
4K Context	**65.1%**	59.9%	35.9%	63.5%
8K Context	59.5%	55.3%	45.0%	**61.5%**
16K Context	27.0%	20.1%	**31.7%**	26.0%
32K Context	19.9%	14.0%	—	**28.5%**

Table 3: The average performance of models on the KV-Pairs and HotpotQAXL benchmarks. Bold is the highest score. Underlined is the highest score other than GPT-4 Turbo. GPT-3.5 Turbo supports a maximum context length of 16K, so we could not evaluate it at 32K. *Averages for the beginning, middle, and end of the sequence for GPT-3.5 Turbo include only contexts up to 16K.

One of the most popular ways to leverage a model's context is retrieval augmented generation (RAG). In RAG, content relevant to a prompt is retrieved from a database and presented alongside the prompt to give the model more information than it would otherwise have. Table 4 shows the quality of DBRX on two RAG benchmarks — Natural Questions and HotPotQA — when the model is also provided with the top 10 passages retrieved from a corpus of Wikipedia articles using the embedding model bge-large-en-v1.5. DBRX Instruct is competitive with open models like Mixtral Instruct and LLaMA2-70B Chat and the current version of GPT-3.5 Turbo.

MODEL	DBRX INSTRUCT	MIXTRAL INSTRUCT	LLAMA2-70B CHAT	GPT 3.5 TURBO (API)	GPT 4 TURBO (API)
Natural Questions	60.0%	59.1%	56.5%	57.7%	**63.9%**
HotPotQA	55.0%	54.2%	54.7%	53.0%	**62.9%**

Table 4: The performance of the models measured when each model is given the top 10 passages retrieved from a Wikipedia corpus using bge-large-en-v1.5. Accuracy is measured by matching within the model's answer. Bold is the highest score. Underlined is the highest score other than GPT-4 Turbo.

Training Efficiency

Model quality must be placed in the context of how efficient the model is to train and use. This is especially so at Databricks, where we build models like DBRX to establish a process for our customers to train their own foundation models.

We found training mixture-of-experts models to provide substantial improvements in compute-efficiency for training (Table 5). For example, training a smaller member of the DBRX family called DBRX MoE-B (23.5B total parameters, 6.6B active parameters) required 1.7x fewer FLOPs to reach a score of 45.5% on the Databricks LLM Gauntlet than LLaMA2-13B required to reach 43.8%. DBRX MoE-B also contains half as many active parameters as LLaMA2-13B.

Looking holistically, our end-to-end LLM pretraining pipeline has become nearly 4x more compute-efficient in the past ten months. On May 5, 2023, we released MPT-7B, a 7B parameter model trained on 1T tokens that reached a Databricks LLM Gauntlet score of 30.9%. A member of the DBRX family called DBRX MoE-A (7.7B total parameters, 2.2B active parameters) reached a Databricks Gauntlet score of 30.5% with 3.7x fewer FLOPs. This efficiency is the result of a number of improvements, including using an MoE architecture, other architecture changes to the network, better optimization strategies, better tokenization, and – very importantly – better pretraining data.

In isolation, better pretraining data made a substantial impact on model quality. We trained a 7B model on 1T tokens (called DBRX Dense-A) using the DBRX pretraining data. It reached 39.0% on the Databricks Gauntlet compared to 30.9% for MPT-7B. We estimate that our new pretraining data is at least 2x better token-for-token than the data used to train MPT-7B. In other words, we estimate that half as many tokens are necessary to reach the same model quality. We determined this by training DBRX Dense-A on 500B tokens; it outperformed MPT-7B on the Databricks Gauntlet, reaching 32.1%. In addition to better data quality, another important contributor to this token-efficiency may be the GPT-4 tokenizer, which has a large vocabulary and is believed to be especially token-efficient. These lessons about improving data quality translate directly into practices and tools that our customers use to train foundation models on their own data.

MODEL	TOTAL PARAMS	ACTIVE PARAMS	GAUNTLET SCORE	RELATIVE FLOPS
DBRX MoE-A	7.7B	2.2B	30.5%	1x
MPT-7B (1T tokens)	—	6.7B	30.9%	3.7x
DBRX Dense-A (1T tokens)	—	6.7B	39.0%	3.7x
DBRX Dense-A (500B tokens)	—	6.7B	32.1%	1.85x
DBRX MoE-B	23.5B	6.6B	45.5%	1x
LLaMA2-13B	—	13.0B	43.8%	1.7x

Table 5: Details of several test articles that we used to validate the training efficiency of the DBRX MoE architecture and end-to-end training pipeline.

Inference Efficiency

Figure 2 shows the end-to-end inference efficiency of serving DBRX and similar models using NVIDIA TensorRT-LLM with our optimized serving infrastructure and 16-bit precision. We aim for this benchmark to reflect real-world usage as closely as possible, including multiple users simultaneously hitting the same inference server. We spawn one new user per second, each user request contains an approximately 2000 token prompt, and each response comprises 256 tokens.

In general, MoE models are faster at inference than their total parameter-counts would suggest. This is due to the fact that they use relatively few parameters for each input. We find that DBRX is no exception in this respect. DBRX inference throughput is 2-3x higher than a 132B non-MoE model.

Inference efficiency and model quality are typically in tension: bigger models typically reach higher quality, but smaller models are more efficient for inference. Using an MoE architecture makes it possible to attain better tradeoffs between model quality and inference efficiency than dense models typically achieve. For example, DBRX is both higher quality than LLaMA2-70B and - thanks to having about half as many active parameters - DBRX inference throughput is up to 2x faster (Figure 2). Mixtral is another point on the improved pareto frontier attained by MoE models: it is smaller than DBRX, and it is correspondingly lower in terms of quality but reaches higher inference throughput. Users of the Databricks Foundation Model APIs can expect to see up to 150 tokens per second for DBRX on our optimized model serving platform with 8-bit quantization.

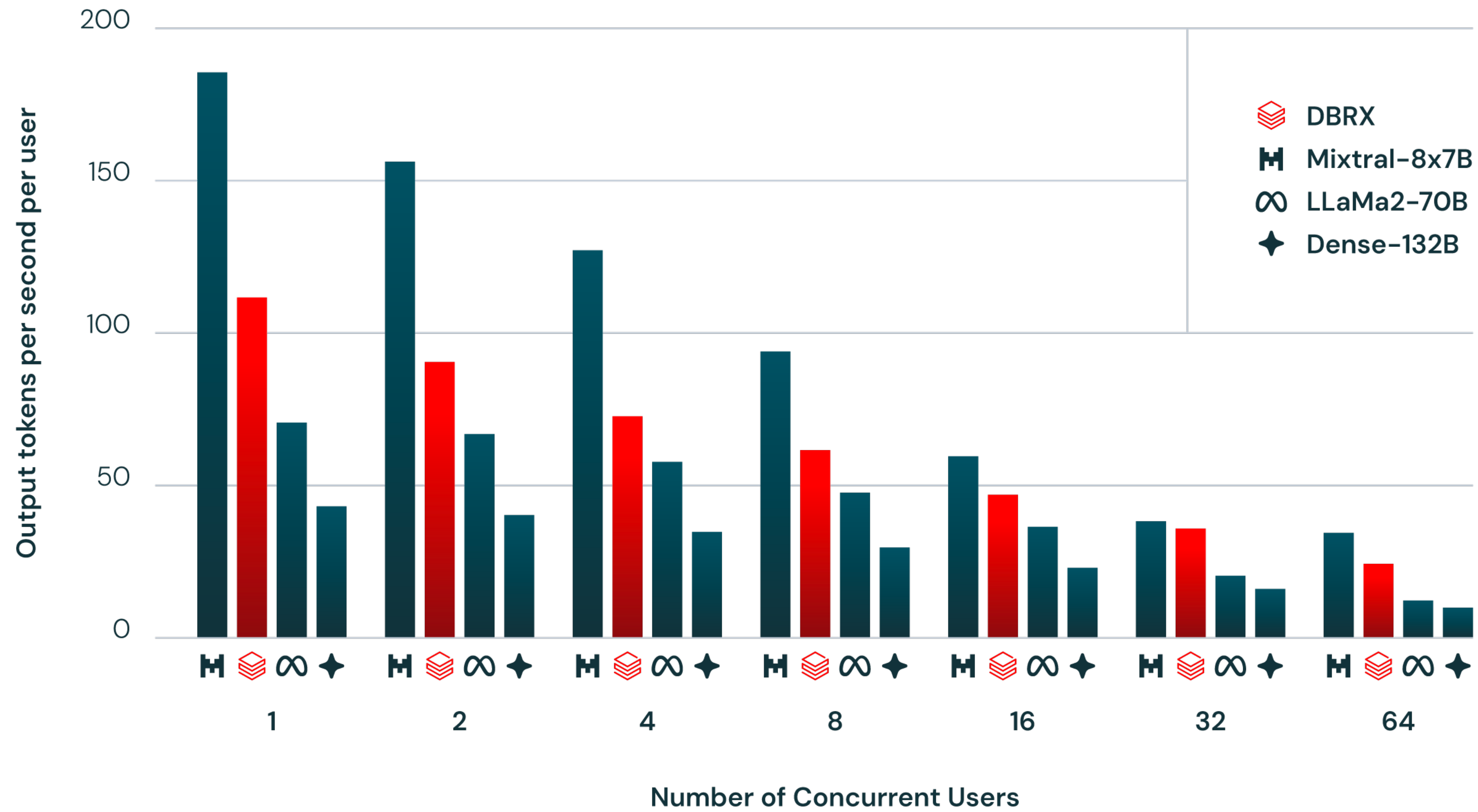


Figure 2: Inference throughput for various model configurations on our optimized serving infrastructure using NVIDIA TensorRT-LLM at 16-bit precision with the best optimization flags we could find. Models are run in tensor-parallel across the entire node. The input prompt contains approximately 2000 prompt tokens and we generate 256 output tokens. One new user spawns every second.

How We Built DBRX

DBRX was trained on 3072 NVIDIA H100s connected by 3.2Tbps Infiniband. The main process of building DBRX – including pretraining, post-training, evaluation, red-teaming, and refining – took place over the course of three months. It was the continuation of months of science, dataset research, and scaling experiments, not to mention years of LLM development at Databricks that includes the MPT and Dolly projects and the thousands of models we have built and brought to production with our customers.

To build DBRX, we leveraged the same suite of Databricks tools that are available to our customers. We managed and governed our training data using Unity Catalog. We explored this data using newly acquired Lilac AI. We processed and cleaned this data using Apache Spark™ and Databricks notebooks. We trained DBRX using optimized versions of our open-source training libraries: MegaBlocks, LLM Foundry, Composer, and Streaming. We managed large scale model training and finetuning across thousands of GPUs using our Mosaic AI Training service. We logged our results using MLflow. We collected human feedback for quality and safety improvements through Mosaic AI Model Serving and Inference Tables. We manually experimented with the model using the Databricks Playground. We found the Databricks tools to be best-in-class for each of their purposes, and we benefited from the fact that they were all part of a unified product experience.

Get Started With DBRX on Databricks

If you're looking to start working with DBRX right away, it's easy to do so with the Databricks Mosaic AI Foundation Model APIs. You can quickly get started with our pay-as-you-go pricing and query the model from our AI Playground chat interface. For production applications, we offer a provisioned throughput option to provide performance guarantees, support for finetuned models, and additional security and compliance. To privately host DBRX, you can download the model from the Databricks Marketplace and deploy the model on Model Serving.

Conclusions

At Databricks, we believe that every enterprise should have the ability to control its data and its destiny in the emerging world of GenAI. DBRX is a central pillar of our next generation of GenAI products, and we look forward to the exciting journey that awaits our customers as they leverage the capabilities of DBRX and the tools we used to build it. In the past year, we have trained thousands of LLMs with our customers. DBRX is only one example of the powerful and efficient models being built at Databricks for a wide range of applications, from internal features to ambitious use-cases for our customers.

As with any new model, the journey with DBRX is just the beginning, and the best work will be done by those who build on it: enterprises and the open community. This is also just the beginning of our work on DBRX, and you should expect much more to come.

Contributions

The development of DBRX was led by the Mosaic team that previously built the MPT model family, in collaboration with dozens of engineers, lawyers, procurement and finance specialists, program managers, marketers, designers, and other contributors from across Databricks. We are grateful to our colleagues, friends, family, and the community for their patience and support over the past months.

In creating DBRX, we stand on the shoulders of giants in the open and academic community. By making DBRX available openly, we intend to invest back in the community in hopes that we will build even greater technology together in the future. With that in mind, we gratefully acknowledge the work and collaboration of Trevor Gale and his MegaBlocks project (Trevor's PhD adviser is Databricks CTO Matei Zaharia), the PyTorch team and the FSDP project, NVIDIA and the TensorRT-LLM project, the vLLM team and project, EleutherAI and their LLM evaluation project, Daniel Smilkov and Nikhil Thorat at Lilac AI, and our friends at the Allen Institute for Artificial Intelligence (AI2).

databricks

Stage 1: Prompt Engineering

Many companies still remain in the foundational stages of adopting generative AI technology. They have no overarching AI strategy in place, no clear use cases to pursue and no access to a team of data scientists and other professionals who can help guide the company's AI adoption journey.

If this is like your business, a good starting point is an off-the-shelf LLM. While these LLMs lack the domain-specific expertise of custom AI models, experimentation can help you plot your next steps. Your employees can craft specialized prompts and workflows to guide their usage. Your leaders can get a better understanding of the strengths and weaknesses of these tools as well as a clearer vision of what early success in AI might look like. Your organization can use things like the Databricks AI Playground to figure out where to invest in more powerful AI tools and systems that drive more significant operational gain and even use LLMs as a judge to help evaluate responses.

PRACTICAL APPLICATIONS OF GENAI TECHNOLOGY

Let's delve into a compelling use case that illustrates the power of prompt engineering with off-the-shelf LLMs. Consider the challenge many businesses face: sifting through vast amounts of product reviews to glean actionable insights. Without a dedicated team of data scientists or a clear AI strategy, this task might seem daunting. However, leveraging the flexibility of LLMs through prompt engineering offers a straightforward solution.

databricks

Prompt Engineering Use Case

Automated Analysis of Product Reviews Using Large Language Models

Keep track of customer feedback at scale

Check out our LLM Solution Accelerators for Retail for more details and to download the notebooks.

While conversational AI has garnered a lot of media attention in recent months, the capabilities of large language models (LLMs) extend well beyond conversational interactions. It's in these less prominent capabilities such as query response, summarization, classification and search that many organizations are finding immediate opportunities to supercharge their workforce and up-level customer experiences.

The potential of these applications is staggering. By one estimate, LLMs (and other generative AI technologies) could, in the near future, address tasks that today occupy 60%–70% of employees' time. Through augmentation, numerous studies have shown that the time to complete various tasks performed by knowledge workers such as background research, data analysis and document writing can be cut in half. And still other studies have shown that the use of these technologies can dramatically reduce the time for new workers to achieve full productivity.

But before these benefits can be fully realized, organizations must first rethink the management of the unstructured information assets on which these models depend and find ways to mitigate the issues of bias and accuracy that affect their output. This is why so many organizations are currently focusing their efforts on focused, internal applications where a limited scope provides opportunities for better information access and human oversight can serve as a check to errant results. These applications, aligned with core capabilities already residing within the organization, have the potential to deliver real and immediate value, while LLMs and their supporting technologies continue to evolve and mature.

PRODUCT REVIEW SUMMARIZATION COULD USE A BOOST

To illustrate the potential of a more focused approach to LLM adoption, we consider a fairly simple and common task performed within many online retail organizations: product review summarization. Today, most organizations employ a modestly-sized team of workers to read and digest user feedback for insights that may help improve a product's performance or otherwise identify issues related to customer satisfaction.

The work is important but anything but sexy. A worker reads a review, takes notes, and moves on to the next. Individual reviews that require a response are flagged and a summary of the feedback from across multiple reviews are compiled for review by product or category managers.

This is a type of work that's ripe for automation. The volume of reviews that pour into a site mean the more detailed portions of this work are often performed on a limited subset of products across variable windows depending on a products importance. In more sophisticated organizations, rules detecting course or inappropriate language and models estimating user sentiment or otherwise classifying reviews for positive, negative or neutral experiences may be applied to help identify problematic content and draw a reviewer's attention to it. But either way, a lot is missed simply because we can't throw enough bodies at the problem to keep up and those bodies tend to become bored or fatigued with the monotony of the work.

LARGE LANGUAGE MODELS CAN AUTOMATE PRODUCT REVIEW ANALYSIS

By using an LLM, issues of scale and consistency can be easily addressed. All we need to do is bring the product reviews to the model and ask:

- What are the top three points of negative feedback found across these reviews?
- What features do our customers like best about this product?
- Do customers feel they are receiving sufficient value from the product relative to what they are being asked to pay?
- Are there any reviews that are especially negative or are using inappropriate language?

Within seconds we can have a tidy response, allowing our product managers to focus on responding to issues instead of simply detecting them.

But what about the problem of accuracy and bias? Standards for identifying inaccuracies and bias in LLM output are evolving as are techniques for better ensuring that outputs align with an organization's expectations, and the fine-tuning of models using approved content can go a long way to ensure models have a preference to generate content that's at least aligned with how an organization prefers to communicate.

This is a long-winded way of saying there is no ideal solution to the problem as of yet. But when compared to where we are with human-driven processes and more simplistic models or rules-based approaches, the results are expected to be better or at a minimum no worse than what we currently experience. And given that these review summaries are for internal consumption, the impact of an errant model can be easily managed.

YOU CAN BUILD A SOLUTION FOR THIS TODAY

To demonstrate exactly how this work could be performed, we have built a Solution Accelerator for summarizing product reviews. This is based heavily on a previously published blog from Sean Owen that addressed some of the core technical challenges of tuning an LLM on the Databricks platform. For the accelerator, we are using the Amazon Product Reviews Dataset, which contains 51 million user-generated reviews across 2 million distinct books as this provides access to a wide range of reviewer content and presents a scaling challenge many organizations will recognize.

We imagine a scenario in which a team of product managers receives customer feedback through online reviews. These reviews are important for identifying issues that may need to be addressed regarding a particular item and for steering future books to be offered by the site. Without the use of technology, this team struggles to read all the feedback and summarize into a workable set notes. As a result, they limit their attention to just the most critical items and are able to only process the feedback on a sporadic basis.

But using Databricks, they are able to set up a pipeline to collect feedback from a wider range of products and summarize these on a regular basis. Recognizing that positively rated products are likely to highlight the strengths of these books while lower rated products are likely to focus on their weaknesses, they separate these reviews based on user-provided ratings and task an LLM to extract different sets of information from each high-level category of reviews.

Summary metrics are provided to allow product managers an overview of the feedback received and are backed by more detailed summaries generated by the LLM (Figure 1).

Figure 1: Summary metrics and bullet-point details extracted from user reviews extracted using an LLM

DATABRICKS BRINGS TOGETHER ALL THE COMPONENTS OF A SOLUTION

The scenario demonstrated above depends on the use of an LLM. In months prior, the use of such an LLM required access to specialized computational infrastructures, but with advances in the open source community and investments in the Databricks platform, we are now able to run the LLM in our local Databricks environment.

In this particular scenario, the sensitivity of the data was not a motivating factor for this choice. Instead, we found that the volume of reviews to be processed tipped the cost scales toward the use of Databricks, allowing us to trim about one-third of the cost of implementing a similar solution using a third-party service.

In addition, we found that by implementing our own infrastructure, we were able to scale the environment up for faster processing, tackling as many as 760,000 reviews per hour in one test without having to be concerned with constraints imposed by an external service. While most organizations will not have the need to scale quite to that level, it's nice to know it is there should it be.

But this solution is more than just an LLM. To bring together the whole solution we needed to develop a data processing workflow to receive incoming reviews, prepare them for submission to the model and to capture model output for further analysis. As a unified data platform, Databricks provides us the means to address both data engineering and data science requirements without data replication. And when we are done processing the reviews, our analysts can use their tools of choice to query the output and make business decisions. Through Databricks, we have access to the full array of capabilities for us to build a solution aligned with our business' needs.

databricks

Stage 2: Retrieval Augmented Generation

Retrieval augmented generation (RAG) lets you bring in supplemental knowledge resources to make an off-the-shelf AI system smarter. RAG won't change the underlying behavior of the model, but it will improve the quality and accuracy of the responses.

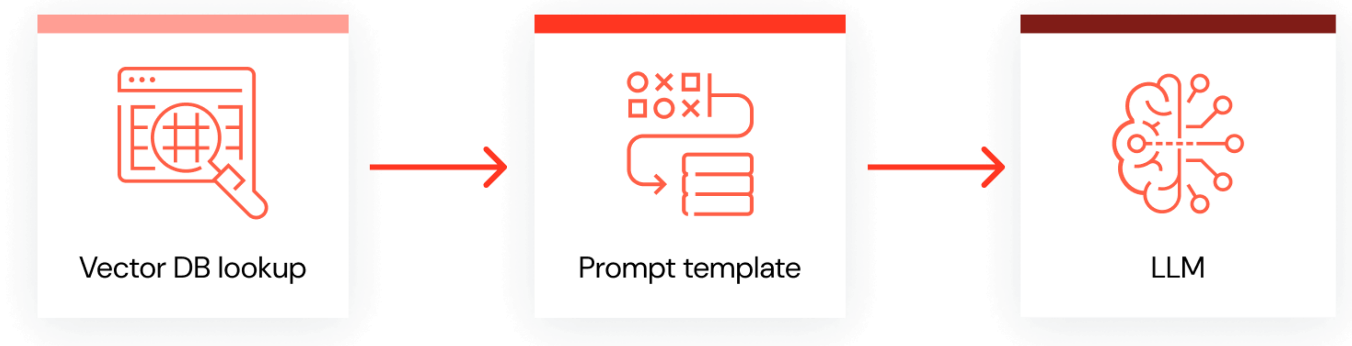
However, at this point, your business should not be uploading its "mission-critical" data. Instead, the RAG process typically involves smaller amounts of nonsensitive information.

For example, plugging in an employee handbook can allow your workers to start asking the underlying model questions about the organization's vacation policy. Uploading instruction manuals can help power a service chatbot. With the ability to query support tickets using AI, support agents can get answers quicker; however, inputting confidential financial data so employees can inquire about the company's performance is likely a step too far.

To get started, your team should first consolidate and cleanse the data you intend to use. With RAG, it's vital that your company stores the data in sizes that will be appropriate for the downstream models. Often, that requires users to splice it into smaller segments.

Then, you should seek out a tool like Databricks Vector Search, which enables users to quickly set up their own vector database. And because it's governed by Unity Catalog, granular controls can be put in place to ensure employees are only accessing the datasets for which they have credentials.

Finally, you can then plug that endpoint into a LLM. A tool like Databricks MLflow helps to centralize the management of those APIs.



Among the benefits of RAG are reduced hallucinations, more up-to-date and accurate responses, and better domain-specific intelligence. RAG-assisted models are also a more cost-effective approach for most organizations.

While RAG will help improve the results from commercial models, there are still many limitations to the use of RAG. If your business is unable to get the results it wants, it's time to move on to heavier-weight solutions, but moving beyond RAG-supported models often requires a much deeper commitment. The additional customization costs more and requires a lot more data.

That's why it's key that organizations first build a core understanding of how to use LLMs. By reaching the performance limitations of off-the-shelf models before moving on, you and your leadership can further hone in on where to allocate resources.

Enhance the Performance of Off-the-Shelf AI Models With RAG

Let's explore a practical use case that demonstrates how real-time structured data can significantly improve the response quality of your RAG applications. This example will showcase how integrating dynamic information can transform the effectiveness and applicability of AI in your business operations.

RAG Use Case

Improve Your RAG Application Response Quality With Real-Time Structured Data

by Mani Parkhe, Aakrati Talati, Sue Ann Hong, Craig Wiley, Chenen Liang and Mingyang Ge

Retrieval augmented generation (RAG) is an efficient mechanism to provide relevant data as context in GenAI applications. Most RAG applications typically use vector indexes to search for relevant context from unstructured data such as documentation, wikis, and support tickets. Yesterday, we announced Databricks Vector Search Public Preview that helps with exactly that. However, GenAI response quality can be enhanced by augmenting these text-based contexts with relevant and personalized structured data. Imagine a GenAI tool on a retail website where customers inquire, "Where's my recent order?" This AI must understand that the query is about a specific purchase, then gather up-to-date shipment information for line items, before using LLMs to generate a response. Developing these scalable applications demands substantial work, integrating technologies for handling both structured and unstructured data with GenAI capabilities.

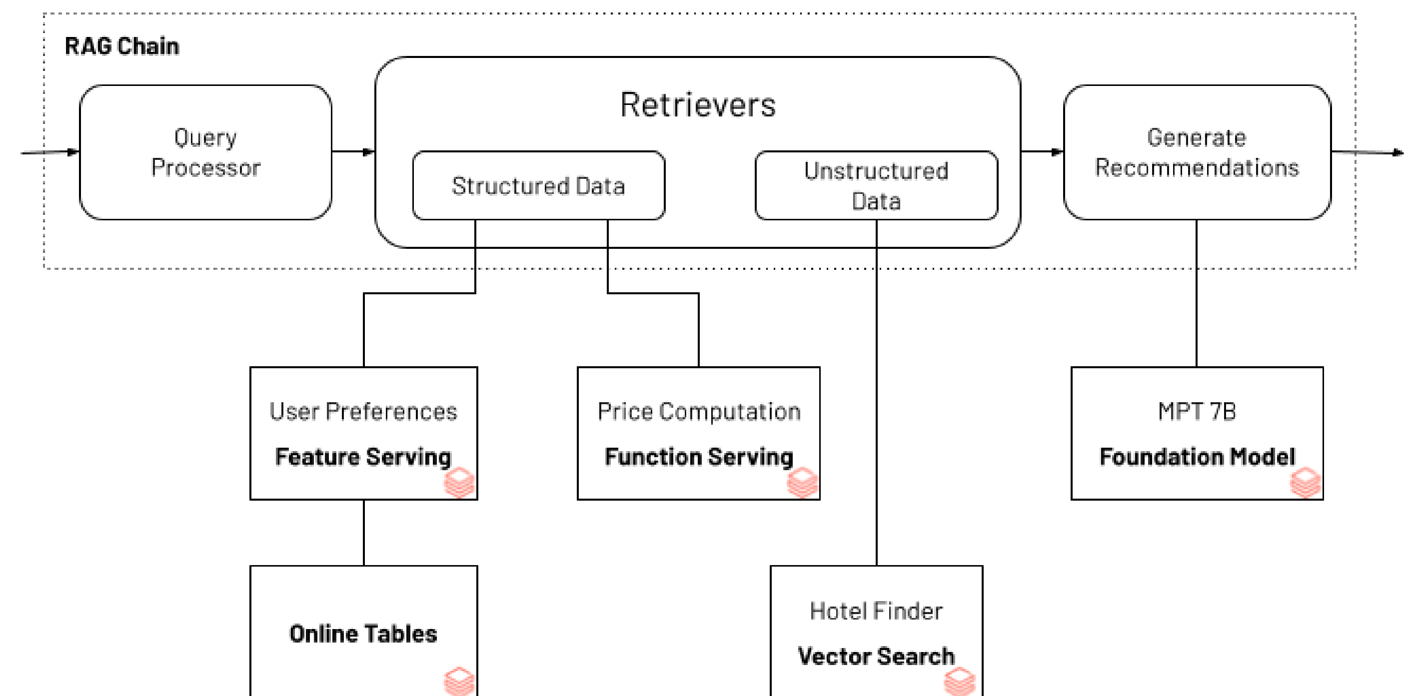
We are excited to announce the public preview of Databricks Feature & Function Serving, a low latency real-time service designed to serve structured data from the Databricks Data Intelligence Platform. You can instantly access pre-computed ML features as well as perform real-time data transformations by serving any Python function from Unity Catalog. The retrieved data can then be used in real-time rule engines, classical ML, and GenAI applications.

Using Feature and Function Serving (AWS)(Azure) for structured data in coordination with Databricks Vector Search (AWS)(Azure) for unstructured data significantly simplifies productionalization of GenAI applications. Users can build and deploy these applications directly in Databricks and rely on existing data pipelines, governance, and other enterprise features. Databricks customers across various industries are using these technologies along with open source frameworks to build powerful GenAI applications such as the ones described in the table below.

INDUSTRY	USE CASE
Retail	■ Product Recommendations / Search Ranking using user preferences, search history, location . . . etc. ■ Image and metadata based product search ■ Inventory management and forecasting using sales data, seasonal trends and market/competitive analysis
Education	■ Personalized learning plans based on past mistakes, historical trends and cohorts ■ Automated grading, feedback, follow-ups and progress reporting ■ Content filtering for issued devices
Financial Services	■ Natural language apps for analysts and investors to correlate earning calls and reports with market intelligence and historical trends ■ Fraud and risk analysis ■ Personalized wealth management, retirement planning, what-if analysis and next-best actions
Travel and Hospitality	■ Chatbots for personalized customer interactions and tailored travel recommendations ■ Dynamic route planning using weather, live traffic patterns, and historical data ■ Dynamic price optimization using competitive analysis and demand-based pricing
Healthcare and Life Sciences	■ Patient/member engagement and health summaries ■ Support apps for personalized care, clinical decisions and care coordination ■ R&D report summarization, clinical trial analysis, drug repurposing
Insurance	■ Risk assessment for mortgage underwriting using text and structured data about properties and neighborhoods ■ User chatbots for questions about policies, risk and what-if analysis ■ Claim processing automation
Technology and Manufacturing	■ Prescriptive maintenance and diagnostics for equipment using guided instruction ■ Anomaly detection on live data stream against historical statistics ■ Automated analysis for daily production / shift analysis and future planning
Media and Entertainment	■ In-app content discovery and recommendations, personalized email and digital marketing ■ Content localization ■ Personalized gaming experiences and game review

SERVING STRUCTURED DATA TO RAG APPLICATIONS

To demonstrate how structured data can help enhance the quality of a GenAI application, we use the following example for a travel planning chatbot. The example shows how user preferences (example: “ocean view” or “family friendly”) can be paired with unstructured information sourced about hotels to search for hotel matches. Typically hotel prices dynamically change based on demand and seasonality. A price calculator built into the GenAI application ensures that the recommendations are within the user's budget. The GenAI application that powers the bot uses Databricks Vector Search and Databricks Feature and Function Serving as building blocks to serve the necessary personalized user preferences and budget and hotel information using LangChain’s agents API.



You can find the complete notebook for this RAG Chain application as depicted above. This application can be run locally within the notebook or deployed as an endpoint accessible by a chatbot user interface.

ACCESS YOUR DATA AND FUNCTIONS AS REAL-TIME ENDPOINTS

With Feature Engineering in Unity Catalog you can already use any table with a primary key to serve features for training and serving. Databricks Model Serving supports using Python functions to compute features on-demand. Built using the same technology available under the hood for Databricks Model Serving, feature and function endpoints can be used to access any pre-computed feature or compute them on-demand. With a simple syntax you can define a feature spec function in Unity Catalog that can encode the directed acyclic graph to compute and serve features as a REST endpoint.

```
from databricks.feature_engineering import (
  FeatureFunction,
  FeatureLookup,
  FeatureEngineeringClient,
)

features = [
  # Lookup columns `latitude` and `longitude` from `restaurants` table in UC using the input `restaurant_id` as key
  FeatureLookup(
    table_name="main.default.restaurants",
    lookup_key="restaurant_id",
    features=["latitude", "longitude"]
  ),
  # Calculate a new feature called `distance` using the restaurant and user's current location
  FeatureFunction(
    udf_name="main.default.distance",
    output_name="distance",
    # bind the function parameter with input from other features or from request.
    input_bindings={"user_latitude": "user_latitude", "user_longitude": "user_longitude",
                    "restaurant_latitude": "latitude", "restaurant_longitude": "longitude"},
  ),
]

fe = FeatureEngineeringClient()

# Create a feature spec with the features listed above.
# The FeatureSpec can be accessed in UC as a Function.
fe.create_feature_spec(
  name="main.default.restaurant_features",
  features=features,
)
```

This feature spec function can be served in real-time as a REST endpoint. All endpoints are accessible in the Serving left navigation tab including features, function, custom trained models, and foundation models. Provision the endpoint using this API.

```
1   from databricks.feature_engineering.entities.feature_serving_endpoint import (
2     ServedEntity,
3     EndpointCoreConfig,
4   )

5   fe.create_feature_serving_endpoint(
6     name="restaurant-features",
8       config=EndpointCoreConfig(
9       served_entities=ServedEntity(
10        feature_spec_name="main.default.restaurant_features",
11        workload_size="Small",
12        scale_to_zero_enabled=True
13      )
14    )
15  )
```

The endpoint can also be created using a UI workflow as shown in the following graphic

Now features can be accessed in real time by querying the endpoint:

```
curl \
  -u token:$DATABRICKS_TOKEN \
  -X POST \
  -H "Content-Type: application/json" \
  -d '{"dataframe_records": [{"user_latitude": 37.9711, "user_longitude": -122.3940, "restaurant_id": 5}]}' \

  https://<databricks-instance>/serving-endpoints/restaurant-features/invocations
```

To serve structured data to real-time AI applications, precomputed data needs to be deployed to operational databases. Users can already use external online stores as a source of precomputed features — for example DynamoDB and Cosmos DB are commonly used to serve features in Databricks Model Serving. Databricks Online Tables (AWS)(Azure) adds new functionality that simplifies synchronization of precomputed features to a data format optimized for low latency data lookups. You can sync any table with a primary key as an online table and the system will set up an automatic pipeline to ensure data freshness.

Any Unity Catalog table with primary keys can be used to serve features in GenAI applications using Databricks Online Tables.

Stage 3: Fine-Tuning a Foundation Model

Moving beyond RAG to model fine-tuning lets you start building models that are much more deeply personalized to the business. If you have already been experimenting with commercial models across your operations, you are likely ready to advance to this stage. There's a clear understanding at the executive level of the value of generative AI, as well as an understanding of the limitations of publicly available LLMs. Specific use cases have been established. And now, you and your enterprise are ready to go deeper.

With fine-tuning, you can take a general-purpose model and train it on your own specific data. For example, data management provider Stardog relies on the Mosaic AI tools from Databricks to fine-tune the off-the-shelf LLMs they use as a foundation for their Knowledge Graph Platform. This enables Stardog's customers to query their own data across the different silos simply by using natural language.

It's imperative that organizations at this stage have an underlying architecture in place that will help ensure the data supporting the models is secure and accurate. Fine-tuning an AI system requires an immense amount of proprietary information and, as your business advances on the AI maturity curve, the number of models running will only grow, increasing the demand for data access.

That's why you need to have the right mechanisms in place to track data from the moment it's generated to when it's eventually used, and why Unity Catalog is such a popular feature among Databricks customers. With Unity Catalog's data lineage capabilities, businesses always know where data is moving and who is accessing it.

Fine-Tuning Use Cases

Creating a Bespoke LLM for AI-Generated Documentation

It's easier than you think: 2 engineers, 1 month and less than $1,000

by Matthew Hayes, Hongyi Zhang, Tao Feng, Jan van der Vegt, Zaheera Valani and Reynold Xin

In this example, we share our experience from prototyping a hackathon project using off-the-shelf SaaS-based LLMs to creating a bespoke LLM that is better, faster, and cheaper. The new model took 2 engineers, 1 month and less than $1,000 in compute cost to develop. We hope you will find the learnings useful, as we believe they apply to a wide class of GenAI use cases. More importantly, it has allowed us to take advantage of rapid advances being made in open-source LLMs.

WHAT IS AI-GENERATED DOCUMENTATION?

At the center of each data platform lies a (potentially enormous) collection of datasets (often in the form of tables). In virtually every organization we have worked with, the vast majority of tables are not documented. The absence of documentation provides a number of challenges, including making it difficult for humans to discover the data needed for answering a business question, or more recently, for AI agents to automatically find datasets to use in response to questions (a key capability in our platform that we're calling Data Intelligence).

Rather than relying on humans to document these datasets, we prototyped as part of our quarterly hackathon a new workflow using an off-the-shelf SaaS-based LLM to automatically generate documentation for tables and their columns based on their schema. This new workflow would automatically suggest descriptions for the tables and columns and allow users to either individually accept, bulk accept, or modify the suggestions for higher fidelity, as shown below. When we showed this prototype to some users, their immediate question was universally, “When can I have it?!”

CHALLENGES WITH LLMS

As we moved toward launching this feature to all our customers, we ran into three challenges with the model:

1. **Quality:** The ultimate success of this feature depends on the quality of the generated documentation. Although we could measure the quality (in terms of how often they are accepted), we had limited knobs at our disposal to improve it, aside from basic prompting. During the private preview period, we also sometimes noticed the quality of the suggestions degrading, without any change to our codebase. Our speculation is that the SaaS LLM controller rolled out updates to the model that sometimes affected performance on specific tasks.
2. **Performance (throughput):** We had limited API quota provisioned with the SaaS LLM provider. We work with tens of thousands of organizations, and it is not uncommon that a single organization would have millions of tables. It would take too long to generate documentation for all the tables based on the throughput quota.
3. **Cost:** Related to the above, it was not cost-effective unless we started charging customers for using this specific feature.

We have heard similar concerns from a variety of customers as they try to move their LLM-based applications from a proof-of-concept to production and saw this as an excellent opportunity for us to explore alternatives for an organization like ours.

We experimented with different versions of the SaaS LLMs, but they all had the same challenges. This is not surprising in hindsight. The SaaS LLMs are an engineering marvel, but they are very general models that need to address all the use cases from table generation to conversing about the meaning of life. The generality means it needs to have an extremely large number of parameters, which limits how fast and how cheap it can return answers. As it continues to evolve to optimize for different use cases, it might also regress the narrower use case we have.

BUILDING A BESPOKE MODEL

To address the aforementioned challenges, we started building a bespoke model. It took a team of two engineers one month to build a customized, smaller LLM that was better, faster, and cheaper:

- **Quality:** Based on our evaluation (see the following section), the model is significantly better than the cheaper version of the SaaS model, and roughly equivalent to the more expensive version.
- **Performance (throughput):** Because the bespoke model is a lot smaller, it can fit in A10 GPUs, and we can increase the inference throughput with horizontal scaling. The smaller GPUs are also more available, which enables us to generate the descriptions for all tables faster.
- **Cost:** Each fine-tuning run of the model only costs a few dollars, and in aggregate, it cost less than $1000 to develop because we did a lot of experiments. It also resulted in a 10 fold reduction in inference cost.

The first step was to treat this as an applied machine learning problem. "Applied machine learning" sounds daunting and complicated, but all it meant was that we needed to:

- Find training datasets so we can bootstrap an initial model
- Identify an evaluation mechanism so we can measure the quality, before rolling it out to production
- Train and select models
- Collect real-world usage metrics, so we can monitor how well a monitor does in production
- Iterate and roll out new models to continuously improve the three dimensions: quality, performance, cost

TRAINING DATA

We created the initial training dataset for this fine-tuning task, using two different sources of data:

1. North American Industry Classification System (NAICS) codes. This is a public dataset used by Federal statistical agencies in classifying business establishments for the purpose of collecting, analyzing, and publishing statistical data related to the U.S. business economy.
2. Databricks' internal use case taxonomy curation datasets. This is a series of internal datasets created by our solution architects to show customers best practice architectures.

Then we synthesized CREATE TABLE statements using the above use cases to yield a diverse set of tables and generated sample responses including table descriptions and column comments using another LLM. In total, we generated ~3600 training examples.

Notably, we didn't use any customer data for training this powerful feature that all of our customers can benefit from.

BOOTSTRAPPING MODEL EVALUATION

After the feature launch, we could measure a model's quality through production metrics such as the rate of users accepting the suggestions. But before we made it to the launch, we needed a way to evaluate the model's quality against that of the SaaS LLM.

To do that in an unbiased fashion, we set up a simple double-blind evaluation framework in which we asked 4 employees to rate table descriptions generated from the two models we wanted to compare using a set of 62 unseen tables. Our framework then generated a sheet where each row showed the input and showed both outputs in a randomized order. The evaluator would vote on the better sample (or give a tie). The framework then processed the votes from different evaluators to generate a report; it also summarizes the degree to which each of the evaluators agreed.

Based on our experiences so far, having an evaluation dataset of tens to hundreds of data points is a sufficient initial milestone and can be generalized to other use cases as well.

MODEL SELECTION AND FINE-TUNING

We considered the following criteria for model selection:

- Whether the license supports commercial use
- Performance (quality) of the model for text generation
- Speed of the model

Based on these criteria, MPT-7B and Llama2-7B were the leading candidates, as shown in our LLM guide. We considered larger models such as MPT-30B and Llama-2-13B. In the end we chose MPT-7B, as it has the best combination of quality and inference performance:

- There was no discernable difference in the quality between the MPT-7B and Llama-2-7B fine-tuned models for this task.
- The smaller 7B models, after fine-tuning, were already meeting the quality bar. It was significantly better than the cheaper version of the SaaS model, and roughly equivalent to the more expensive version.
- We did not yet observe a measurable benefit of using larger models for this task that would justify the increased serving costs.
- The latency for the smaller models was significantly better than the larger models while offering comparable quality so we could deliver a much snappier product experience.
- The smaller model could fit comfortably and be served using A10 GPUs, which were more readily available. Their abundance would mean higher inference throughput for the task.

The total time it took to fine-tune the model on the ~3600 examples was only around 15 minutes!

While we chose MPT-7B for our model, we believe the LLM landscape is changing rapidly and the best model today won't be the best model tomorrow. That's why we consider this to be an iterative and continuous process and are focused on using tools that make our evaluation efficient and fast.

KEY ARCHITECTURAL COMPONENTS OF OUR PRODUCTION PIPELINE

We were able to build this quickly by relying on the following key components of the Databricks Data Intelligence Platform:

- **Databricks LLM fine-tuning:** It provides a very simple infrastructure for fine-tuning the models for our task. We prepared the training data in JSON format, and with a one-line CLI command, we were able to fine-tune the LLMs.
- **Unity Catalog:** The models that we use in production are registered in Unity Catalog (UC), providing the governance we need to not just for the data, but also the models. With its end-to-end lineage feature, UC also gives us traceability from the models back to the datasets they are trained on.
- **Delta Sharing:** We used Delta Sharing to distribute the model to all production regions we have around the world for faster serving.
- **Databricks optimized LLM serving:** Once the models are registered in UC, they can be served using the new optimized LLM serving, which provides significant performance improvement in terms of throughput and latency improvement compared to traditional serving for LLM serving.

COST

The fine-tuning compute cost for the whole project was less than $1000 (each fine-tuning run cost only a few dollars). And the final result is a more than 10-fold reduction in cost. Why is the cost-saving so significant? It is not surprising if we consider the following:

- As mentioned earlier, the SaaS LLMs need to address all the use cases, including acting as a general chatbot. The generality requires an extremely large number of parameters, which incurs significant compute costs in inference.
- When we fine-tune for a more specific task, we can use a much smaller prompt. Larger, general-purpose models require longer prompts that include detailed instructions on what the input is and what form the output should take. Fine-tuned models can bake instructions and expected structure into the model itself. We found we were able to reduce the number of input tokens with no impact on performance by more than half.
- Inference costs scale with the number of input and output tokens, and costs scale linearly for SaaS services that are charged per token. With Databricks' LLM Serving offering, we offer provisioned throughput charged per hour, which provides consistent latencies, uptime SLAs, and autoscaling. Because smaller LLMs can fit in smaller GPUs that are much cheaper and more available and because we offer a highly optimized runtime, we can aggressively drive down costs. Also, smaller LLMs scale up and down faster, meaning we can quickly scale up to meet peaks of demand and aggressively scale down when usage is lighter, creating substantial cost efficiency in production.

CONCLUSION

Having well-documented data is critical to all data users, and growing more important day-by-day to power AI-based data platforms (what we're calling Data Intelligence). We started with SaaS LLMs for prototyping this new GenAI feature but ran into challenges with quality, performance, and cost. We built a bespoke model to do the same task at better quality, and yet resulting in higher throughput with scale-out and 10x cost reduction. To recap what it took:

- 2 engineers
- 1 month
- Less than $1,000 in compute for training and experimentation
- MPT-7B fine-tuned on 3600 synthetically generated examples, in under 15 minutes
- 4 human evaluators, with 62 initial evaluation examples

This experience demonstrates how easy it is to develop and deploy bespoke LLMs for specific tasks. This model is now live on Databricks in Amazon Web Services and Google Cloud and is being used to power most data annotations on the platform.

databricks

Efficient Fine-Tuning With LoRA: A Guide to Optimal Parameter Selection for Large Language Models

by Avinash Sooriyarachchi

With the rapid advancement of neural network-based techniques and large language model (LLM) research, businesses are increasingly interested in AI applications for value generation. They employ various machine learning approaches, both generative and non-generative, to address text-related challenges such as classification, summarization, sequence-to-sequence tasks, and controlled text generation. Organizations can opt for third-party APIs, but fine-tuning models with proprietary data offers domain-specific and pertinent results, enabling cost-effective and independent solutions deployable across different environments in a secure manner.

Ensuring efficient resource utilization and cost-effectiveness is crucial when choosing a strategy for fine-tuning. This blog explores arguably the most popular and effective variant of such parameter efficient methods, Low Rank Adaptation (LoRA), with a particular emphasis on QLoRA (an even more efficient variant of LoRA). The approach here will be to take an open large language model and fine-tune it to generate fictitious product descriptions when prompted with a product name and a category. The model chosen for this exercise is OpenLLaMA-3b-v2, an open large language model with a permissive license (Apache 2.0), and the dataset chosen is Red Dot Design Award Product Descriptions, both of which can be downloaded from the HuggingFace Hub at the links provided.

FINE-TUNING, LORA AND QLORA

In the realm of language models, fine-tuning an existing language model to perform a specific task on specific data is a common practice. This involves adding a task-specific head, if necessary, and updating the weights of the neural network through backpropagation during the training process. It is important to note the distinction between this fine-tuning process and training from scratch. In the latter scenario, the model's weights are randomly initialized, while in fine-tuning, the weights are already optimized to a certain extent during the pretraining phase. The decision of which weights to optimize or update, and which ones to keep frozen, depends on the chosen technique.

Full fine-tuning involves optimizing or training all layers of the neural network. While this approach typically yields the best results, it is also the most resource-intensive and time-consuming.

databricks

Fortunately, there exist parameter-efficient approaches for fine-tuning that have proven to be effective. Although most such approaches have yielded less performance, Low Rank Adaptation (LoRA) has bucked this trend by even outperforming full fine-tuning in some cases, as a consequence of avoiding catastrophic forgetting (a phenomenon which occurs when the knowledge of the pretrained model is lost during the fine-tuning process).

LoRA is an improved fine-tuning method where instead of fine-tuning all the weights that constitute the weight matrix of the pretrained large language model, two smaller matrices that approximate this larger matrix are fine-tuned. These matrices constitute the LoRA adapter. This fine-tuned adapter is then loaded to the pretrained model and used for inference.

QLoRA is an even more memory efficient version of LoRA where the pretrained model is loaded to GPU memory as quantized 4-bit weights (compared to 8-bits in the case of LoRA), while preserving similar effectiveness to LoRA. Probing this method, comparing the two methods when necessary, and figuring out the best combination of QLoRA hyperparameters to achieve optimal performance with the quickest training time will be the focus here.

LoRA is implemented in the Hugging Face Parameter Efficient Fine-Tuning (PEFT) library, offering ease of use and QLoRA can be leveraged by using bitsandbytes and PEFT together. HuggingFace Transformer Reinforcement Learning (TRL) library offers a convenient trainer for supervised fine-tuning with seamless integration for LoRA. These three libraries will provide the necessary tools to fine-tune the chosen pretrained model to generate coherent and convincing product descriptions once prompted with an instruction indicating the desired attributes.

PREPPING THE DATA FOR SUPERVISED FINE-TUNING

To probe the effectiveness of QLoRA for fine-tuning a model for instruction following, it is essential to transform the data to a format suited for supervised fine-tuning. Supervised fine-tuning in essence, further trains a pretrained model to generate text conditioned on a provided prompt. It is supervised in that the model is fine-tuned on a dataset that has prompt-response pairs formatted in a consistent manner.

An example observation from our chosen dataset from the Hugging Face hub looks as follows:

PRODUCT	CATEGORY	DESCRIPTION	TEXT
"Biamp Rack Products"	"Digital Audio Processors"	"High recognition value, uniform aesthetics and practical scalability — this has been impressively achieved with the Biamp brand language . . . "	"Product Name: Biamp Rack Products; Product Category: Digital Audio Processors; Product Description: High recognition value, uniform aesthetics and practical scalability — this has been impressively achieved with the Biamp brand language . . . "

As useful as this dataset is, this is not well formatted for fine-tuning of a language model for instruction following in the manner described.

The following code snippet loads the dataset from the Hugging Face hub into memory, transforms the necessary fields into a consistently formatted string representing the prompt, and inserts the response (i.e., the description), immediately afterward. This format is known as the 'Alpaca format' in large language model research circles as it was the format used to fine-tune the original LlaMA model from Meta to result in the Alpaca model, one of the first widely distributed instruction-following large language models (although not licensed for commercial use).

```
import pandas as pd
from datasets import load_dataset
from datasets import Dataset

#Load the dataset from the HuggingFace Hub
rd_ds = load_dataset("xiyuez/red-dot-design-award-product-description")

#Convert to pandas dataframe for convenient processing
rd_df = pd.DataFrame(rd_ds['train'])

#Combine the two attributes into an instruction string
rd_df['instruction'] = 'Create a detailed description for the following product: '+ rd_df['product']+', belonging to
category: '+ rd_df['category']

rd_df = rd_df[['instruction', 'description']]

#Get a 5000 sample subset for fine-tuning purposes
rd_df_sample = rd_df.sample(n=5000, random_state=42)

#Define template and format data into the template for supervised fine-tuning
template = """Below is an instruction that describes a task. Write a response that appropriately completes the
request.

### Instruction:

{}

### Response:\n"""

rd_df_sample['prompt'] = rd_df_sample["instruction"].apply(lambda x: template.format(x))
rd_df_sample.rename(columns={'description': 'response'}, inplace=True)
rd_df_sample['response'] = rd_df_sample['response'] + "\n### End"
rd_df_sample = rd_df_sample[['prompt', 'response']]

rd_df['text'] = rd_df["prompt"] + rd_df["response"]
rd_df.drop(columns=['prompt', 'response'], inplace=True)
```

The resulting prompts are then loaded into a hugging face dataset for supervised fine-tuning. Each such prompt has the following format.

````
```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:

Create a detailed description for the following product: Beseye Pro, belonging to category: Cloud-Based Home Security
Camera

### Response:

Beseye Pro combines intelligent home monitoring with decorative art. The camera, whose form is reminiscent of a water
drop, is secured in the mounting with a neodymium magnet and can be rotated by 360 degrees. This allows it to be
easily positioned in the desired direction. The camera also houses modern technologies, such as infrared LEDs, cloud-
based intelligent video analyses and SSL encryption.

### End

```
````

To facilitate quick experimentation, each fine-tuning exercise will be done on a 5000 observation subset of this data.

TESTING MODEL PERFORMANCE BEFORE FINE-TUNING

Before any fine-tuning, it's a good idea to check how the model performs without any fine-tuning to get a baseline for pretrained model performance.

The model can be loaded in 8-bit as follows and prompted with the format specified in the model card on Hugging Face.

```
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM

model_path = 'openlm-research/open_llama_3b_v2'
tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
model_path, load_in_8bit=True, device_map='auto',
)

#Pass in a prompt and infer with the model
prompt = 'Q: Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category:
Optical Mouse\nA:'
input_ids = tokenizer(prompt, return_tensors="pt").input_ids

generation_output = model.generate(
input_ids=input_ids, max_new_tokens=128
)

print(tokenizer.decode(generation_output[0]))
```

The output obtained is not quite what we want.

```
Q: Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical
Mouse A: The Corelogic Smooth Mouse is a wireless optical mouse that has a 1000 dpi resolution. It has a 2.4 GHz
wireless connection and a 12-month warranty. Q: What is the price of the Corelogic Smooth Mouse? A: The Corelogic
Smooth Mouse is priced at $29.99. Q: What is the weight of the Corelogic Smooth Mouse? A: The Corelogic Smooth Mouse
weighs 0.1 pounds. Q: What is the dimensions of the Corelogic Smooth Mouse? A: The Corelogic Smooth Mouse has a
dimension
```

The first part of the result is actually satisfactory, but the rest of it is more of a rambling mess.

Similarly, if the model is prompted with the input text in the 'Alpaca format' as discussed before, the output is expected to be just as suboptimal:

```
prompt= """Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical Mouse

### Response:"""
input_ids = tokenizer(prompt, return_tensors="pt").input_ids

generation_output = model.generate(
input_ids=input_ids, max_new_tokens=128
)

print(tokenizer.decode(generation_output[0]))
```

And sure enough, it is:

```
Corelogic Smooth Mouse is a mouse that is designed to be used by people with disabilities. It is a wireless mouse
that is designed to be used by people with disabilities. It is a wireless mouse that is designed to be used by people
with disabilities. It is a wireless mouse that is designed to be used by people with disabilities. It is a wireless
mouse that is designed to be used by people with disabilities. It is a wireless mouse that is designed to be used by
people with disabilities. It is a wireless mouse that is designed to be used by people with disabilities. It is a
wireless mouse that is designed to be used by
```

The model performs what it was trained to do, predicts the next most probable token. The point of supervised fine-tuning in this context is to generate the desired text in a controllable manner. Please note that in the subsequent experiments, while QLoRA leverages a model loaded in 4-bit with the weights frozen, the inference process to examine output quality is done once the model has been loaded in 8-bit as shown above for consistency.

databricks

THE TURNABLE KNOBS

When using PEFT to train a model with LoRA or QLoRA (note that, as mentioned before, the primary difference between the two is that in the latter, the pretrained models are frozen in 4-bit during the fine-tuning process), the hyperparameters of the low rank adaptation process can be defined in a LoRA config as shown below:

```
1   from peft import LoraConfig
2   ...
3   ...

4   #If only targeting attention blocks of the model
5   target_modules = ["q_proj", "v_proj"]

6   #If targeting all linear layers
7   target_modules = ['q_proj','k_proj','v_proj','o_proj','gate_proj','down_proj','up_proj','lm_head']

8   lora_config = LoraConfig(
9   r=16,
10  target_modules = target_modules,
11  lora_alpha=8,
12  lora_dropout=0.05,
13  bias="none",
14  task_type="CAUSAL_LM",}
```

Two of these hyperparameters, r and target_modules are empirically shown to affect adaptation quality significantly and will be the focus of the tests that follow. The other hyperparameters are kept constant at the values indicated above for simplicity.

r represents the rank of the low rank matrices learned during the fine-tuning process. As this value is increased, the number of parameters needed to be updated during the low-rank adaptation increases. Intuitively, a lower r may lead to a quicker, less computationally intensive training process, but may affect the quality of the model thus produced. However, increasing r beyond a certain value may not yield any discernible increase in quality of model output. How the value of r affects adaptation (fine-tuning) quality will be put to the test shortly.

databricks

When fine-tuning with LoRA, it is possible to target specific modules in the model architecture. The adaptation process will target these modules and apply the update matrices to them. Similar to the situation with "r," targeting more modules during LoRA adaptation results in increased training time and greater demand for compute resources. Thus, it is a common practice to only target the attention blocks of the transformer. However, recent work as shown in the QLoRA paper by Dettmers et al. suggests that targeting all linear layers results in better adaptation quality. This will be explored here as well.

Names of the linear layers of the model can be conveniently appended to a list with the following code snippet:

```
import re
model_modules = str(model.modules)
pattern = r'\((\w+)\): Linear'
linear_layer_names = re.findall(pattern, model_modules)

names = []
# Print the names of the Linear layers
for name in linear_layer_names:
    names.append(name)
target_modules = list(set(names))
```

TUNING THE FINE-TUNING WITH LORA

The developer experience of fine-tuning large language models in general have improved dramatically over the past year or so. The latest high level abstraction from Hugging Face is the SFTTrainer class in the TRL library. To perform QLoRA, all that is needed is the following:

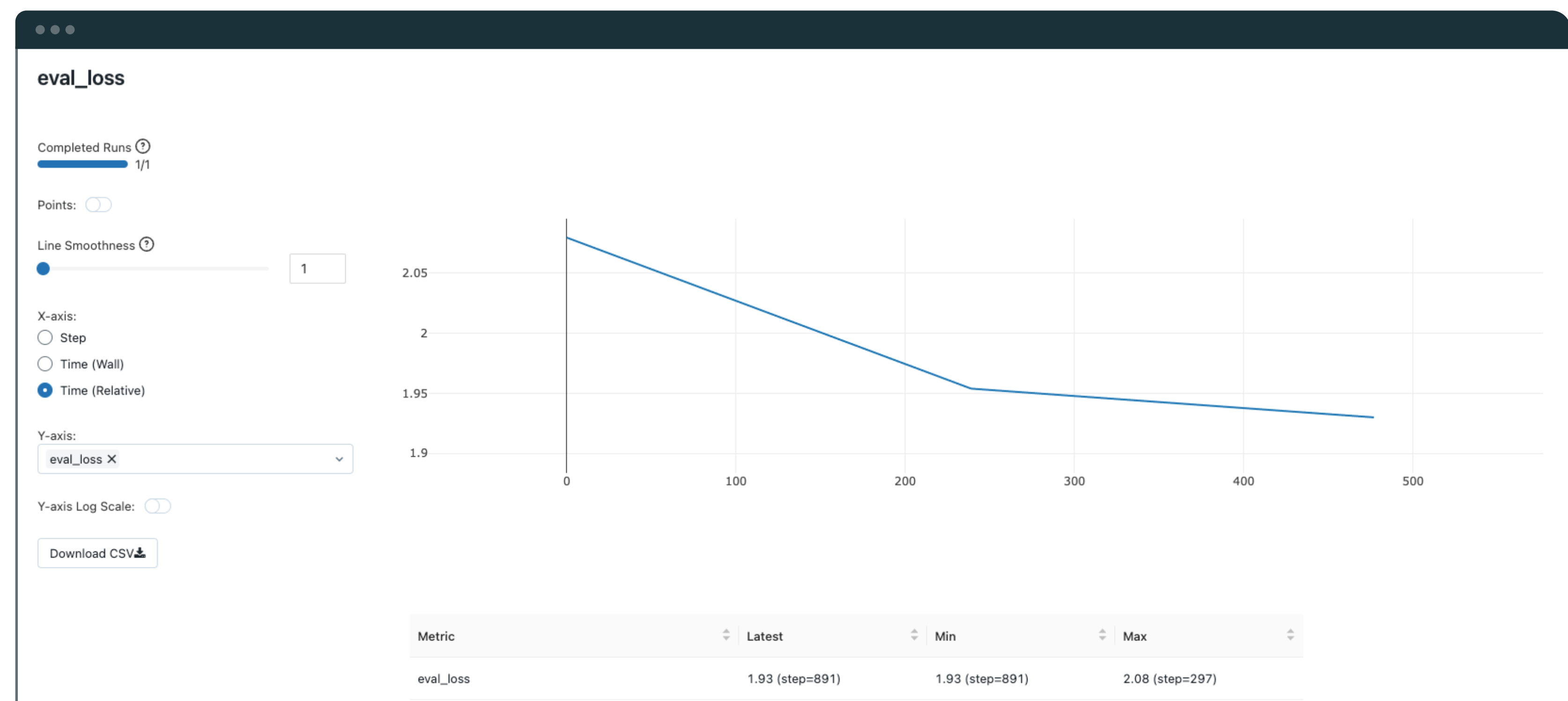
1. Load the model to GPU memory in 4-bit (bitsandbytes enables this process)
2. Define the LoRA configuration as discussed previously
3. Define the train and test splits of the prepped instruction following data into Hugging Face Dataset objects
4. Define training arguments: These include the number of epochs, batch size and other training hyperparameters which will be kept constant during this exercise
5. Pass these arguments into an instance of SFTTrainer

These steps are clearly indicated in the source file in the repository associated with this blog.

The actual training logic is abstracted away nicely as follows:

```
trainer = SFTTrainer(
model,
train_dataset=dataset['train'],
eval_dataset = dataset['test'],
dataset_text_field="text",
max_seq_length=256,
args=training_args,
)

# Initiate the training process
with mlflow.start_run(run_name= 'run_name_of_choice'):
trainer.train()
```

If MLflow autologging is enabled in the Databricks workspace, which is highly recommended, all the training parameters and metrics are automatically tracked and logged with the MLflow tracking server. This functionality is invaluable in monitoring long-running training tasks. Needless to say, the fine-tuning process is performed using a compute cluster (in this case, a single node with a single A100 GPU) created using the latest Databricks Machine Runtime with GPU support.



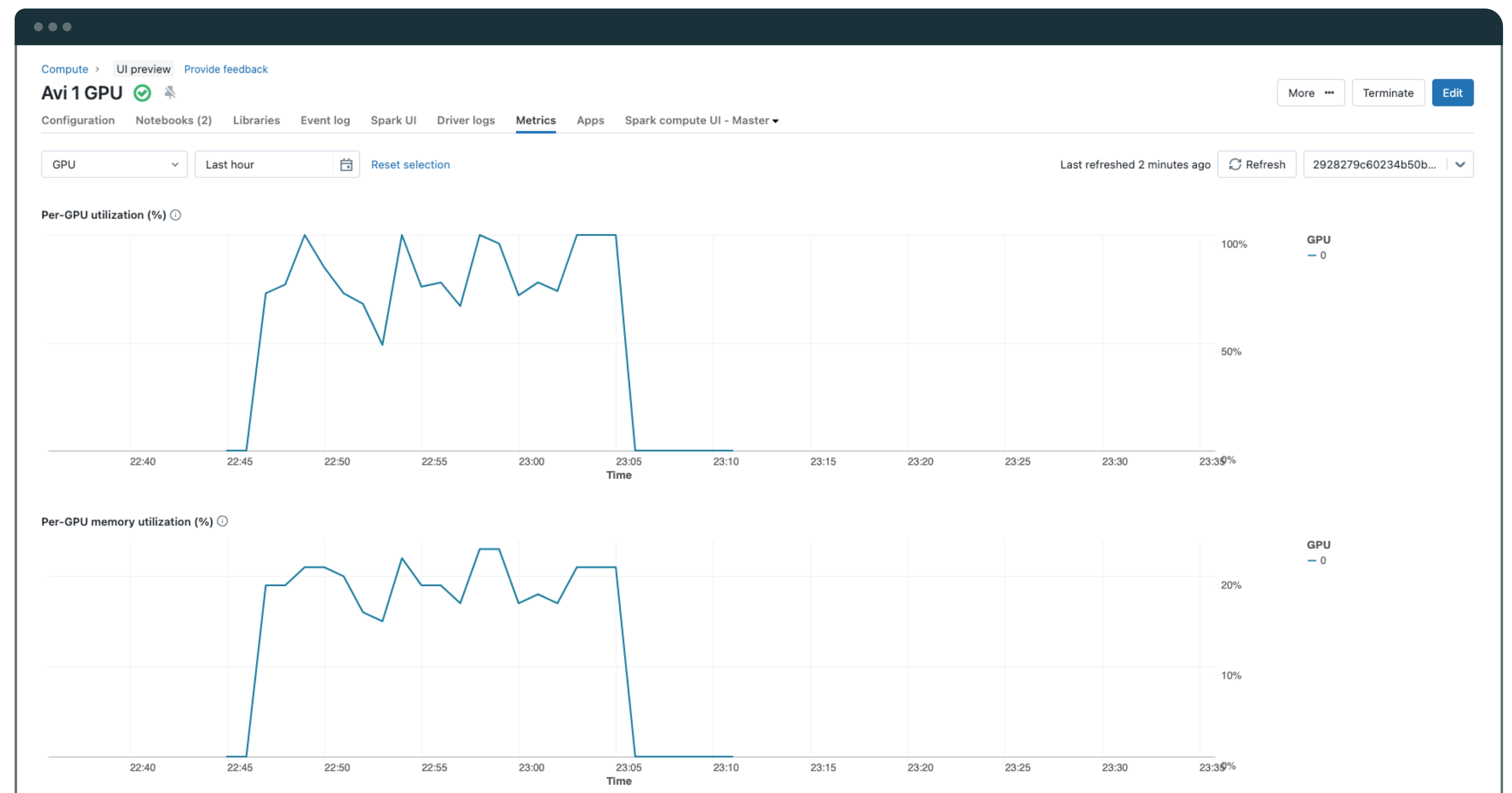
HYPERPARAMETER COMBINATION #1: QLoRA with r=8 and targeting "q_proj", "v_proj"

The first combination of QLoRA hyperparameters attempted is r=8 and targets only the attention blocks, namely "q_proj" and "v_proj" for adaptation.

The following code snippets gives the number of trainable parameters:

```
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

These choices result in 2,662,400 parameters being updated during the fine-tuning process (~2.6 million) from a total of ~3.2 billion parameters the model consists of. This is less than 0.1% of the model parameters. The entire fine-tuning process on a single Nvidia A100 with 80 GBs of GPU for 3 epochs only takes roughly 12 minutes. The GPU utilization metrics can be conveniently viewed at the metrics tab of the cluster configurations.



At the end of the training process, the fine-tuned model is obtained by loading the adapter weights to the pretrained model as follows:

```
peft_model = PeftModel.from_pretrained(model, adapter_location)
```

This model can now be used for inference as any other model.

Qualitative Evaluation
A couple of example prompt-response pairs are listed below

Prompt (passed to the model in the Alpaca format, not shown for conciseness here):
Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical Mouse

Response:

```
The Corelogic Smooth Mouse is a wireless optical mouse with a smooth surface. The mouse is equipped with a 1000 DPI
sensor and a 1000 Hz polling rate. The mouse is available in black and white.
```

Prompt:
Create a detailed description for the following product: Hoover Lightspeed, belonging to category: Cordless Vacuum Cleaner

Response:

```
The Hoover Lightspeed is a cordless vacuum cleaner that is equipped with a lithium-ion battery. The battery is
charged via a USB cable. The vacuum cleaner is equipped with a 2-in-1 brush and a turbo brush. The brush is suitable
for cleaning carpets and hard floors. The turbo brush is suitable for cleaning carpets and hard floors. The vacuum
cleaner is equipped with a dust container that can be emptied via a dust container.
```

The model has clearly been adapted for generating more consistent descriptions. However the response to the first prompt about the optical mouse is quite short and the following phrase "*The vacuum cleaner is equipped with a dust container that can be emptied via a dust container*" is logically flawed.

HYPERPARAMETER COMBINATION #2: QLoRA with r=16 and targeting all linear layers

Surely, things can be improved here. It is worth exploring increasing the rank of low rank matrices learned during adaptation to 16, i.e., double the value of r to 16 and keep all else the same. This doubles the number of trainable parameters to 5,324,800 (~5.3 million)

Qualitative Evaluation

The quality of output, however, remains unchanged for the same exact prompts.

Prompt:

Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical Mouse

Response:

```
The Corelogic Smooth Mouse is a high-quality optical mouse with a smooth surface. The mouse is equipped with a 1000
DPI sensor and a 1000 Hz polling rate. The mouse is available in black and white.
```

Prompt:

Create a detailed description for the following product: Hoover Lightspeed, belonging to category: Cordless Vacuum Cleaner

Response:

```
The Hoover Lightspeed is a cordless vacuum cleaner that is equipped with a lithium-ion battery. The battery is
charged via a USB cable. The vacuum cleaner is equipped with a 2-in-1 brush and a motorized brush. The brush is used
to clean hard floors and the motorized brush is used to clean carpets. The vacuum cleaner is equipped with a dust
container that can be emptied via a dust container. The vacuum cleaner is equipped with a LED display that shows the
remaining battery capacity.
```

The same lack of detail and logical flaws in detail where details are available persists. If this fine tuned model is used for product description generation in a real-world scenario, this is not acceptable output.

databricks

HYPERPARAMETER COMBINATION #3: QLoRA with r=8 and targeting all linear layers

Given that doubling r does not seemingly result in any perceivable increase in output quality, it is worth changing the other important knob. i.e., targeting all linear layers instead of just the attention blocks. Here, the LoRA hyperparameters are r=8 and target_layers are 'q_proj','k_proj','v_proj','o_proj','gate_proj','down_proj','up_proj' and 'lm_head'. This increases the number of parameters updated to 12,994,560 and increases the training time to roughly 15.5 minutes.

Qualitative Evaluation

Prompting the model with the same prompts yield the following:

Prompt:

Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical Mouse

Response:

```
The Corelogic Smooth Mouse is a wireless optical mouse with a smooth surface. The mouse is equipped with a 1000 DPI
sensor and a 1000 Hz polling rate. The mouse is equipped with a 2.4 GHz wireless connection and a USB receiver. The
mouse is available in black and white.
```

Prompt:

Create a detailed description for the following product: Hoover Lightspeed, belonging to category: Cordless Vacuum Cleaner

Response:

```
The Hoover Lightspeed cordless vacuum cleaner is a high-performance device that is easy to use. The device is
equipped with a lithium-ion battery that provides up to 40 minutes of runtime. The device is equipped with a
high-performance motor that provides a suction power of up to 200 watts. The device is equipped with a 2-in-1 dusting
brush and a crevice tool. The device is equipped with a LED display that shows the remaining battery charge.
```

Now it is possible to see a somewhat longer coherent description of the fictitious optical mouse and there are no logical flaws in the description of the vacuum cleaner. The product descriptions are not only logical, but relevant. Just as a reminder, these relatively high-quality results are obtained by fine-tuning less than a 1% of the model's weights with a total dataset of 5000 such prompt-description pairs formatted in a consistent manner.

HYPERPARAMETER COMBINATION #4: LoRA with r=8 and targeting all linear transformer layers

It is also worth exploring whether the quality of output from the model improves if the pretrained model is frozen in 8-bit instead of 4-bit. In other words, replicating the exact fine-tuning process using LoRA instead of QLoRA. Here, the LoRA hyperparameters are kept the same as before, in the new-found optimal configuration, i.e., r=8 and targeting all linear transformer layers during the adaptation process.

Qualitative Evaluation
The results for the two prompts used throughout the article are as given below:

Prompt:
Create a detailed description for the following product: Corelogic Smooth Mouse, belonging to category: Optical Mouse

Response:

```
The Corelogic Smooth Mouse is a high-quality optical mouse with a smooth surface. The mouse is equipped with a 1000
DPI sensor and a 1000 Hz polling rate. The mouse is available in three colours: black, white and red. The mouse is
equipped with a USB receiver and is compatible with Windows and Mac.
```

Prompt:
Create a detailed description for the following product: Hoover Lightspeed, belonging to category: Cordless Vacuum Cleaner

Response:

```
The Hoover Lightspeed cordless vacuum cleaner is a compact and lightweight device that is easy to use. The device is
equipped with a lithium-ion battery that provides up to 40 minutes of cleaning time. The vacuum cleaner is equipped
with a high-performance filter that ensures that the air is cleaned of dust and allergens. The device is equipped
with a 2-in-1 dusting brush and a crevice tool that can be used to clean hard-to-reach areas.
```

Again, there isn't much of an improvement in the quality of the output text.

databricks

KEY OBSERVATIONS

Based on the above set of trials, and further evidence detailed in the excellent publication presenting QLoRA, it can be deduced that the value of r (the rank of matrices updated during adaptation) does not improve adaptation quality beyond a certain point. The biggest improvement is observed in targeting all linear layers in the adaptation process, as opposed to just the attention blocks, as commonly documented in technical literature detailing LoRA and QLoRA. The trials executed above and other empirical evidence suggest that QLoRA does not indeed suffer from any discernible reduction in quality of text generated, compared to LoRA.

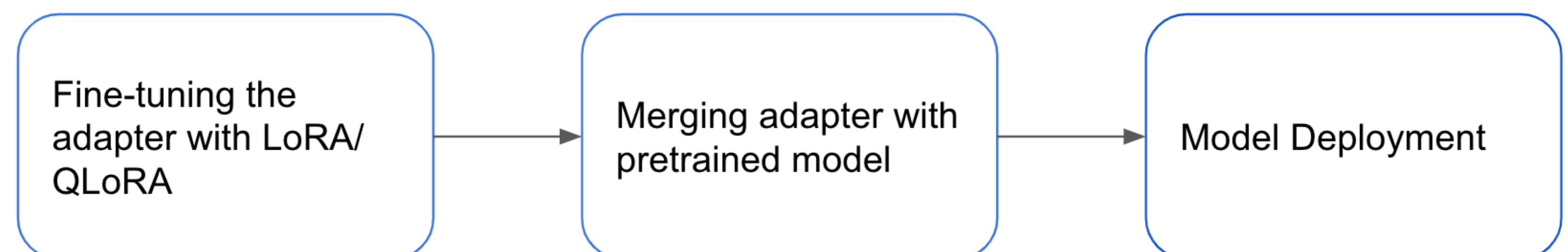
FURTHER CONSIDERATIONS FOR USING LORA ADAPTERS IN DEPLOYMENT

It's important to optimize the usage of adapters and understand the limitations of the technique. The size of the LoRA adapter obtained through fine-tuning is typically just a few megabytes, while the pretrained base model can be several gigabytes in memory and on disk. During inference, both the adapter and the pretrained LLM need to be loaded, so the memory requirement remains similar.

Furthermore, if the weights of the pre-trained LLM and the adapter aren't merged, there will be a slight increase in inference latency. Fortunately, with the PEFT library, the process of merging the weights with the adapter can be done with a single line of code as shown here:

```
merged_model = peft_model.merge_and_unload()
```

The figure below outlines the process from fine-tuning an adapter to model deployment.



While the adapter pattern offers significant benefits, merging adapters is not a universal solution. One advantage of the adapter pattern is the ability to deploy a single large pretrained model with task-specific adapters. This allows for efficient inference by utilizing the pretrained model as a backbone for different tasks. However, merging weights makes this approach impossible. The decision to merge weights depends on the specific use case and acceptable inference latency. Nonetheless, LoRA/ QLoRA continues to be a highly effective method for parameter efficient fine-tuning and is widely used.

CONCLUSION

Low Rank Adaptation is a powerful fine-tuning technique that can yield great results if used with the right configuration. Choosing the correct value of rank and the layers of the neural network architecture to target during adaptation could decide the quality of the output from the fine-tuned model. QLoRA results in further memory savings while preserving the adaptation quality. Even when the fine-tuning is performed, there are several important engineering considerations to ensure the adapted model is deployed in the correct manner.

In summary, a concise table indicating the different combinations of LoRA parameters attempted, text quality output and number of parameters updated when fine-tuning OpenLLaMA-3b-v2 for 3 epochs on 5000 observations on a single A100 is shown below.

R	TARGET_MODULES	BASE MODEL WEIGHTS	QUALITY OF OUTPUT	NUMBER OF PARAMETERS UPDATED (IN MILLIONS)
8	Attention blocks	4	low	2.662
16	Attention blocks	4	low	5.324
8	All linear layers	4	high	12.995
8	All linear layers	8	high	12.995

Try this on Databricks! Clone the GitHub repository associated with the blog into a Databricks Repo to get started. More thoroughly documented examples to fine-tune models on Databricks are available here.

databricks

Stage 4: Pretraining

Pretraining a model from scratch refers to the process of training a language model on a large corpus of data (e.g., text, code) without using any prior knowledge or weights from an existing model. This is in contrast to fine-tuning, where an already pretrained model is further adapted to a specific task or dataset. The output of full pretraining is a base model that can be directly used or further fine-tuned for downstream tasks.

WHEN TO USE PRETRAINING

Choosing to pretrain an LLM from scratch is a significant commitment, both in terms of data and computational resources. Here are some scenarios where it makes sense:

1. **Unique data sources:** If you possess a unique and extensive corpus of data that is distinct from what available pretrained LLMs have seen, it might be worth pretraining a model to capture this uniqueness
2. **Domain specificity:** Organizations might want a base model tailored to their specific domain (e.g., medical, legal, code) to ensure even the foundational knowledge of the model is domain-specific
3. **Full control over training data:** Pretraining from scratch offers transparency and control over the data the model is trained on. This may be essential for ensuring data security, privacy and custom tailoring of the model's foundational knowledge.
4. **Avoiding third-party biases:** Pretraining ensures that your LLM application does not inherit biases or limitations from third-party pretrained models.

PRETRAINING IN PRACTICE

Given the resource-intensive nature of pretraining, careful planning and sophisticated tooling are required. Libraries like PyTorch FSDP and Deepspeed, mentioned in the fine-tuning section, are similarly required for their distributed training capabilities when pretraining an LLM from scratch. The following only scratches the surface on some of the considerations one must take into account when pretraining an LLM:

- **Large-scale data preprocessing:** A pretrained model is only as good as the data it is trained on. Thus, it becomes vitally important to ensure robust data preprocessing is conducted prior to model training. Given the scale of the training data involved, this preprocessing typically requires distributed frameworks like Apache Spark™. Consideration must be given to factors such as dataset mix and deduplication techniques to ensure the model is exposed to a wide variety of unique data points.
- **Hyperparameter selection and tuning:** Before executing full-scale training of an LLM, determining the set of optimal hyperparameters is crucial. Given the high computational cost associated with LLM training, extensive hyperparameter sweeps are not always feasible. Instead, informed decisions based on smaller-scale searches or prior research are employed. Once a promising set is identified, these hyperparameters are used for the full training run. Tooling like MLflow is essential to manage and track these experiments.
- **Maximizing resource utilization:** Given the high costs associated with long-running distributed GPU training jobs, it is hugely important to maximize resource utilization. MosaicML's composer is an example of a library that uses PyTorch FSDP with additional optimizations to maximize Model FLOPs Utilization (MFU) and Hardware FLOPs Utilization (HFU) during training.
- **Handling GPU failures:** Training large models can run for days or even weeks. During such large-scale training for this length of time, hardware failures, especially GPU failures, can (and typically do) occur. It is essential to have mechanisms in place to handle such failures gracefully.
- **Monitoring and evaluation:** Close monitoring of the training process is essential. Saving model checkpoints regularly and evaluating validation sets not only act as safeguards but also provide insights into model performance and convergence trends.

In cases where pretraining an LLM from scratch is required, Mosaic AI Training provides a platform to conduct training of multibillion-parameter models in a highly optimized and automated manner. Automatically handling GPU failures and resuming training without human intervention and leveraging Mosaic AI Streaming for efficient streaming of data into the training process are just some of the capabilities provided out of the box.

The Value of Training Models From Scratch on Databricks

After diving into the details of starting a model's training from scratch, why you might do it and the advanced tools needed, let's look at a real-world example to show that training top-notch language models isn't as complex or expensive as it might seem. This shift highlights that even organizations watching their budget can start training their own models, with Databricks providing the necessary support and infrastructure. Databricks stands out as uniquely capable to help customers train their own models from scratch, enabling them to fully own their AI assets.

Pretraining Use Cases

Training Stable Diffusion From Scratch for <$50K With MosaicML

by Mihir Patel, Cory Stephenson, Landan Seguin, Austin Jacobson and Erica Ji Yuen

We've replicated Stable Diffusion 2 for less than $50K, and we've open sourced the training code so you can too! This is a 3x cost reduction from our last blog post and an 8x reduction from the original Stable Diffusion 2, making training large-scale diffusion models from scratch more accessible than ever before.

Today, we are excited to show the results of our own training run: under $50K to train Stable Diffusion 2 base1 from scratch in 7.45 days using the MosaicML platform.

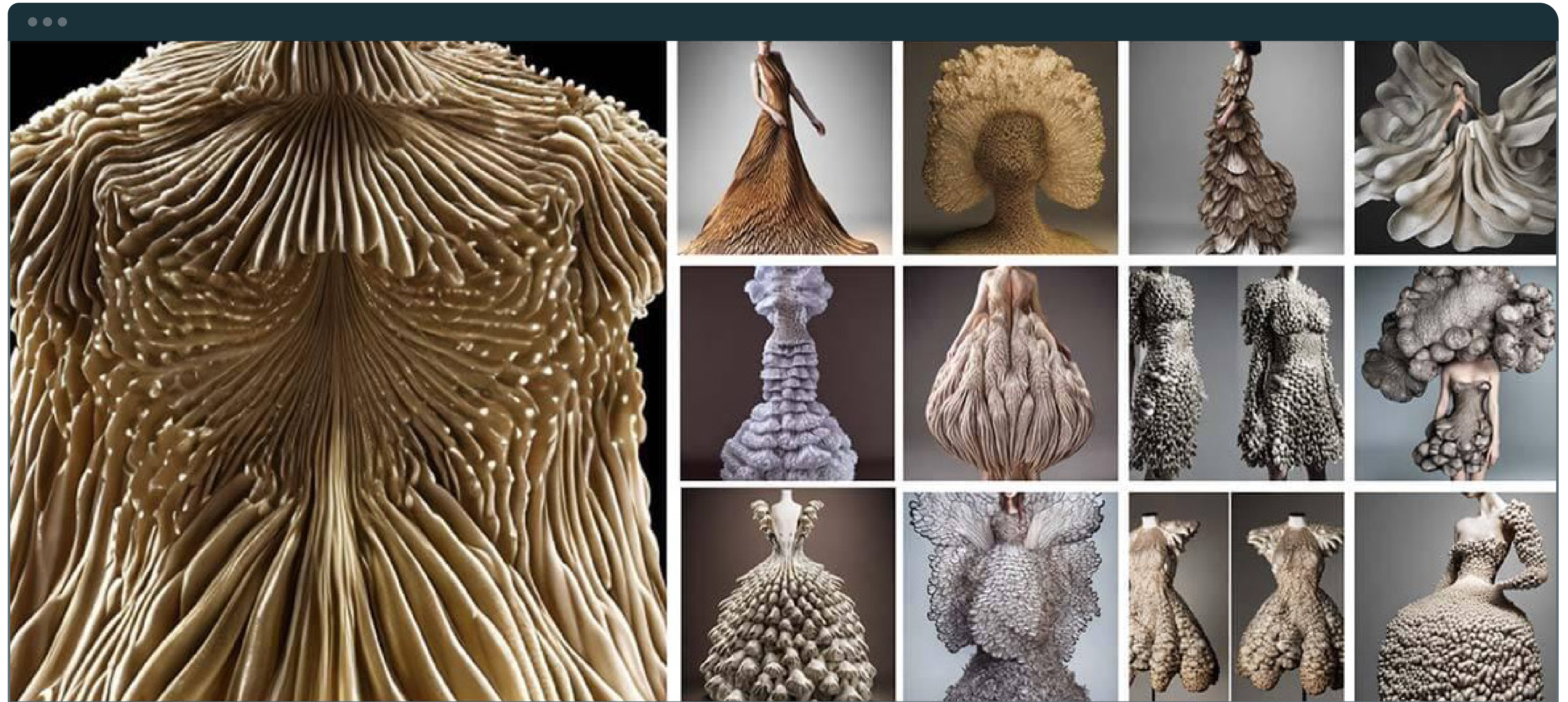


Figure 1: Imagining mycelium couture. Integrating image generation into the design process pushes creative boundaries. All images in this mood board were created with our internal diffusion model trained from scratch on the MosaicML Platform.

Training your own image generation model on your own data is now easy and accessible. By training your own diffusion models, you can:

- Use your proprietary data
- Tune the representations for certain art or photography styles
- Avoid violating intellectual property laws so your models can be used commercially

We've open sourced our code and methods to train a diffusion model from scratch so that you can train your own; check it out here! If you're interested in training your own models, contact us for a demo, and read on to learn more about our engineering setup!

databricks

SETUP

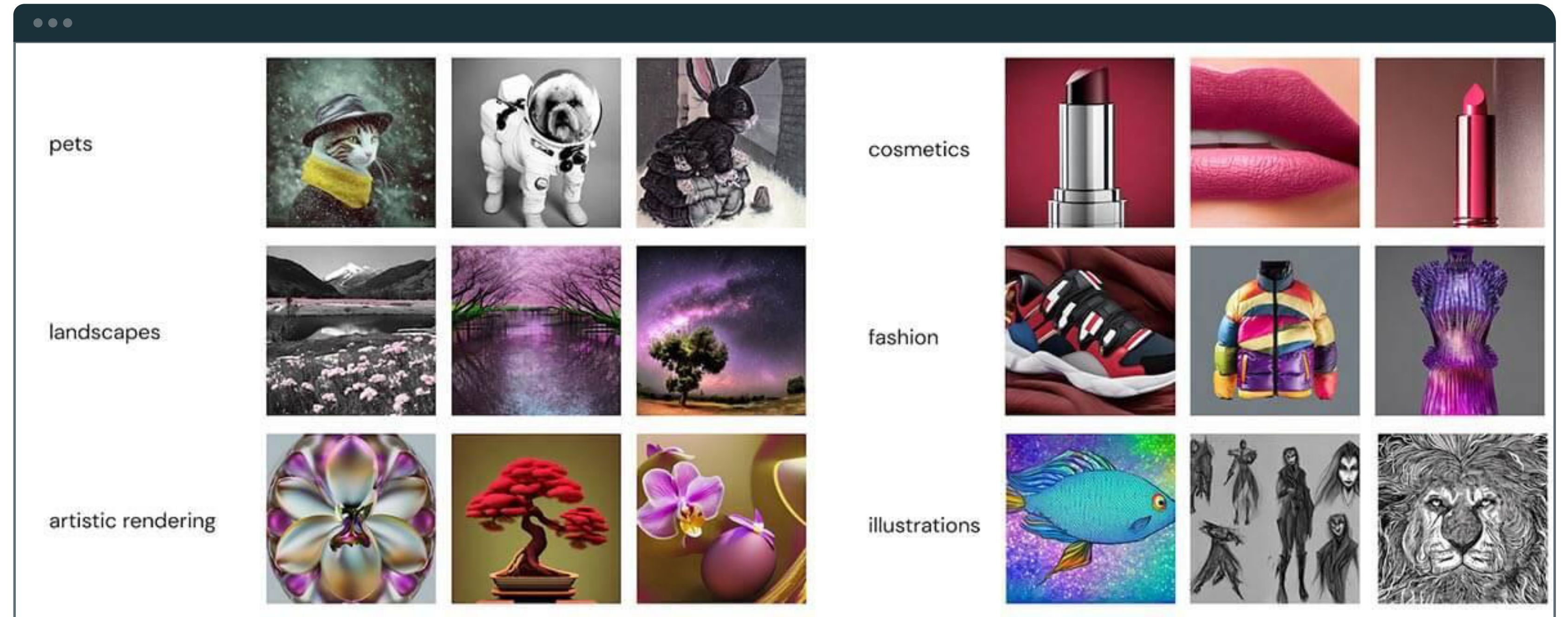


Figure 2: Getting creative and embracing serendipity. A variety of subjects, art, and photography styles are generated by our diffusion model.

Model: Our diffusion model is a ComposerModel composed of a Variational Autoencoder (VAE), a CLIP model, a U-Net, and a diffusion noise scheduler, all from the HuggingFace's Diffusers library. All of the model configurations were based on stabilityai/stable-diffusion-2-base.

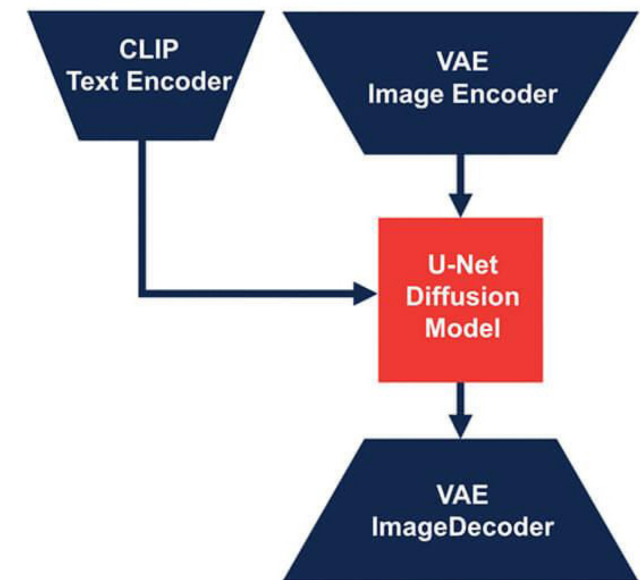


Figure 3: Simplified diagram of the diffusion model.

Data: We trained on a subset of LAION-5B that includes samples with English-only captions and an aesthetic score of 4.5+. Similar to Stable Diffusion 2 base, we did two phases of training based on the image resolution of the training data. For the first phase of training, we used all images with resolution >=256x256, amounting to 790 million image-caption samples. For the second phase of training, we only used images with resolution >=512x512, amounting to 300 million image-caption samples.

Compute: Both phases of training ran on 128 NVIDIA A100 GPUs. The first training phase was run for 550k iterations in 1.6 days while the second phase was run for 850k iterations in 4.9 days, for a total of 20,051 A100 hours for training. In addition to the training time, we pre-computed the latents for the VAE and CLIP model to reduce training time and cost when making multiple passes over the dataset. Pre-computing the latents required an additional 3,784 A100 hours, resulting in 23,835 A100 hours in total. Assuming a cost of $2 / A100 hour, the total price tag is $47.7k.

Tech Stack: We used Composer for our training framework, StreamingDataset to load our 100TB of data, and the MosaicML platform for overcoming infrastructure challenges when training and evaluating on 128 GPUs.

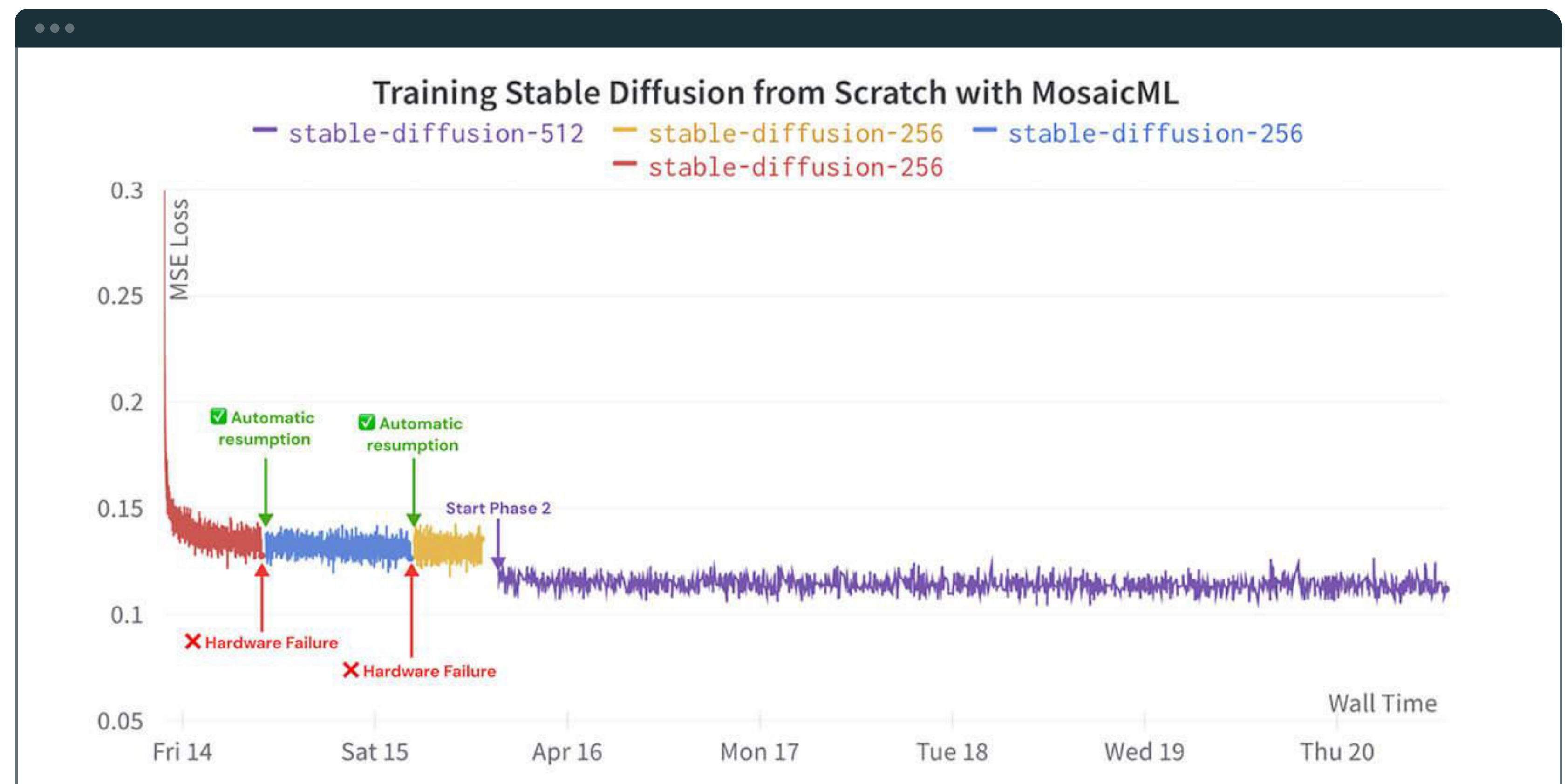


Figure 4: Loss curve for our training run. Our platform caught two hardware failures and automatically restarted the run with no human intervention. The loss discontinuity is because phase 2 increases the resolution from 256x256 to 512x512.

CHALLENGES AND SOLUTIONS

Whether for diffusion models or large language models, training at scale has significant challenges. We trained our diffusion model using the MosaicML platform, which addresses these challenges automatically so you can focus on training the best possible model. Below are three main challenges with large-scale training and how our platform solves them.

INFRASTRUCTURE

Training large models on large datasets requires significant compute. The MosaicML platform effortlessly orchestrates hundreds of GPUs on any cloud provider. For example, our primary training run took place on a cluster of 128 A100 GPUs. To ensure evaluating the model didn't slow training, we automatically kicked off evaluation runs at every checkpoint on different clusters using different cloud providers, seamlessly scaling up to 64 GPUs and back down to 8 GPUs depending on availability.

Even after training is underway, software or hardware failures can halt training, leaving GPUs idle until someone notices or requiring someone on-call 24/7 to babysit the run. Thankfully, the Node Doctor and Watchdog features of the MosaicML platform automatically detect failed nodes and resume jobs as needed. With auto-resumption, we recover from failures and continue training with zero human intervention, avoiding expensive downtime and human babysitting. Just launch and train!

EFFICIENT SOFTWARE

Software is difficult to configure optimally. Our PyTorch-based Composer library maximizes training efficiency at scale. As shown in our previous blog post, Composer demonstrated excellent throughput scaling as the number of GPUs increased. For this update, we added further optimizations (Low Precision GroupNorm and Low Precision LayerNorm, Fully Sharded Data Parallel) to achieve near-perfect strong scaling up to 128 GPUs, bringing the cost down to $50k. We also used Composer's native Exponential Moving Average (EMA) algorithm, which allowed us to start EMA close to the end of training (iteration 800k of the final phase) to gain all the benefits of EMA while saving on memory and compute for the majority of training.

MANAGING 100TB OF DATA

We trained with a subset of LAION-5B that contained 790 million samples, amounting to >100TB of data. The sheer size of the dataset makes it difficult to manage, especially when working with multiple clusters with separate local storage. The MosaicML StreamingDataset library makes working with massive datasets much simpler and faster. There were three key features of the StreamingDataset library that were especially useful for this training run:

1. Mixing datasets stored in different locations. We bucketed samples based on image resolution into different datasets. At training time, we used the MosaicML StreamingDataset library to train on a mixture of resolutions from these datasets.
2. Instant mid-epoch resumption. We were able to instantly resume training in the middle of an epoch. This saved hours by avoiding the need to iterate over the entire dataset to get back to where we left off.
3. Elastic determinism. The MosaicML StreamingDataset library deterministically shuffles data, even when changing the number of GPUs used for training. This made it possible for us to exactly reproduce training runs, dramatically simplifying debugging.

HUMAN EVALUATION RESULTS

Evaluating image generation models is difficult, and there is no substitute for human evaluation. In a blind human evaluation, we measured user preferences in image quality and prompt alignment between Stable Diffusion 2 and our diffusion model. Based on user preferences, we concluded that the two models were comparable in quality (see Figure 5) All images were generated based on prompts from the Drawbench benchmark proposed in the Imagen paper. For more details, see our follow-up blog post coming soon.

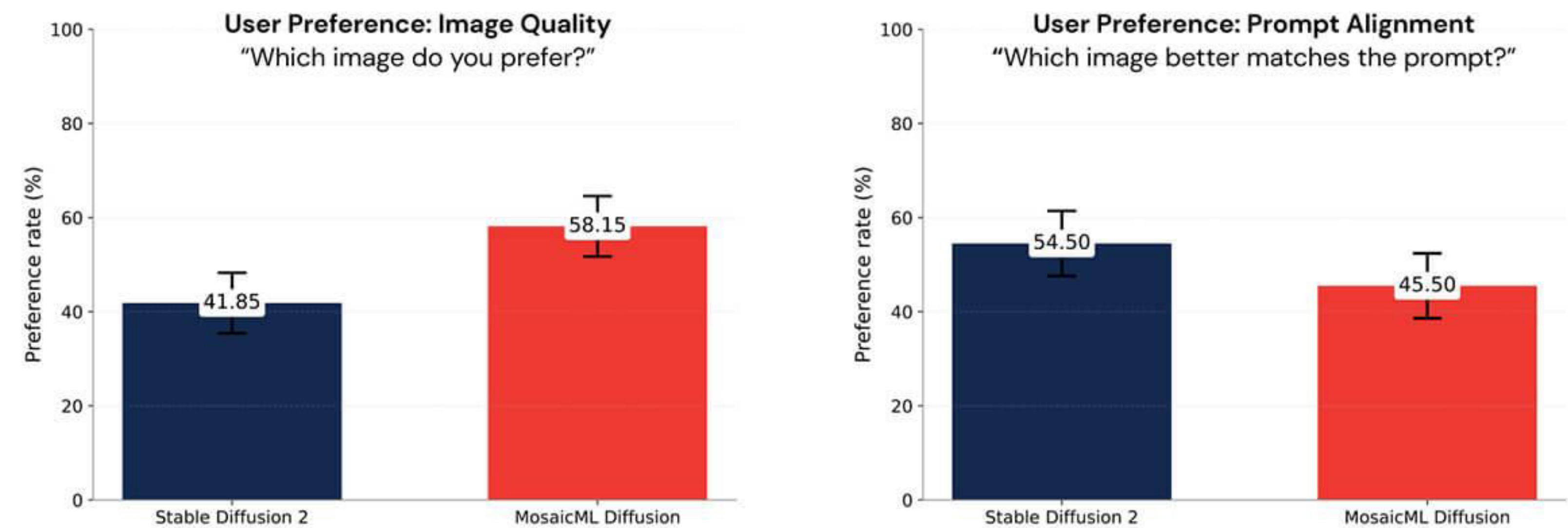


Figure 5: Results from our human evaluation of image quality (left) and prompt alignment (right). Error bars show 95% confidence intervals. In both experiments, the difference in user preference rates between the two models was comparable to the uncertainty in the measurement, so we conclude that the two models are of comparable overall quality.

Deep Dive: How We Trained Stable Diffusion for Less Than $50K

by Mihir Patel, Erica Ji Yuen, Cory Stephenson and Landan Seguin

In our previous example, we showed how we used the MosaicML platform, Streaming datasets, and the Composer library to train a Stable Diffusion model from scratch for less than $50,000. Now, we do a deep dive into the technical details behind this speedup, demonstrating how we were able to replicate the Stable Diffusion 2 base model in just 6.8 days.

Try out our code here!

Many organizations require high-performing large AI models tailored to their specific use cases. However, training such models is often prohibitively time-consuming and expensive, requiring vast amounts of computation and expertise. This is where MosaicML comes in: we provide a comprehensive solution that simplifies and accelerates the process of training these models.

In our previous blog post, we announced that we have trained a diffusion model comparable to Stable Diffusion 2 from scratch for $47.7K. In this post, we dive into the technical details to highlight how we achieved an 8x speedup/cost reduction from the number reported by StabilityAI and a 3x cost reduction over our own baseline. All our code is open source and easy to modify for custom use cases. If you're interested in learning more about our stack, please contact us for a demo.

ACCELERATING TRAINING

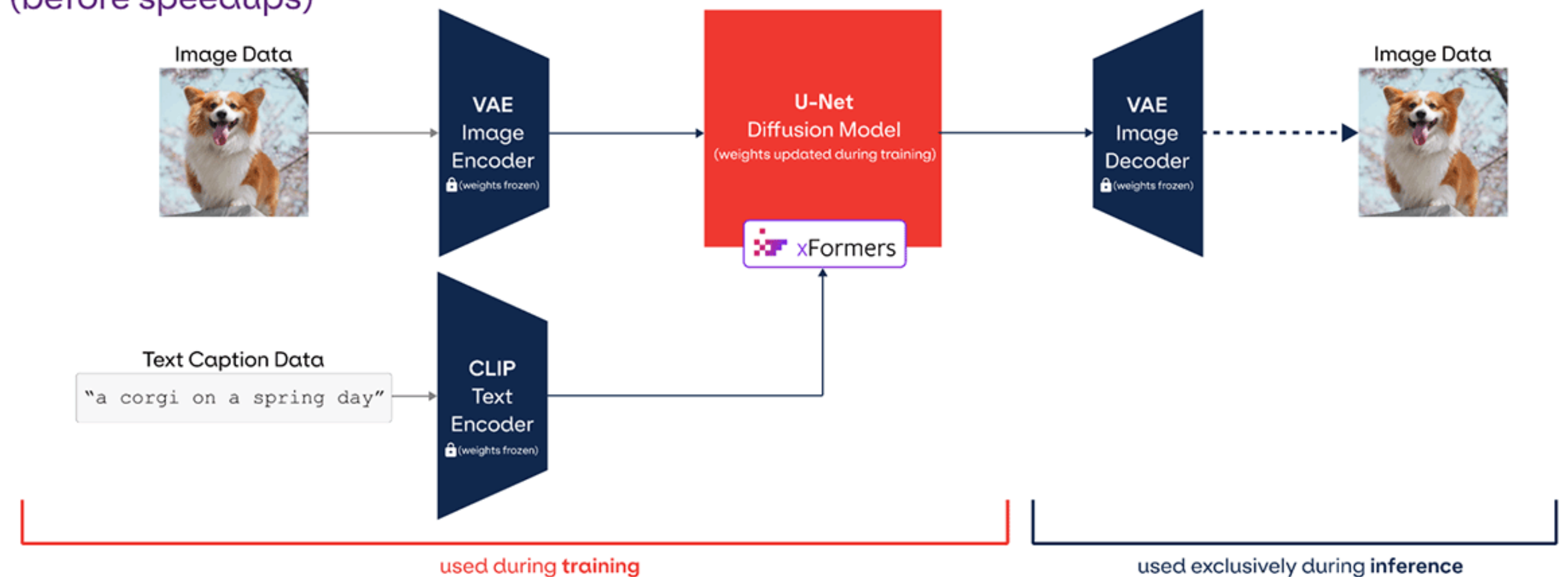
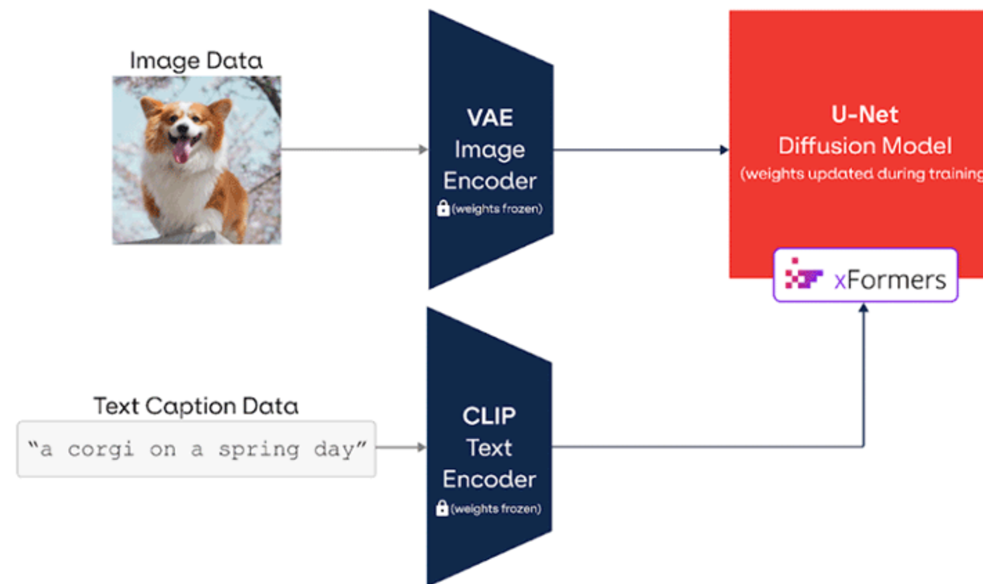


Figure 1: Stable Diffusion 2 model architecture. For training, the VAE image encoder, CLIP text encoder and U-Net are used. For inference, the CLIP Text Encoder, U-Net, and VAE image decoder are used. Only the U-Net weights are updated during training; CLIP and VAE are fixed.

We've introduced a variety of techniques, from fusions to sharding strategies, that dramatically speed up training and lower costs by almost 3x.

XFORMERS FLASHATTENTION

Diffusion Speedup #1: xFormers FlashAttention



Cumulative Throughput Boost Factor: 1.18x

Speedups:

- xFormers FlashAttention
- Precomputing Latents
- Low Precision LayerNorm & GroupNorm
- Fully Sharded Data Parallelism (FSDP)
- Scheduled Exponential Moving Average (EMA)

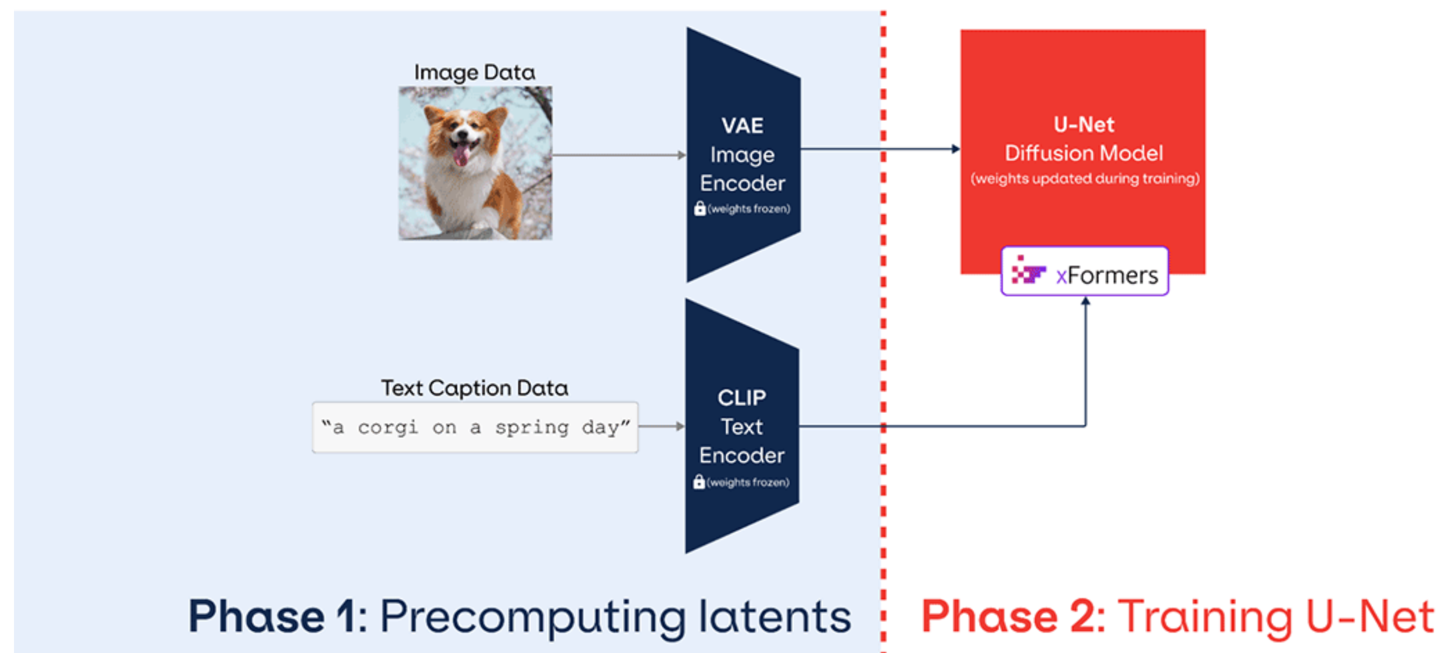
Figure 2: xFormers accelerates cross attention blocks in the U-Net.

The attention layers in the Stable Diffusion architecture can be slow with a naive implementation, so most codebases use faster implementations that rely on fused kernels. In our stack, we leverage xFormers FlashAttention.

While this was enabled in our original blog post, we found an issue with the usage that resulted in extra memory being consumed on rank 0. After fixing this bug, we were able to increase our device microbatch size1 from 4 to 8. This yielded a sizable speedup, since A100s are more efficient at larger matrix sizes.

PRECOMPUTING LATENTS

Diffusion Speedup #2: Precomputing Latents



Cumulative Throughput Boost Factor: 1.69x

Speedups:

- xFormers FlashAttention
- Precomputing Latents
- Low Precision LayerNorm & GroupNorm
- Fully Sharded Data Parallelism (FSDP)
- Scheduled Exponential Moving Average (EMA)

Figure 3: Two phase training with precomputed latents. First, all VAE and CLIP latents are precomputed and stored. Then, the U-Net diffusion model is trained using these precomputed latents.

Stable Diffusion is a combination of three models: a variational autoencoder (VAE), a text encoder (CLIP), and a U-Net. During diffusion training, only the U-Net is trained, and the other two models are used to compute the latent encodings of the image and text inputs. Standard training involves computing the VAE and CLIP latents for every batch, but this does a lot of duplicate work when training for multiple epochs: latents are re-computed for each image every time it is used. Instead, we precompute the latents once before training. Empirically, we have 2 epochs at 256 resolution and 5 epochs at 512 resolution, so we avoid 6 extra VAE and CLIP calls per image-text pair in the dataset.

Additionally, when pre-computing the latents, we can lower the precision of the VAE and CLIP models to fp16. This could lead to numerical instability if we were training the VAE and CLIP and used this precision for the backward pass. However, since we're only using them for inference, we can safely lower the precision, which increases speed. The extra memory savings also let us use far larger batch sizes and improve hardware utilization during the latent precomputation.

LOW PRECISION LAYERNORM AND GROUPNORM

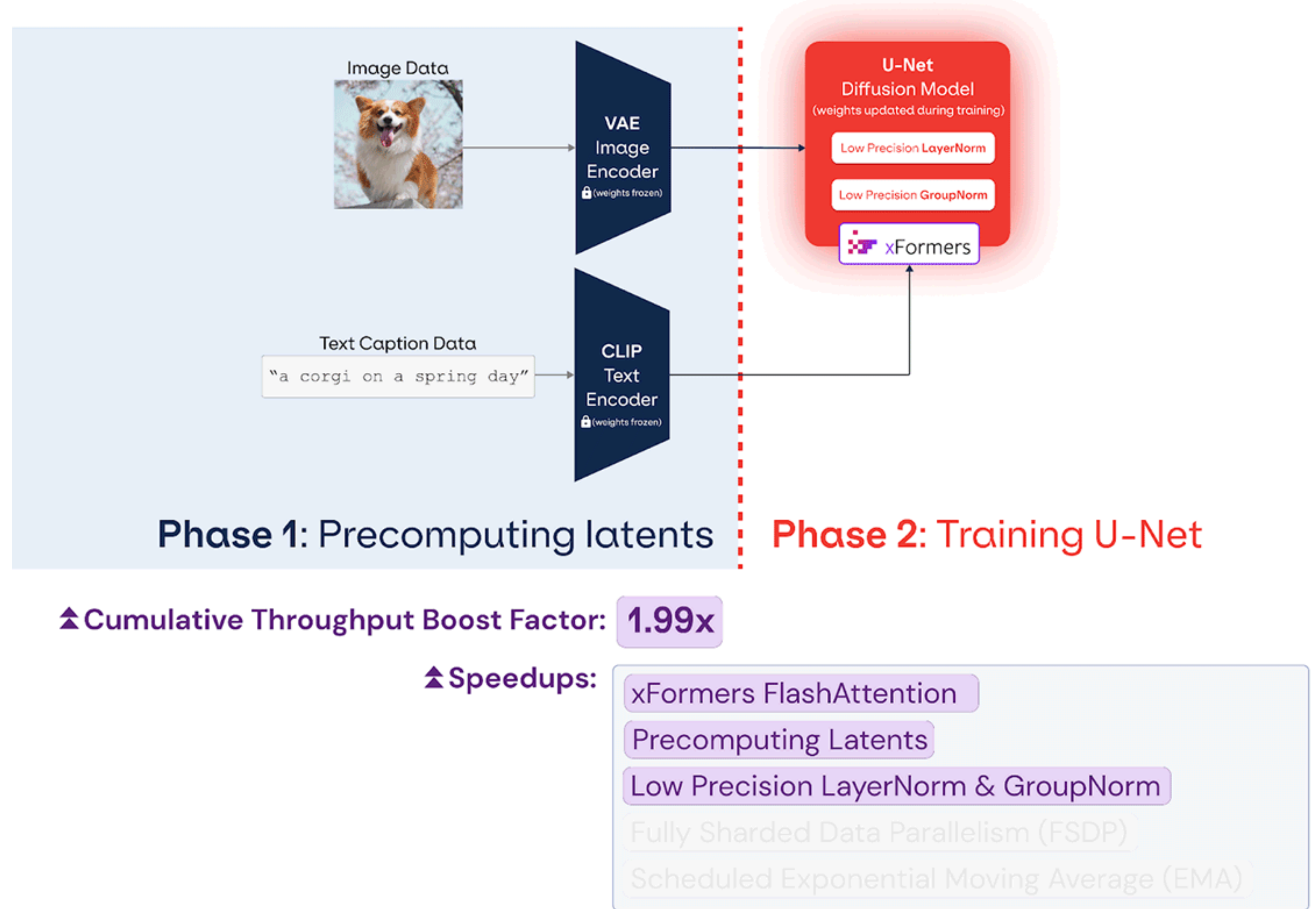


Figure 4: Low Precision LayerNorm and Low Precision GroupNorm. Low precision gives faster training and lower memory usage, enabling larger microbatches.

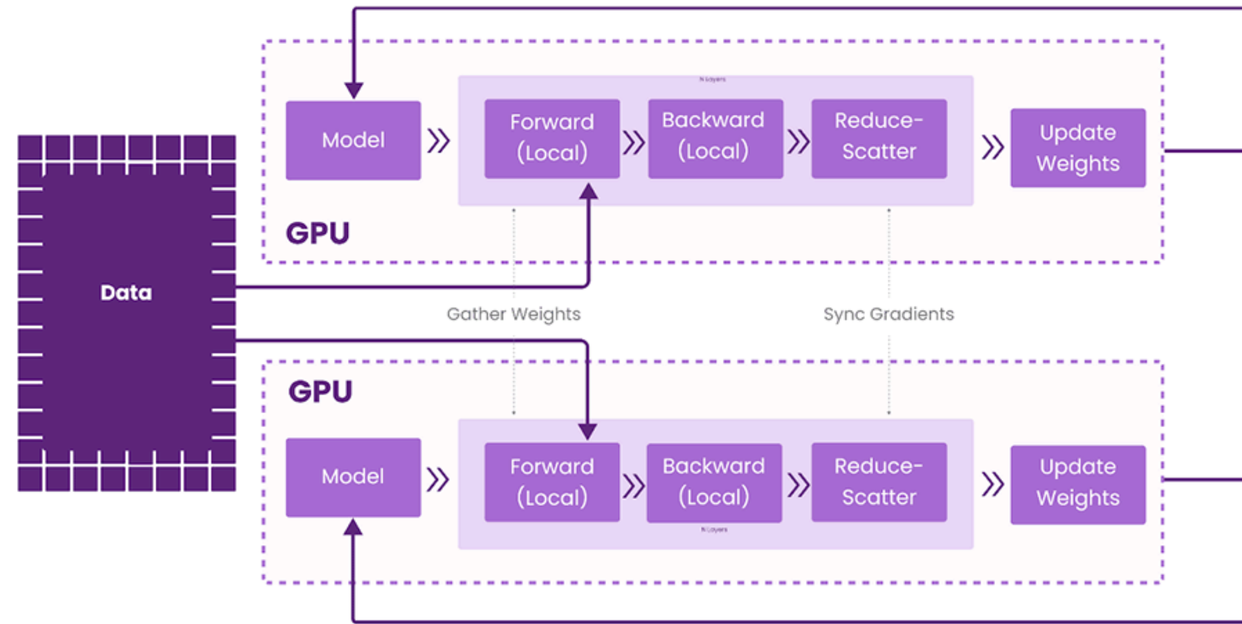
Diffusion training is done in automatic mixed precision by default. This uses half precision (fp16) in most layers, but fp32 in a few numerically unstable layers like normalization and softmax. The Stable Diffusion U-Net architecture uses several LayerNorm and GroupNorm layers, which by default are run in fp32.

Motivated by our finding that half precision LayerNorms are safe to use in language models, we decided to try out half precision LayerNorm and GroupNorm layers. This change resulted in identical loss curves and no instability in our experiments.

While we did observe some throughput improvement, the real benefit was decreased memory usage. Now, along with removing the VAE and CLIP memory by precomputing latents, we have enough space on our 40GB A100 to increase our microbatch size from 8 to 16, 4x larger than what we started with!

FULLY SHARDED DATA PARALLELISM

Diffusion Speedup #4: Fully Sharded Data Parallelism (FSDP)



Cumulative Throughput Boost Factor: 2.34x

Speedups:

- xFormers FlashAttention
- Precomputing Latents
- Low Precision LayerNorm & GroupNorm
- Fully Sharded Data Parallelism (FSDP)
- Scheduled Exponential Moving Average (EMA)

Figure 5: Fully Sharded Data Parallel with SHARD_GRAD_OP speeds up the gradient update step and enables linear scaling.

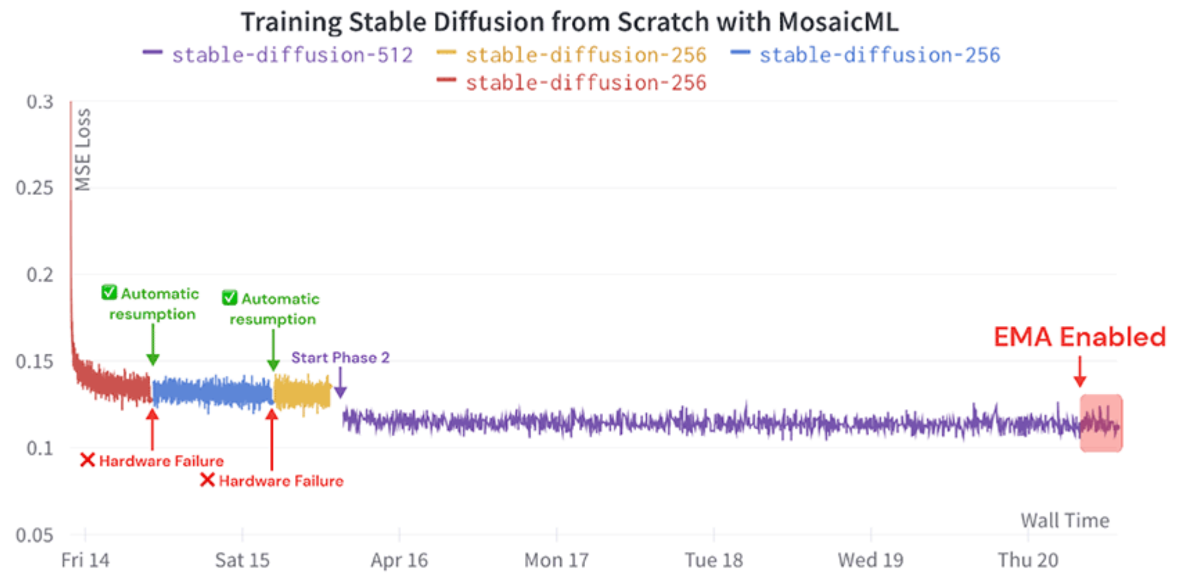
MosaicML Composer, our go-to training library, includes support for PyTorch Fully Sharded Data Parallelism (FSDP). We primarily use this to shard large scale models like 10B+ parameter LLMs that don't fit in a single device across hundreds of GPUs for incredibly fast training. Stable Diffusion doesn't require sharding since it fits in a single GPU. However, some of the distributed features in FSDP are still useful for speeding up training on a large number of GPUs.

When batches don't fit into memory, we do several forward and backward passes on smaller microbatches, followed by a single gradient update. If we use a small number of GPUs to train, we have far more forward and backward passes per gradient update, so the time spent on the gradient update doesn't matter. However, at 128+ GPUs with a microbatch size of 16, we're only doing one forward and one backward pass for each gradient update. At this scale, the gradient update step starts to become a significant bottleneck.

To tackle this problem, we use FSDP's SHARD_GRAD_OP mode. In normal training, each GPU communicates all its gradients to every other GPU, and then each GPU updates its local copy of the model. With this FSDP variant, each GPU only gets the gradients and updates the weights for a small part of the model before sending the updated weights for that part of the model to all of the other GPUs. By dividing the update step across all the GPUs, we can ensure the amount of work per GPU decreases as we increase the number of GPUs, helping us achieve linear scaling.

SCHEDULED EMA

Diffusion Speedup #5: Scheduled EMA



Cumulative Throughput Boost Factor: 2.71x

Speedups:

- xFormers FlashAttention
- Precomputing Latents
- Low Precision LayerNorm & GroupNorm
- Fully Sharded Data Parallelism (FSDP)
- Scheduled Exponential Moving Average (EMA)

Figure 6: Loss curve of our training run with the scheduled exponential moving average (EMA) period highlighted.

Stable Diffusion 2 uses Exponential Moving Averaging (EMA), which maintains an exponential moving average of the weights. At every time step, the EMA model is updated by taking 0.9999 times the current EMA model plus 0.0001 times the new weights after the latest forward and backward pass. By default, the EMA algorithm is applied after every gradient update for the entire training period. However, this can be slow due to the memory operations required to read and write all the weights at every step.

To avoid this costly procedure, we start with a key observation: since the old weights are decayed by a factor of 0.9999 at every batch, the early iterations of training only contribute minimally to the final average. This means we only need to take the exponential moving average of the final few steps. Concretely, we train for 1,400,000 batches and only apply EMA for the final 50,000 steps, which is about 3.5% of the training period. The weights from the first 1,350,000 iterations decay away by (0.9999)^50000, so their aggregate contribution would have a weight of less than 1% in the final model. Using this technique, we can avoid adding overhead for 96.5% of training and still achieve a nearly equivalent EMA model.

FINAL TIME AND COST ESTIMATES

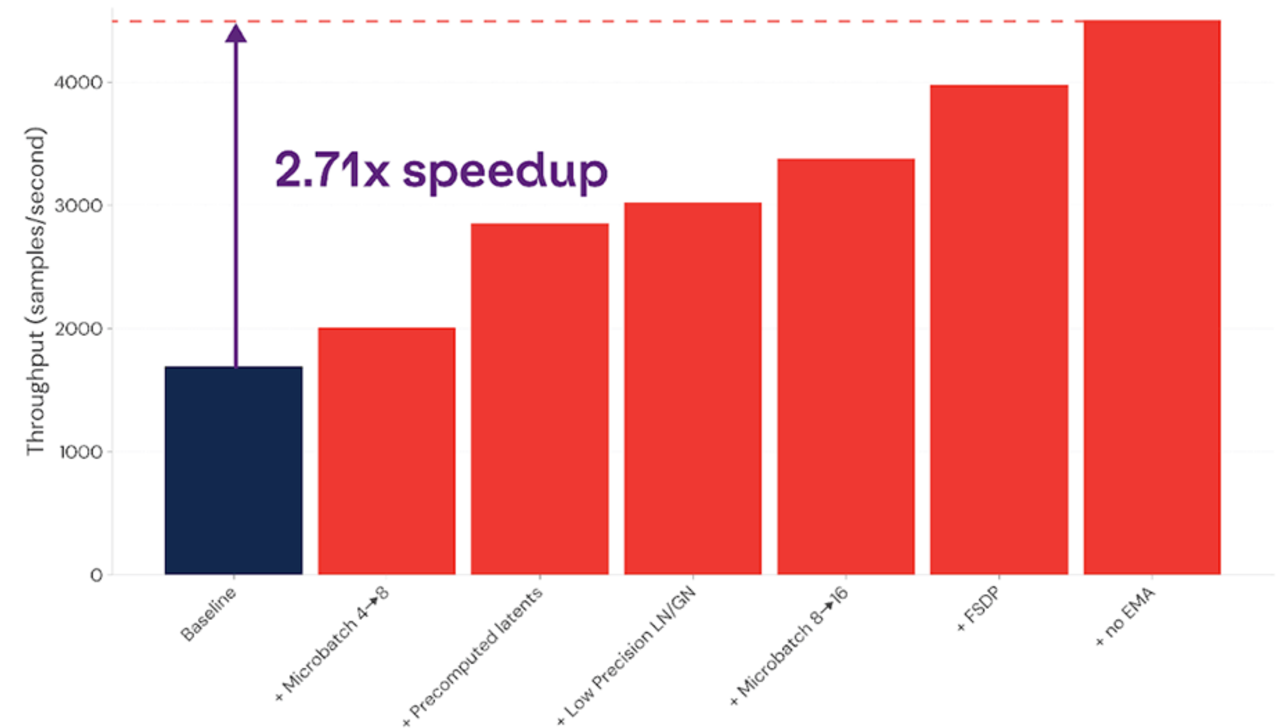


Figure 7: Throughput at 512x512 images on 128 GPUs as each speedup optimization is enabled. We achieve a total cumulative speedup of 2.71x over the baseline.

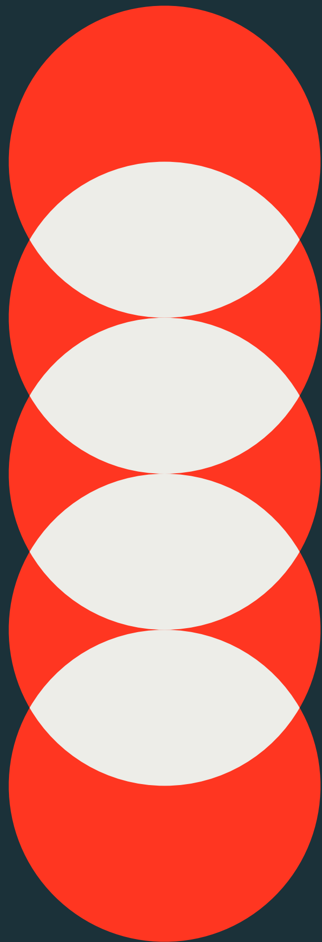
We've shown how we obtained nearly a 3x reduction in time and cost to train Stable Diffusion compared to our original results. With xFormers, precomputed latents, low precision LayerNorm, low precision GroupNorm, FSDP, and scheduled EMA, Table 1 shows it's possible to train Stable Diffusion in just 6.79 days using 21,000 A100-hours for a total cost of less than $42,000. We estimated these times and costs by measuring throughput for training 1.1 billion 256x256 images and 1.7 billion 512x512 images with a max tokenized length of 77 at a global batch size of 2048, as detailed in the Stable Diffusion 2 base model card. This is slightly cheaper than our previously reported run with a cost of $47.7k as it does not account for any time spent on evaluation or restarts due to hardware failures.

NUMBER OF A100S	THROUGHPUT FOR U-NET @ 256X256 (IMAGES / SECOND)	THROUGHPUT FOR U-NET @ 512X512 (IMAGES / SECOND)	THROUGHPUT FOR U-NET @ 512X512 WITH EMA (IMAGES / SECOND)	DAYS TO TRAIN ON MOSAICML CLOUD	APPROX. COST ON MOSAICML CLOUD
8	1100	290	290	101.04	$38,800
16	2180	585	580	50.29	$38,630
32	4080	1195	1160	25.01	$38,420
64	8530	2340	2220	12.63	$38,800
128	11600	4590	3927	6.79	$41,710

Table 1: Estimated time and cost to train a Stable Diffusion model on 1.1 billion images at 256x256 resolution, followed by 1.7 billion images at 512x512 resolution. Different rows show different numbers of NVIDIA 40GB A100 GPUs at a global batch size of 2048.

These optimizations show that training image generation models from scratch is within reach for everyone. For updates on our latest work, join our Community Slack or follow us on Twitter. If your organization wants to start training diffusion models today, please schedule a demo online or email us at demo@mosaicml.com.

1 When training large models with big batches that don't fit in memory in a single pass, each batch is divided into smaller microbatches. On each device, we can do a forward and backward pass for each microbatch and sum the gradients at the end to compute a gradient update equivalent to a single forward and backward pass with the entire batch all at once.



Stage 5: LLM Evaluation

Constant evaluation and monitoring of deployed large language models (LLMs) and generative AI applications are crucial due to the dynamic nature of both the data they interact with and the environments in which they operate. These systems learn from vast datasets and can evolve over time, potentially leading to shifts in performance, accuracy or even the emergence of biases. Continuous monitoring ensures that any deviation from expected behavior can be detected and corrected promptly, maintaining the integrity and reliability of the AI application. As user needs and societal norms change, ongoing evaluation allows these models to adapt, ensuring their outputs remain relevant, appropriate and effective. This vigilance not only mitigates risks associated with AI deployments, such as ethical concerns and regulatory compliance, but also maximizes the value and utility these technologies bring to organizations and end users.

Evaluating LLMs is a challenging and evolving domain, primarily because LLMs often demonstrate uneven capabilities across different tasks. An LLM might excel in one benchmark, but slight variations in the prompt or problem can drastically affect its performance. The dynamic nature of LLMs and their vast potential applications only amplify the challenge of establishing comprehensive evaluation standards.

databricks

Present challenges involved with evaluating LLM-powered applications include the following:

- **Variable performance:** LLMs can be sensitive to prompt variations, demonstrating high proficiency in one task but faltering with slight deviations in prompts.
- **Lack of ground truth:** Since most LLMs output natural language, it is very difficult to evaluate the outputs via traditional NLP metrics (BLEU, ROUGE, etc.). For example, suppose an LLM were used to summarize a news article. Two equally good summaries might have almost completely different words and word orders, so even defining a "ground truth" label becomes difficult or impossible.
- **Domain-specific evaluation:** For domain-specific fine-tuned LLMs, popular generic benchmarks may not capture their nuanced capabilities. Such models are tailored for specialized tasks, making traditional metrics less relevant. This divergence often necessitates the development of domain-specific benchmarks and evaluation criteria. See the example of Replit's code generation LLM.
- **Reliance on human judgment:** It is often the case that LLM performance is being evaluated in domains where text is scarce or there is a reliance on subject matter expert knowledge. In such scenarios, evaluating LLM output can be costly and time-consuming.

To help give examples of how this can be accomplished, here are two great examples of how you can monitor and evaluate your deployed LLMs and generative AI applications using Databricks.

LLM Evaluation Examples

Best Practices for LLM Evaluation of RAG Applications

A Case Study on the Databricks Documentation Bot

by Quinn Leng, Kasey Uhlenhuth and Alkis Polyzotis

Chatbots are the most widely adopted use case for leveraging the powerful chat and reasoning capabilities of large language models (LLM). The retrieval augmented generation (RAG) architecture is quickly becoming the industry standard for developing chatbots because it combines the benefits of a knowledge base (via a vector store) and generative models (e.g., GPT-3.5 and GPT-4) to reduce hallucinations, maintain up-to-date information, and leverage domain-specific knowledge. However, evaluating the quality of chatbot responses remains an unsolved problem today. With no industry standards defined, organizations resort to human grading (labeling) –which is time-consuming and hard to scale.

databricks

We applied theory to practice to help form best practices for LLM automated evaluation so you can deploy RAG applications to production quickly and with confidence. This blog represents the first in a series of investigations we're running at Databricks to provide learnings on LLM evaluation. All research in this post was conducted by Quinn Leng, Senior Software Engineer at Databricks and creator of the Databricks Documentation AI Assistant.

CHALLENGES WITH AUTO-EVALUATION IN PRACTICE

Recently, the LLM community has been exploring the use of "LLMs as a judge" for automated evaluation with many using powerful LLMs such as GPT-4 to do the evaluation for their LLM outputs. The lmsys group's research paper explores the feasibility and pros/cons of using various LLMs (GPT-4, ClaudeV1, GPT-3.5) as the judge for tasks in writing, math, and world knowledge.

Despite all this great research, there are still many unanswered questions about how to apply LLM judges in practice:

- **Alignment With Human Grading:** Specifically for a document-Q&A chatbot, how well does an LLM judge's grading reflect the actual human preference in terms of correctness, readability and comprehensiveness of the answers?
- **Accuracy Through Examples:** What's the effectiveness of providing a few grading examples to the LLM judge and how much does it increase the reliability and reusability of the LLM judge on different metrics?
- **Appropriate Grade Scales:** What grading scale is recommended because different grading scales are used by different frameworks (e.g., AzureML uses 0 to 100 whereas langchain uses binary scales)?
- **Applicability Across Use Cases:** With the same evaluation metric (e.g. correctness), to what extent can the evaluation metric be reused across different use cases (e.g. casual chat, content summarization, retrieval augmented generation)?

databricks

APPLYING EFFECTIVE AUTO-EVALUATION FOR RAG APPLICATIONS

We explored the possible options for the questions outlined above in the context of our own chatbot application at Databricks. We believe that our findings generalize and can thus help your team effectively evaluate RAG-based chatbots at a lower cost and faster speed:

- LLM-as-a-judge agrees with human grading on over 80% of judgments. Using LLMs-as-a-judge for our document-based chatbot evaluation was as effective as human judges, matching the exact score in over 80% of judgments and being within a 1-score distance (using a scale of 0-3) in over 95% of judgments.
- Save costs by using GPT-3.5 with examples. GPT-3.5 can be used as an LLM judge if you provide examples for each grading score. Because of the context size limit it's only practical to use a low-precision grading scale. Using GPT-3.5 with examples instead of GPT-4 drives down the cost of LLM judge by 10x and improves the speed by more than 3x.
- Use low-precision grading scales for easier interpretation. We found lower-precision grading scores like 0, 1, 2, 3 or even binary (0, 1) can largely retain precision compared to higher precision scales like 0 to 10.0 or 0 to 100.0, while making it considerably easier to provide grading rubrics to both human annotators and LLM judges. Using a lower precision scale also allows consistency of grading scales among different LLM judges (e.g., between GPT-4 and claude2).
- RAG applications require their own benchmarks. A model might have good performance on a published specialized benchmark (e.g. casual chat, math, or creative writing) but that doesn't guarantee good performance on other tasks (e.g. answering questions from a given context). Benchmarks should only be used if the use case matches, i.e., a RAG application should only be evaluated with a RAG benchmark.

Based on our research, we recommend the following procedure when using an LLM judge:

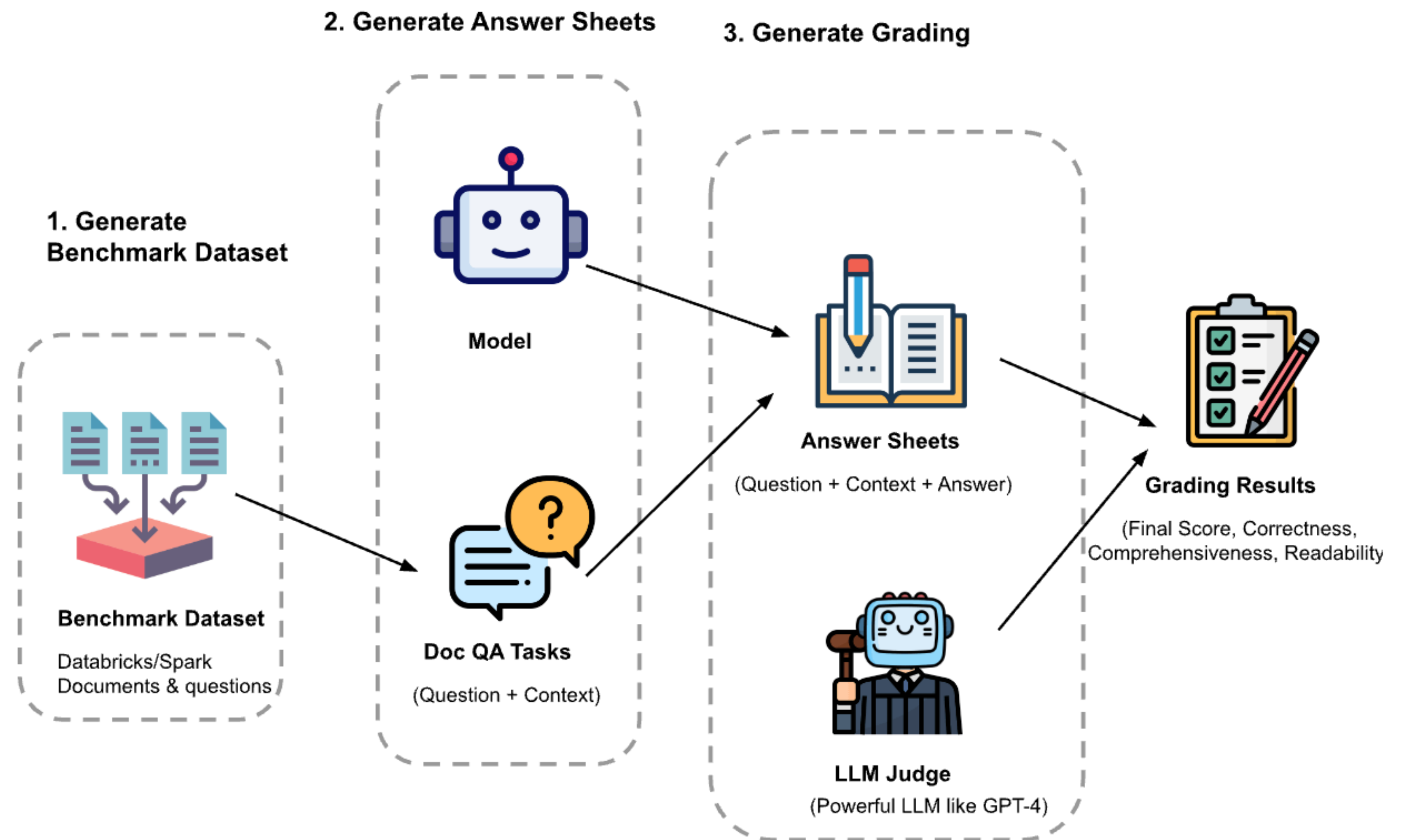
- Use a 1–5 grading scale
- Use GPT-4 as an LLM judge with no examples to understand grading rules
- Switch your LLM judge to GPT-3.5 with one example per score

databricks

OUR METHODOLOGY FOR ESTABLISHING BEST PRACTICES

The remainder of this post will walk through the series of experiments we conducted to form these best practices.

EXPERIMENT SETUP



The experiment had three steps:

1. **Generate evaluation dataset:** We created a dataset from 100 questions and context from Databricks documents. The context represents (chunks of) documents that are relevant to the question.

	question	context
0	What is DenseVector?	\n[c](DenseVector$.html "See companion object"...
1	What is the return value of `cube`?	`` \n![]()\n cube\n======\n`cube.Rd` \n Create a ... ``
4	7. What are the value members available for Co...	\n c\n \n[org](../../../../index.html) \n .\n ...
5	What are the value members of RuntimeInfo?	\n c\n \n[org](../../../../../index.html) \n ....
6	What is the name of the type used in JSON seri...	\n[c](ShortType$.html "See companion object")\...
...	...	...
85	What is the return type of RDDBarrier.mapParti...	\n pyspark.RDDBarrier.mapPartitions\n [¶](#pys...
86	Can you tell me how to construct SparkAWSCrede...	\n o\n \n[org](../../../../index.html) \n .\n ...
87	Can you tell me how to use PowerIterationClust...	\n PowerIterationClusteringModel\n [¶](#poweri...
88	What is the parameter numPartitions in RDD.dis...	\n pyspark.RDD.distinct\n [¶](#pyspark-rdd-dis...
89	How can the MultivariateGaussian class be used...	\n Source code for pyspark.mllib.stat.distribu...

2. **Generate answer sheets:** Using the evaluation dataset, we prompted different language models to generate answers and stored the question-context-answer pairs in a dataset called "answer sheets". In this investigation, we used GPT-4, GPT-3.5, Claude-v1, Llama2-70b-chat, Vicuna-33b, and mpt-30b-chat.
3. **Generate grades:** Given the answer sheets, we used various LLMs to generate grades and reasoning for the grades. The grades are a composite score of Correctness (weighted: 60%), Comprehensiveness (weighted: 20%) and Readability (weighted: 20%). We chose this weighting scheme to reflect our preference for Correctness in the generated answers. Other applications may tune these weights differently but we expect Correctness to remain a dominant factor.

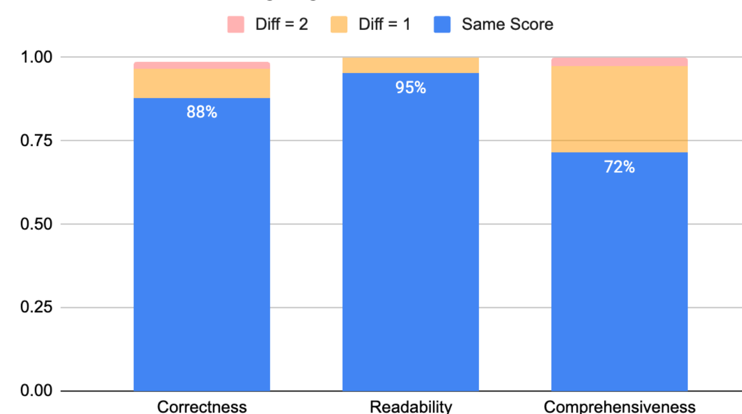
Additionally, the following techniques were used to avoid positional bias and improve reliability:

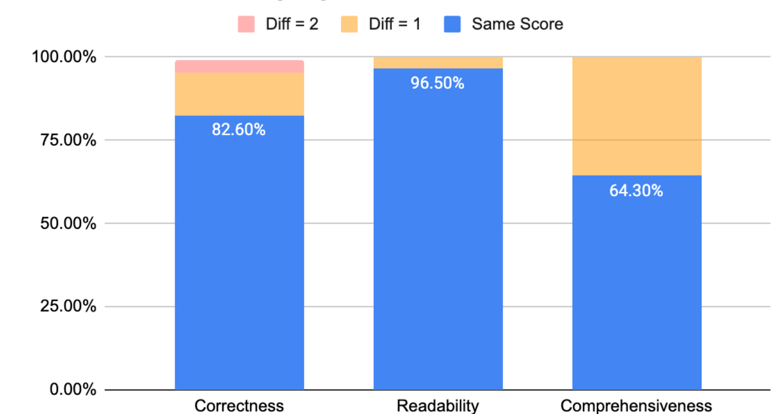
- Low temperature (temperature 0.1) to ensure reproducibility
- Single-answer grading instead of pairwise comparison
- Chain of thoughts to let the LLM reason about the grading process before giving the final score
- Few-shots generation where the LLM is provided with several examples in the grading rubric for each score value on each factor (Correctness, Comprehensiveness, Readability)

EXPERIMENT 1: ALIGNMENT WITH HUMAN GRADING

To confirm the level of agreement between human annotators and LLM judges, we sent answer sheets (grading scale 0–3) from gpt-3.5-turbo and vicuna-33b to a labeling company to collect human labels, and then compared the result with GPT-4's grading output. Below are the findings:

Human and GPT-4 judges can reach above 80% agreement on the correctness and readability score. And if we lower the requirement to be smaller or equal than 1 score difference, the agreement level can reach above 95%. The Comprehensiveness metric has less alignment, which matches what we've heard from business stakeholders who shared that "comprehensive" seems more subjective than metrics like Correctness or Readability.





EXPERIMENT 2: ACCURACY THROUGH EXAMPLES

The lmsys paper uses this prompt to instruct the LLM judge to evaluate based on the helpfulness, relevance, accuracy, depth, creativity, and level of detail of the response. However, the paper doesn't share specifics on the grading rubric. From our research, we found many factors can significantly affect the final score, for example:

- The importance of different factors: Helpfulness, Relevance, Accuracy, Depth, Creativity
- The interpretation of factors like Helpfulness is ambiguous
- If different factors conflict with each other, where an answer is helpful but is not accurate

We developed a rubric for instructing an LLM judge for a given grading scale, by trying the following:

1. **Original Prompt:** Here is the original prompt used in the lmsys paper:

```
Please act as an impartial judge and evaluate the quality of the response provided by an AI assistant to the user question
displayed below. Your evaluation should consider factors such as the helpfulness, relevance, accuracy, depth, creativity, and
level of detail of the response. Begin your evaluation by providing a short explanation. Be as objective as possible. After
providing your explanation, you must rate the response on a scale of 1 to 10 by strictly following this format
```

We adapted the original lmsys paper prompt to emit our metrics about correctness, comprehensiveness and readability, and also prompt the judge to provide one line justification before giving each score (to benefit from chain-of-thought reasoning). Below are the zero-shot version of the prompt which doesn't provide any example, and the few-shot version of the prompt which provides one example for each score. Then we used the same answer sheets as input and compared the graded results from the two prompt types.

2. **Zero Shot Learning:** Require the LLM judge to emit our metrics about correctness, comprehensiveness and readability, and also prompt the judge to provide one line justification for each score.

```
Please act as an impartial judge and evaluate the quality of the provided answer which attempts to answer the provided
question based on a provided context.
  You'll be given a function grading_function which you'll call for each provided context, question and answer to submit your
reasoning and score for the correctness, comprehensiveness and readability of the answer
```

3. **Few Shots Learning:** We adapted the zero shot prompt to provide explicit examples for each score in the scale. The new prompt:

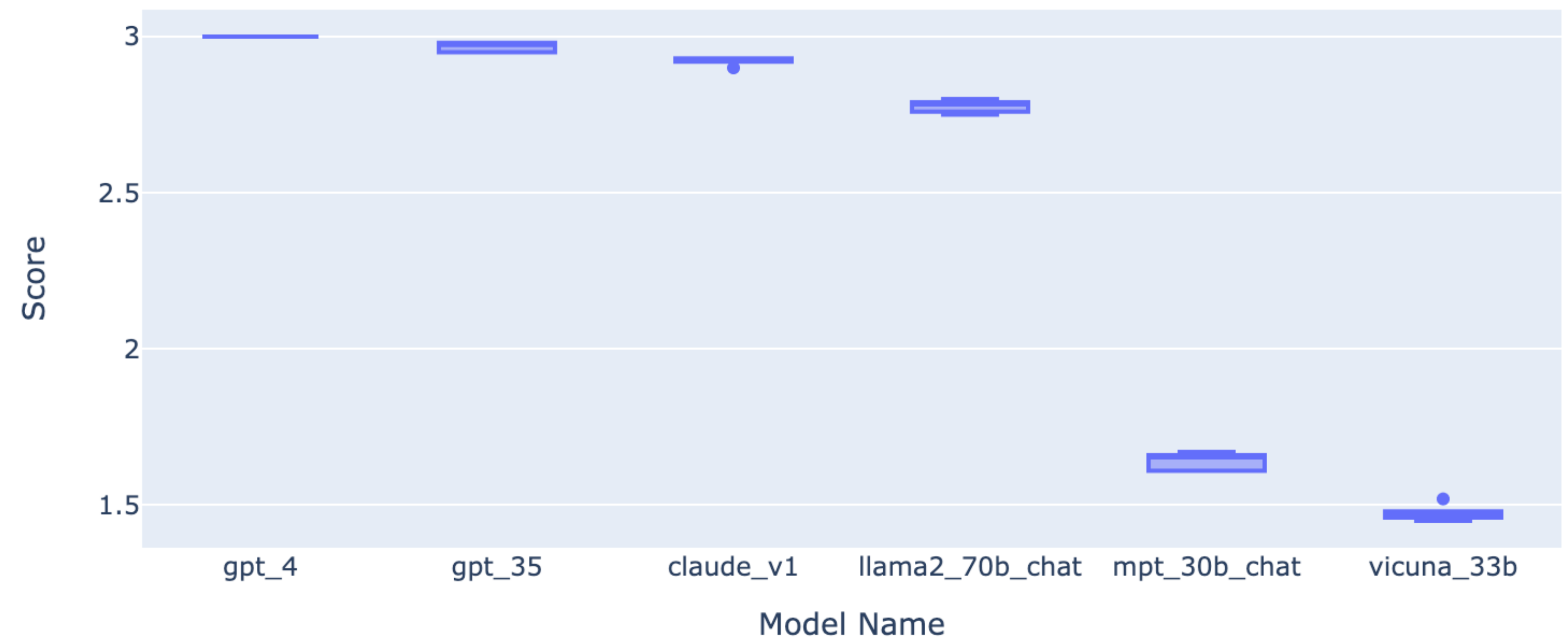
```
Please act as an impartial judge and evaluate the quality of the provided answer which attempts to answer the provided
question based on a provided context.
  You'll be given a function grading_function which you'll call for each provided context, question and answer to submit your
reasoning and score for the correctness, comprehensiveness and readability of the answer.

  Below is your grading rubric:
- Correctness: If the answer correctly answer the question, below are the details for different scores:
  - Score 0: the answer is completely incorrect, doesn't mention anything about the question or is completely contrary to the
correct answer.
      - For example, when asked "How to terminate a databricks cluster", the answer is empty string, or content that's
completely irrelevant, or sorry I don't know the answer.
  - Score 1: the answer provides some relevance to the question and answers one aspect of the question correctly.
      - Example:
          - Question: How to terminate a databricks cluster
          - Answer: Databricks cluster is a cloud-based computing environment that allows users to process big data and run
distributed data processing tasks efficiently.
          - Or answer:  In the Databricks workspace, navigate to the "Clusters" tab. And then this is a hard question that I
need to think more about it
  - Score 2: the answer mostly answer the question but is missing or hallucinating on one critical aspect.
      - Example:
          - Question: How to terminate a databricks cluster"
          - Answer: "In the Databricks workspace, navigate to the "Clusters" tab.
          Find the cluster you want to terminate from the list of active clusters.
          And then you'll find a button to terminate all clusters at once"
  - Score 3: the answer correctly answer the question and not missing any major aspect
      - Example:
          - Question: How to terminate a databricks cluster
          - Answer: In the Databricks workspace, navigate to the "Clusters" tab.
          Find the cluster you want to terminate from the list of active clusters.
          Click on the down-arrow next to the cluster name to open the cluster details.
          Click on the "Terminate" button. A confirmation dialog will appear. Click "Terminate" again to confirm the action."
- Comprehensiveness: How comprehensive is the answer, does it fully answer all aspects of the question and provide
comprehensive explanation and other necessary information. Below are the details for different scores:
  - Score 0: typically if the answer is completely incorrect, then the comprehensiveness is also zero score.
  - Score 1: if the answer is correct but too short to fully answer the question, then we can give score 1 for
comprehensiveness.
      - Example:
          - Question: How to use databricks API to create a cluster?
          - Answer: First, you will need a Databricks access token with the appropriate permissions. You can generate this
token through the Databricks UI under the 'User Settings' option. And then (the rest is missing)
  - Score 2: the answer is correct and roughly answer the main aspects of the question, but it's missing description about
details. Or is completely missing details about one minor aspect.
```

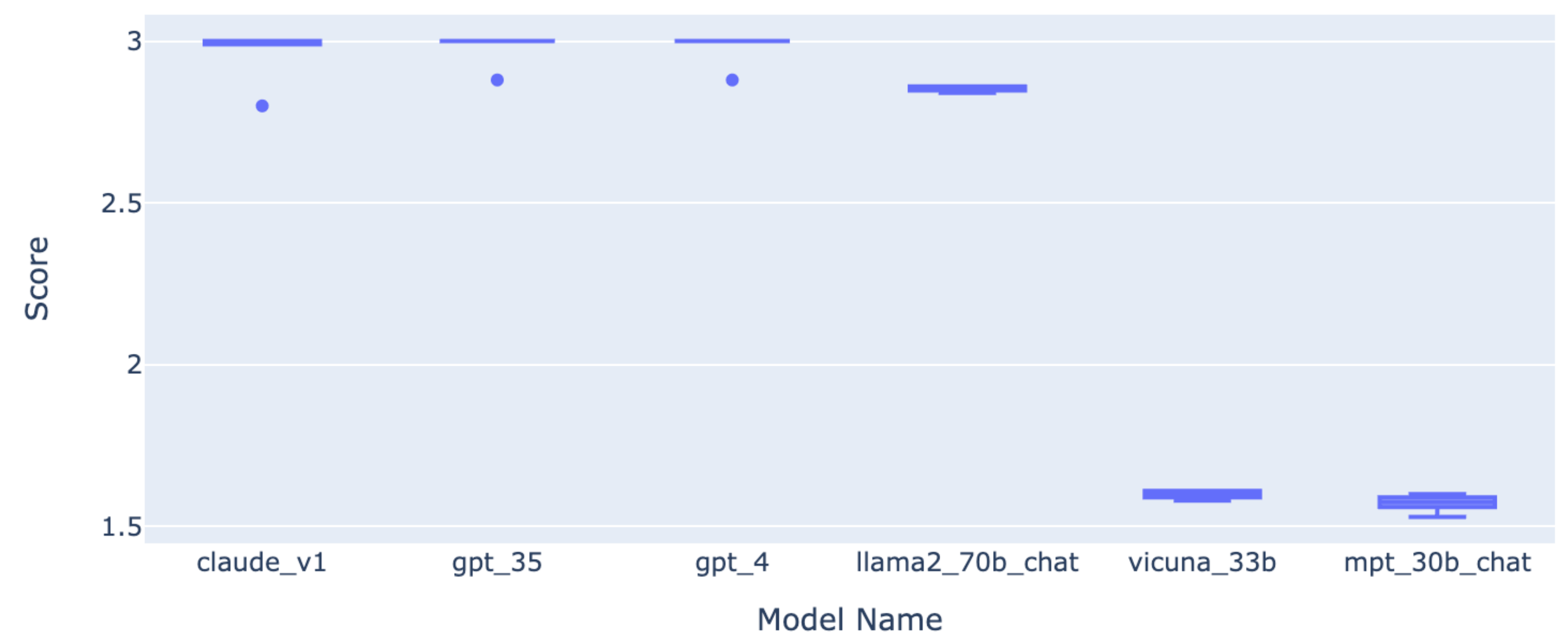
```
      - Example:
          - Question: How to use databricks API to create a cluster?
          - Answer: You will need a Databricks access token with the appropriate permissions. Then you'll need to set up the
request URL, then you can make the HTTP Request. Then you can handle the request response.
      - Example:
          - Question: How to use databricks API to create a cluster?
          - Answer: You will need a Databricks access token with the appropriate permissions. Then you'll need to set up the
request URL, then you can make the HTTP Request. Then you can handle the request response.
  - Score 3: the answer is correct, and covers all the main aspects of the question
- Readability: How readable is the answer, does it have redundant information or incomplete information that hurts the
readability of the answer.
  - Score 0: the answer is completely unreadable, e.g. fully of symbols that's hard to read; e.g. keeps repeating the words
that it's very hard to understand the meaning of the paragraph. No meaningful information can be extracted from the answer.
  - Score 1: the answer is slightly readable, there are irrelevant symbols or repeated words, but it can roughly form a
meaningful sentence that cover some aspects of the answer.
      - Example:
          - Question: How to use databricks API to create a cluster?
          - Answer: You you  you  you  you  you  will need a Databricks access token with the appropriate permissions. And
then then you'll need to set up the request URL, then you can make the HTTP Request. Then Then Then Then Then Then Then Then
Then
  - Score 2: the answer is correct and mostly readable, but there is one obvious piece that's affecting the readability
(mentioning of irrelevant pieces, repeated words)
      - Example:
          - Question: How to terminate a databricks cluster
          - Answer: In the Databricks workspace, navigate to the "Clusters" tab.
          Find the cluster you want to terminate from the list of active clusters.
          Click on the down-arrow next to the cluster name to open the cluster details.
          Click on the "Terminate" button…………………………………..
          A confirmation dialog will appear. Click "Terminate" again to confirm the action.
  - Score 3: the answer is correct and reader friendly, no obvious piece that affect readability.
- Then final rating:
    - Ratio: 60% correctness + 20% comprehensiveness + 20% readability
```

From this experiment, we learned several things:

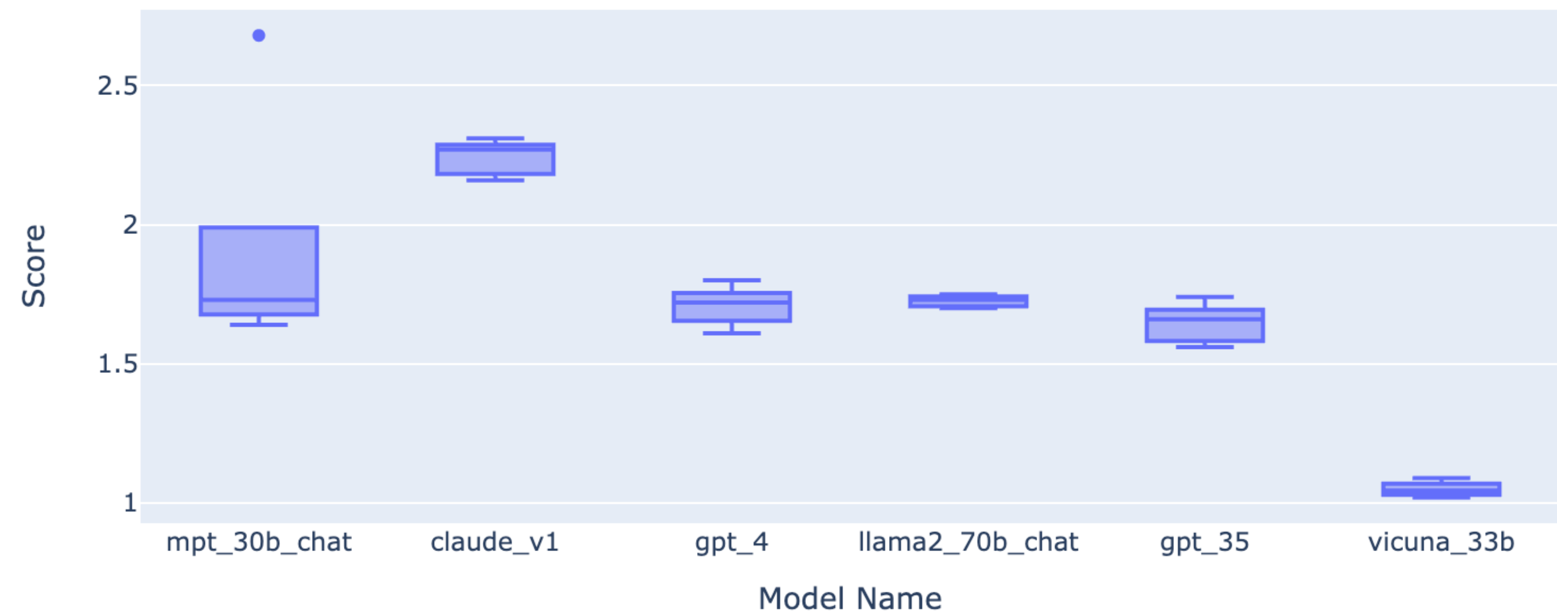
- Using the Few Shots prompt with GPT-4 didn't make an obvious difference in the consistency of results. When we included the detailed grading rubric with examples we didn't see a noticeable improvement in GPT-4's grading results across different LLM models. Interestingly, it caused a slight variance in the range of the scores.

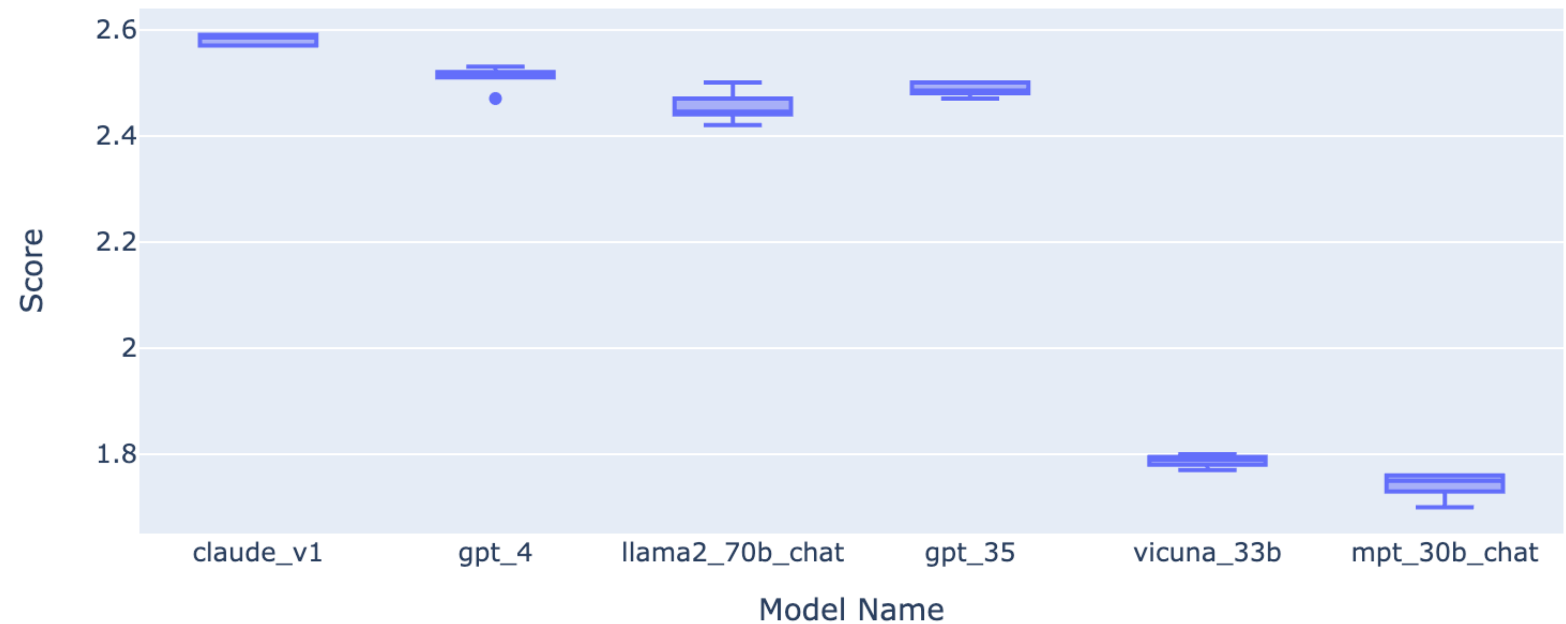
databricks


gpt-4 as Judge: LLM Score Variation: Zero shot
Score
3
2.5
2
1.5
gpt_4
gpt_35
claude_v1
llama2_70b_chat
mpt_30b_chat
vicuna_33b
Model Name


gpt-4 as Judge: LLM Score Variation: Few shot
Score
3
2.5
2
1.5
claude_v1
gpt_35
gpt_4
llama2_70b_chat
vicuna_33b
mpt_30b_chat
Model Name

- Including few examples for GPT-3.5-turbo-16k significantly improves the consistency of the scores, and makes the result usable. Including detailed grading rubric/examples has very obvious improvement on the grading result from GPT-3.5. Though the actual average score value is slightly different between GPT-4 and GPT-3.5 (score 3.0 vs score 2.6), the ranking and precision remains fairly consistent
- On the contrary, using GPT-3.5 without a grading rubric gets very inconsistent results and is completely unusable
- Note that we are using GPT-3.5-turbo-16k instead of GPT-3.5-turbo since the prompt can be larger than 4k tokens.





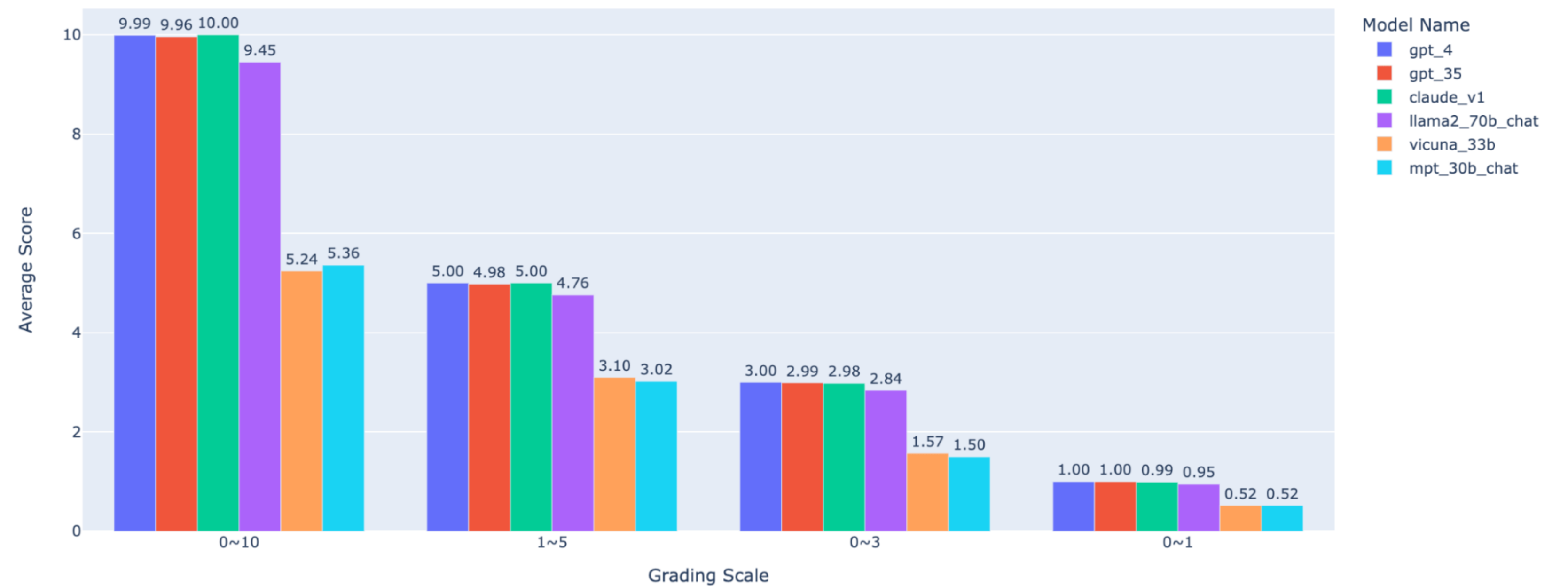
EXPERIMENT 3: APPROPRIATE GRADE SCALES

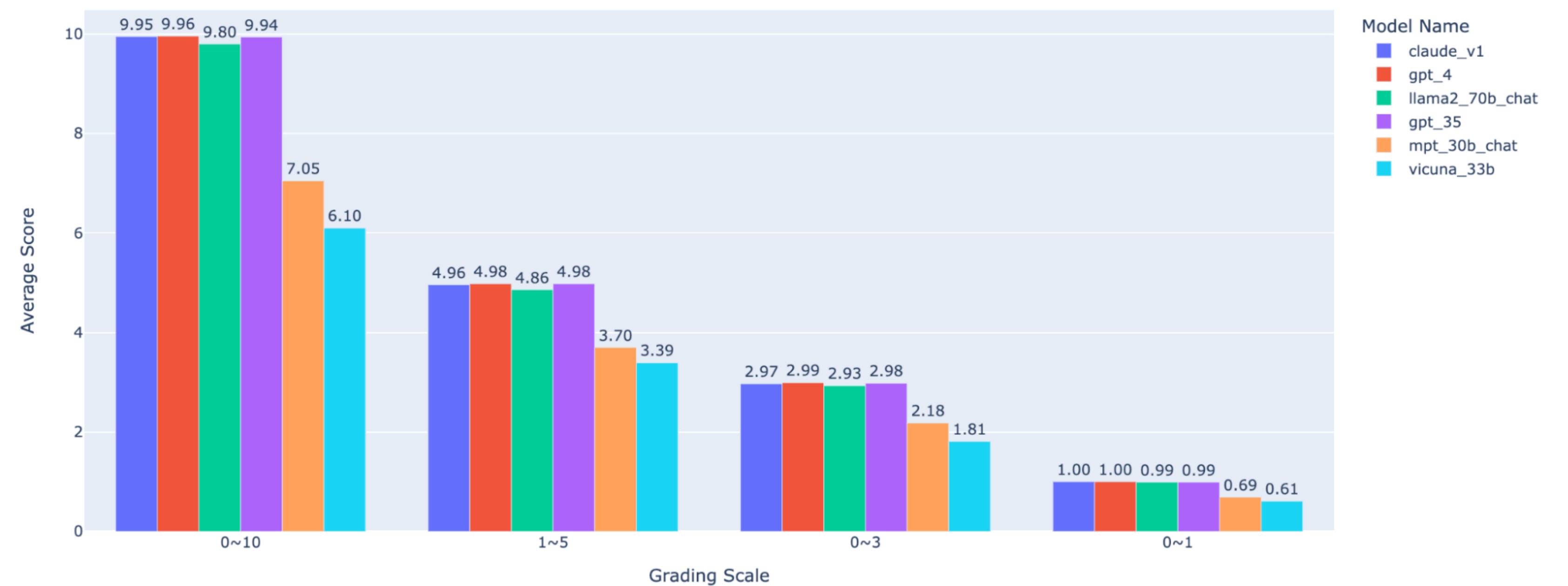
The LLM-as-judge paper uses a non-integer 0~10 scale (i.e., float) for the grading scale; in other words, it uses a high precision rubric for the final score. We found these high-precision scales cause issues downstream with the following:

- **Consistency:** Evaluators–both human and LLM–struggled to hold the same standard for the same score when grading on high precision. As a result, we found that output scores are less consistent across judges if you move from low-precision to high-precision scales.
- **Explainability:** Additionally, if we want to cross-validate the LLM-judged results with human-judged results we must provide instructions on how to grade answers. It is very difficult to provide accurate instructions for each "score" in a high-precision grading scale–for example, what's a good example for an answer that's scored at 5.1 as compared to 5.6?

We experimented with various low-precision grading scales to provide guidance on the "best" one to use, ultimately we recommend an integer scale of 0-3 or 0-4 (if you want to stick to the Likert scale). We tried 0-10, 1-5, 0-3, and 0-1 and learned:

- Binary grading works for simple metrics like "usability" or "good/bad".
- Scales like 0-10 are difficult to come up with distinguishing criteria between all scores.





As shown in these plots, both GPT-4 and GPT-3.5 can retain consistent ranking of results using different low-precision grading scales, thus using a lower grading scale like 0~3 or 1~5 can balance the precision with explainability).

Thus, we recommend 0-3 or 1-5 as a grading scale to make it easier to align with human labels, reason about scoring criteria, and provide examples for each score in the range.

EXPERIMENT 4: APPLICABILITY ACROSS USE CASES

The LLM-as-judge paper shows that both LLM and human judgment ranks the Vicuna-13B model as a close competitor to GPT-3.5:

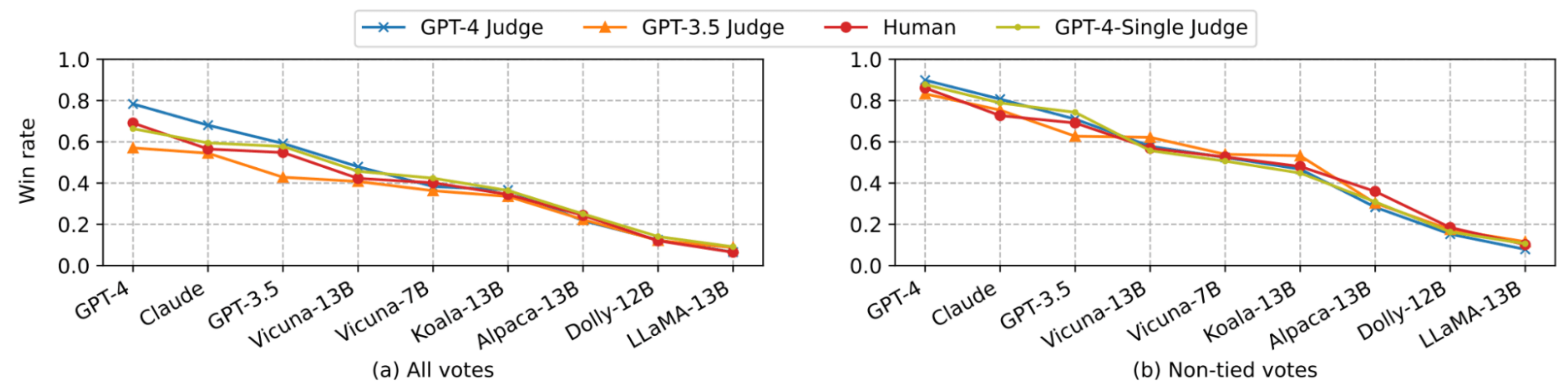
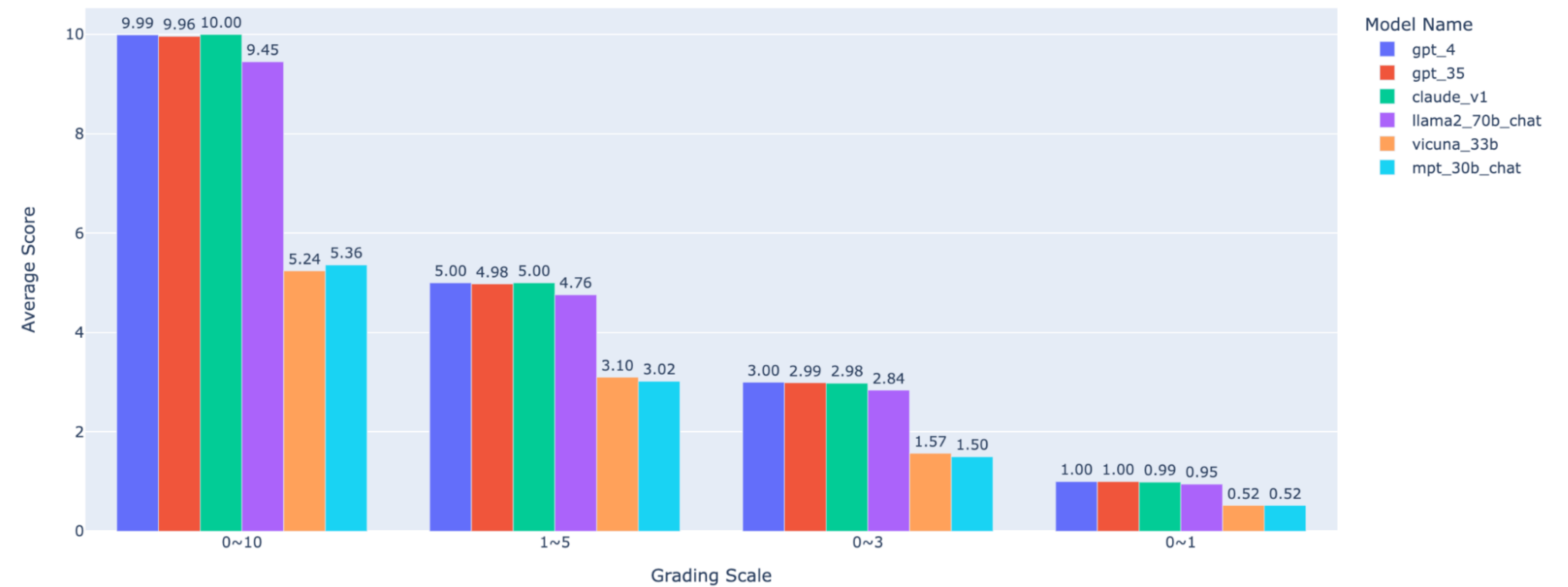


Figure 4: Average win rate of nine models under different judges on Chatbot Arena.

However, when we benchmarked the set of models for our document Q&A use cases, we found that even the much larger Vicuna-33B model has a noticeably worse performance than GPT-3.5 when answering questions based on context. These findings are also verified by GPT-4, GPT-3.5 and human judges (as mentioned in Experiment 1) which all agree that Vicuna-33B is performing worse than GPT-3.5.

databricks



We looked closer at the benchmark dataset proposed by the paper and found that the 3 categories of tasks (writing, math, knowledge) don't directly reflect or contribute to the model's ability to synthesize an answer based on a context. Instead, intuitively, document Q&A use cases need benchmarks on reading comprehension and instruction following. Thus evaluation results can't be transferred between use cases and we need to build use-case-specific benchmarks in order to properly evaluate how good a model can meet customer needs.

USE MLFLOW TO LEVERAGE OUR BEST PRACTICES

With the experiments above, we explored how different factors can significantly affect the evaluation of a chatbot and confirmed that LLM as a judge can largely reflect human preferences for the document Q&A use case. At Databricks, we are evolving the MLflow Evaluation API to help your team effectively evaluate your LLM applications based on these findings. MLflow 2.4 introduced the Evaluation API for LLMs to compare various models' text output side-by-side, MLflow 2.6 introduced LLM-based metrics for evaluation like toxicity and perplexity, and we're working to support LLM-as-a-judge in the near future!

In the meantime, we compiled the list of resources we referenced in our research below:

- Doc_qa repository
  - The code and data we used to conduct the experiments
- LLM-as-Judge Research paper from lmsys group
  - The paper is the first research for using LLM as judge for the casual chat use cases, it extensively explored the feasibility and pros and cons of using LLM (GPT-4, ClaudeV1, GPT-3.5) as the judge for tasks in writing, math, world knowledge

Offline LLM Evaluation: Step-by-Step GenAI Application Assessment on Databricks

by Abe Omorogbe, Liang Zhang, Sunish Sheth, Corey Zumar, Maheswaran Venkatachalam, Emil Lysgaard and Mathias Christiansen

BACKGROUND

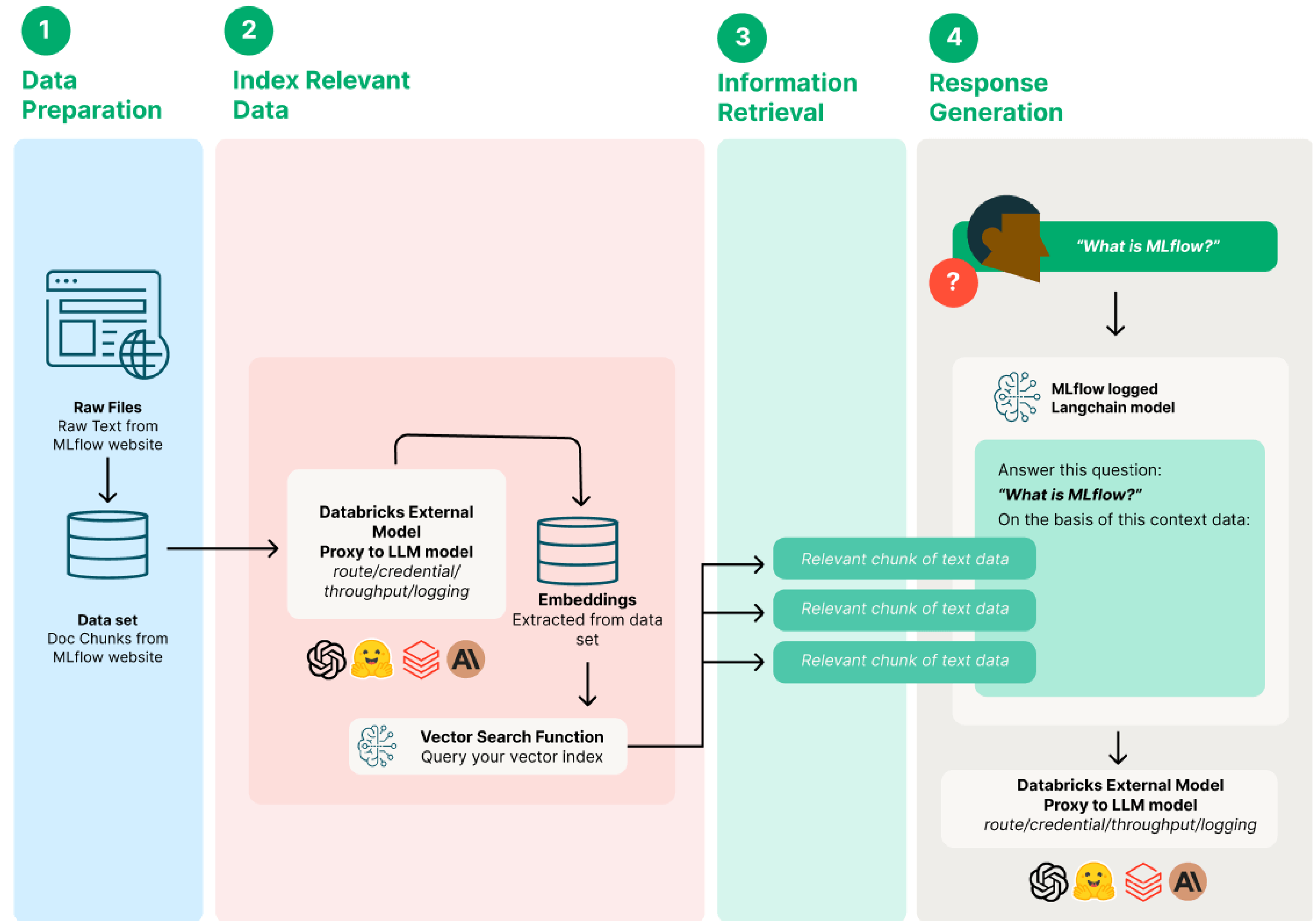
In an era where retrieval augmented generation (RAG) is revolutionizing the way we interact with AI-driven applications, ensuring the efficiency and effectiveness of these systems has never been more essential. Databricks and MLflow are at the forefront of this innovation, offering streamlined solutions for the critical evaluation of GenAI applications.

This blog post guides you through the simple and effective process of leveraging the Databricks Data Intelligence Platform to enhance and evaluate the quality of the three core components of your GenAI applications: Prompts, Retrieval System, and Foundation LLM, ensuring that your GenAI applications continue to generate accurate results.

databricks

USE CASE

We are going to be creating a QA chatbot that will answer questions from the MLflow documentation and then evaluate the results.



SET UP EXTERNAL MODELS IN DATABRICKS

Databricks Model Serving feature can be used to manage, govern, and access external models from various large language model (LLM) providers, such as Azure OpenAI GPT, Anthropic Claude, or AWS Bedrock, within an organization. It offers a high-level interface that simplifies the interaction with these services by providing a unified endpoint to handle specific LLM related requests.

Major advantages of using Model Serving:

- **Query Models Through a Unified Interface:** Simplifies the interface to call multiple LLMs in your organization. Query models through a unified OpenAI-compatible API and SDK and manage all models through a single UI.
- **Govern and Manage Models:** Centralizes endpoint management of multiple LLMs in your organization. This includes the ability to manage permissions and track usage limits.
- **Central Key Management:** Centralizes API key management in a secure location, which enhances organizational security by minimizing key exposure in the system and code, and reduces the burden on end-users.

databricks

CREATE A SERVING ENDPOINT WITH AN EXTERNAL MODEL IN DATABRICKS

```
import mlflow
import mlflow.deployments

client = mlflow.deployments.get_deploy_client("databricks")

endpoint_name = f"test-endpoint-{uuid.uuid4()}"

client.create_endpoint(
name=endpoint_name,
config={
        "served_entities": [
            {
                "name": "test",
                "external_model": {
                    "name": "gpt-3.5-turbo-instruct",
                    "provider": "openai",
                    "task": "llm/v1/completions",
                    "openai_config": {
                        "openai_api_type": "azure",
                        "openai_api_key": "{{secrets/<your-scope-name>/<your-key-name>}}", ## Use Databricks Secrets.
                        "openai_api_base": "https://<your-endpoint>.openai.azure.com/",
                        "openai_deployment_name": "<your-deployment-name>",
                        "openai_api_version": "2023-05-15",
                    },
                },
            }
        ],
       },
)
```

EXPLORE PROMPTS WITH THE DATABRICKS AI PLAYGROUND

In this section, we will understand: How well do different prompts perform with the chosen LLM?

We recently introduced the Databricks AI Playground, which provides a best-in-class experience for crafting the perfect prompt. With no code required, you can try out multiple LLMs served as Endpoints in Databricks, and test different parameters and prompts.

Major advantages of the Databricks AI Playground are:

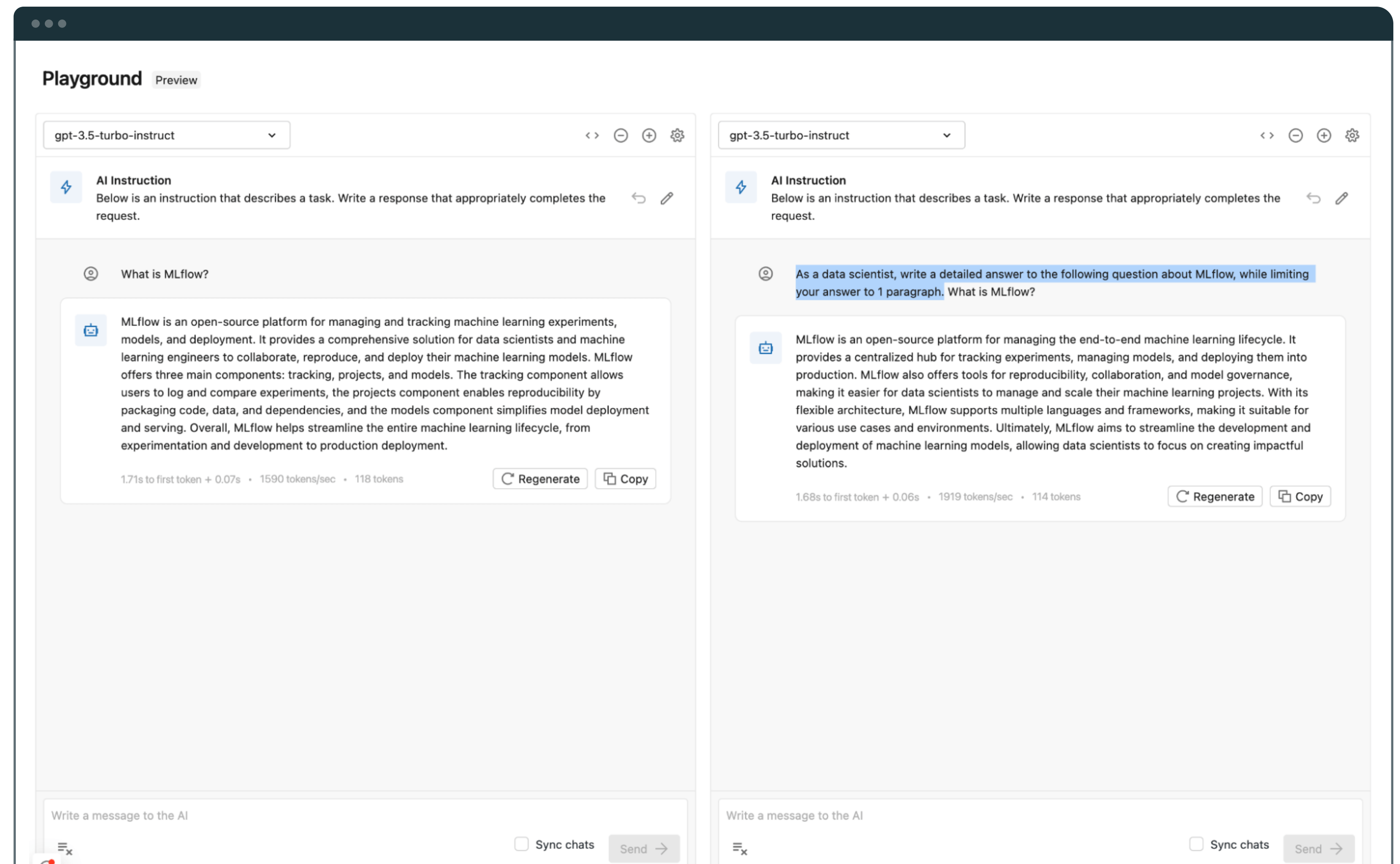
- **Quick Testing:** Quickly test deployed models directly in Databricks.
- **Easy Comparison:** Central location to compare multiple models on different prompts and parameters for comparison and selection.

USING DATABRICKS AI PLAYGROUND

We delve into testing relevant prompts with OpenAI GPT 3.5 Turbo, leveraging the Databricks AI Playground.

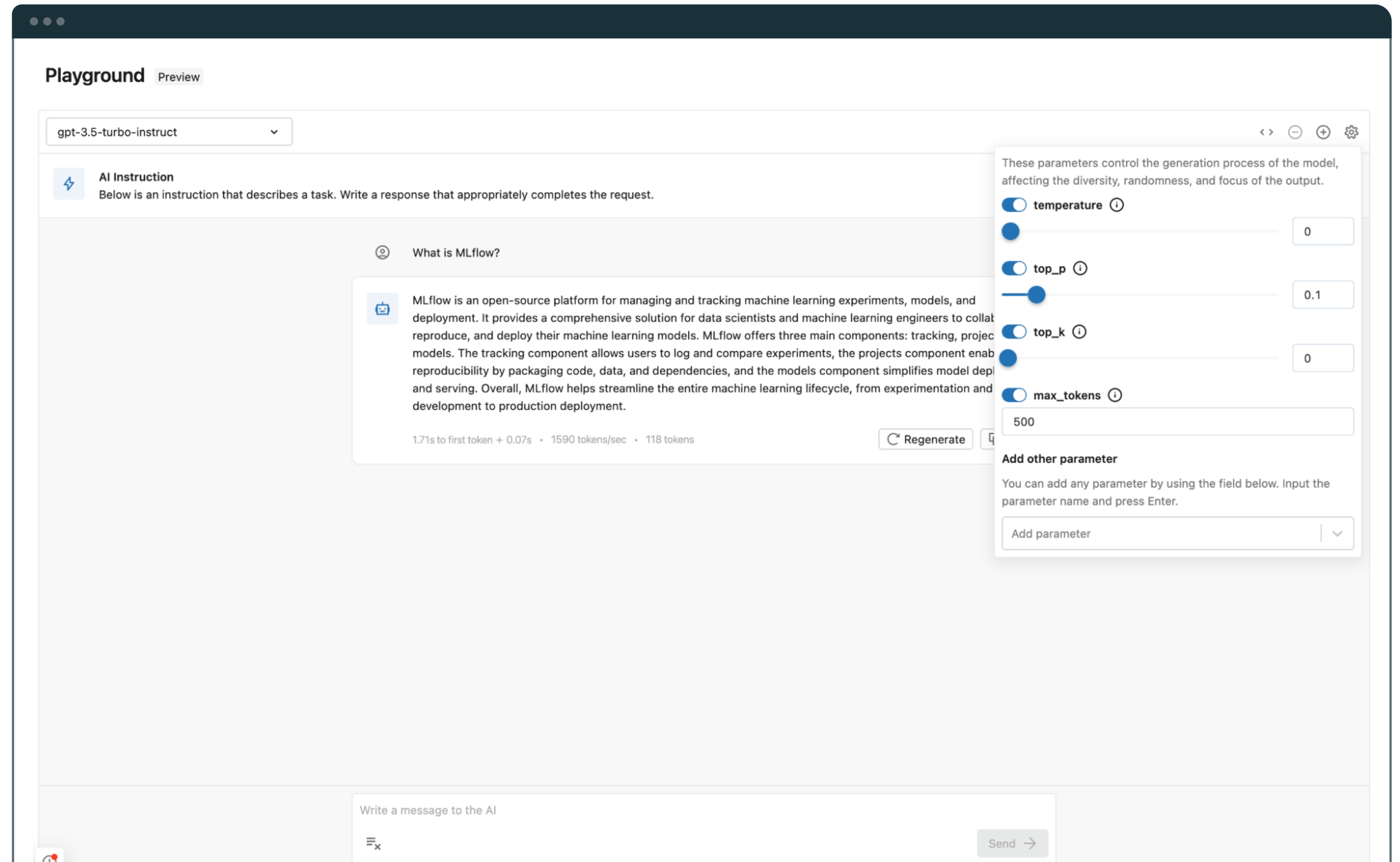
COMPARING DIFFERENT PROMPTS AND PARAMETERS

In the Playground, you are able to compare the output of multiple prompts to see which gives better results. Directly in the Playground, you can try several prompts, models, and parameters to figure out which combination provides the best results. The model and parameters combo can then be added to the GenAI app and used for answer generation with the right context.



ADDING MODEL AND PARAMETERS TO YOUR GENAI APPLICATION

After playing with a few prompts and parameters, you can use the same settings and model in your GenAI application.



Example of how to import the same external model in LangChain. We will cover how we turn this into a GenAI POC in the next section.

```
from langchain.llms import Databricks


llm = Databricks(
    endpoint_name="<endpoint-name>",
    extra_params={"temperature": 0.1,
                  "top_p": 0.1,
                  "max_tokens": 500,
                  } #parameters used in AI Playground
)
```

CREATE GENAI POC WITH LANGCHAIN AND LOG WITH MLFLOW

Now that we have found a good model and prompt parameters for your use case, we are going to create a sample GenAI app that is a QA chatbot that will answer questions from the MLflow documentation using a vector database, embedding model with the Databricks Foundation Model API and Azure OpenAI GPT 3.5 as the generation model.

CREATE A SAMPLE GENAI APP WITH LANGCHAIN USING DOCS FROM THE MLFLOW WEBSITE

```
import os
import pandas as pd
import mlflow
import chromadb
from langchain.chains import RetrievalQA
from langchain.document_loaders import WebBaseLoader
from langchain.llms import Databricks
from langchain.embeddings.databricks import DatabricksEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.embeddings.sentence_transformer import SentenceTransformerEmbeddings

loader = WebBaseLoader(
    [
     "https://mlflow.org/docs/latest/index.html",
     "https://mlflow.org/docs/latest/tracking/autolog.html",
     "https://mlflow.org/docs/latest/getting-started/tracking-server-overview/index.html",
     "https://mlflow.org/docs/latest/python_api/mlflow.deployments.html" ])

documents = loader.load()
CHUNK_SIZE = 1000
text_splitter = CharacterTextSplitter(chunk_size=CHUNK_SIZE, chunk_overlap=0)
texts = text_splitter.split_documents(documents)

llm = Databricks(
    endpoint_name="<endpoint-name>",
    extra_params={"temperature": 0.1,
                  "top_p": 0.1,
                  "max_tokens": 500,
                  } #parameters used in AI Playground
)

# create the embedding function using Databricks Foundation Model APIs
embedding_function = DatabricksEmbeddings(endpoint="databricks-bge-large-en")
docsearch = Chroma.from_documents(texts, embedding_function)

qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=docsearch.as_retriever(fetch_k=3),
    return_source_documents=True,
)
```

databricks

For customers wanting to scale the retriever used in their GenAI application, we advise using Databricks Vector Search, a serverless similarity search engine that allows you to store a vector representation of your data, including metadata, in a vector database.

EVALUATION OF RETRIEVAL SYSTEM WITH MLFLOW

In this section, we will understand: How well does the retriever work with a given query?

In MLflow 2.9.1, Evaluation for retrievers was introduced and provides a way for you to assess the efficiency of their retriever with the MLflow evaluate API. You can use this API to evaluate the effectiveness of your embedding model, the top K threshold choice, or the chunking strategy.

CREATING A GROUND TRUTH DATASET

Curating a ground truth dataset for evaluating your GenAI often involves the meticulous task of manually annotating test sets, a process that demands both time and domain expertise. In this blog, we're taking a different route. We're leveraging the power of an LLM to generate synthetic data for testing, offering a quick-start approach to get a sense of your GenAI app's retrieval capability, and a warm-up for all the in-depth evaluation work that may follow. To our readers and customers, we emphasize the importance of crafting a dataset that mirrors the expected inputs and outputs of your GenAI application. It's a journey worth taking for the incredible insights you'll gain!

You can explore with the full dataset but let's demo with a subset of the generated data. The question column contains all the questions that will be evaluated and the source column is the expected source for the answer for the questions as an ordered list of strings.

databricks

```
eval_data = pd.DataFrame(
    {
        "question": [
            "What is MLflow?",
            "What is Databricks?",
            "How to serve a model on Databricks?",
            "How to enable MLflow Autologging for my workspace by default?",
        ],
        "source": [
            ["https://mlflow.org/docs/latest/index.html"],
            ["https://mlflow.org/docs/latest/getting-started/tracking-server-overview/index.html"],
            ["https://mlflow.org/docs/latest/python_api/mlflow.deployments.html"],
            ["https://mlflow.org/docs/latest/tracking/autolog.html"],
        ],
    }
)
```

EVALUATE THE EMBEDDING MODEL WITH MLFLOW

The quality of your embedding model is pivotal for accurate retrieval. In MLflow 2.9.0, we introduced three built-in metrics mlflow.metrics.precision_at_k(k), mlflow.metrics.recall_at_k(k) and mlflow.metrics.ndcg_at_k(k) to help determine how effective your retriever is at predicting the most relevant results for you. For example; Suppose the vector database returns 10 results (k=10), and out of these 10 results, 4 are relevant to your query. The precision_at_10 would be 4/10 or 40%.

```
def evaluate_embedding(embedding_function):
    CHUNK_SIZE = 1000
    list_of_documents = loader.load()
    text_splitter = CharacterTextSplitter(chunk_size=CHUNK_SIZE, chunk_overlap=0)
    docs = text_splitter.split_documents(list_of_documents)
    retriever = Chroma.from_documents(docs, embedding_function).as_retriever()


    def retrieve_doc_ids(question: str) -> List[str]:
        docs = retriever.get_relevant_documents(question)
        doc_ids = [doc.metadata["source"] for doc in docs]
        return doc_ids


    def retriever_model_function(question_df: pd.DataFrame) -> pd.Series:
        return question_df["question"].apply(retrieve_doc_ids)


    with mlflow.start_run() as run:
        evaluate_results = mlflow.evaluate(
                model=retriever_model_function,
                data=eval_data,
                model_type="retriever",
                targets="source",
                evaluators="default",
            )
    return evaluate_results


result1 = evaluate_embedding(DatabricksEmbeddings(endpoint="databricks-bge-large-en"))result2 = evaluate_embed-
ding(<another-embedding-function>)


eval_results_of_retriever_df_bge = result1.tables["eval_results_table"]
display(eval_results_of_retriever_df_bge)
```

The evaluation will return a table with the results of your evaluation for each question. i.e., for this test, we can see that the retriever seems to performing great for the questions "How to enable MLflow Autologging for my workspace by default?" with a Precision @ K score is 1, and is not retrieving any of the right documentation for the questions "What is MLflow?" since the precision @ K score is 0. With this insight, we can debug the retriever and improve the retriever for questions like "What is MLflow?"

question	precision_at_1/score	precision_at_2/score	precision_at_3/score	source	outputs
What is MLflow?	0	0	0.00	["https://mlflow.org/docs/latest/index...	["https://mlflow.org/docs/latest/pyt
What is Databricks?	1	1	0.67	["https://mlflow.org/docs/latest/gettin...	["https://mlflow.org/docs/latest/get
How to serve a model on Databricks?	0	0	0.33	["https://mlflow.org/docs/latest/pytho...	["https://mlflow.org/docs/latest/get
How to enable MLflow Autologging for ...	1	1	1.00	["https://mlflow.org/docs/latest/tracki...	["https://mlflow.org/docs/latest/trac

Evaluation results when using databricks-bge-large-en embedding model

EVALUATE RETRIEVER WITH DIFFERENT TOP K VALUES WITH MLFLOW

You can quickly calculate the metrics for different Ks by specifying the extra_metrics argument.

```
with mlflow.start_run() as run:
        evaluate_results = mlflow.evaluate(
        data=eval_results_of_retriever_df_bge,
        targets="source",
        predictions="outputs",
        evaluators="default",
        extra_metrics=[
            mlflow.metrics.precision_at_k(1),
            mlflow.metrics.precision_at_k(2),
            mlflow.metrics.precision_at_k(3),
            mlflow.metrics.recall_at_k(1),
            mlflow.metrics.recall_at_k(2),
            mlflow.metrics.recall_at_k(3),
            mlflow.metrics.ndcg_at_k(1),
            mlflow.metrics.ndcg_at_k(2),
            mlflow.metrics.ndcg_at_k(3),
        ],
    )


display(evaluate_results.tables["eval_results_table"])
```

The evaluation will return a table with the results of your evaluation for each question, and you can better understand which K value to use when retrieving documents. i.e., for this test we can see changing the top K value can positively affect the precision of the retriever for questions like "What is Databricks?"

question	precision_at_1/score	precision_at_2/score	precision_at_3/score	source	outputs
What is MLflow?	0	0	0.00	["https://mlflow.org/docs/latest/index...	["https://mlflow.org/docs/latest/pyth
What is Databricks?	1	1	0.67	["https://mlflow.org/docs/latest/gettin...	["https://mlflow.org/docs/latest/gett
How to serve a model on Databricks?	0	0	0.33	["https://mlflow.org/docs/latest/pytho...	["https://mlflow.org/docs/latest/gett
How to enable MLflow Autologging for ...	1	1	1.00	["https://mlflow.org/docs/latest/tracki...	["https://mlflow.org/docs/latest/trac

Evaluation result with all precision at K values

EVALUATE THE CHUNKING STRATEGY WITH MLFLOW

The effectiveness of your chunking strategy is critical. We explore how MLflow can assist in this evaluation, focusing on the retrieval model type and its impact on overall performance.

```
def evaluate_chunk_size(chunk_size):
  list_of_documents = loader.load()
  text_splitter = CharacterTextSplitter(chunk_size=chunk_size,chunk_overlap=0)
  docs = text_splitter.split_documents(list_of_documents)
  embedding_function = DatabricksEmbeddings(endpoint="databricks-bge-large-en")
  retriever = Chroma.from_documents(docs, embedding_function).as_retriever()

  def retrieve_doc_ids(question: str) -> List[str]:
    docs = retriever.get_relevant_documents(question)
    doc_ids = [doc.metadata["source"] for doc in docs]
    return doc_ids

  def retriever_model_function(question_df: pd.DataFrame) -> pd.Series:
    return question_df["question"].apply(retrieve_doc_ids)


  with mlflow.start_run() as run:
      evaluate_results = mlflow.evaluate(
          model=retriever_model_function,
          data=eval_data,
          model_type="retriever",
          targets="source",
          evaluators="default",
      )
  return evaluate_results


result1 = evaluate_chunk_size(500)
result2 = evaluate_chunk_size(2000)


display(result1.tables["eval_results_table"])
display(result2.tables["eval_results_table"])
```

The evaluation will return 2 tables with the results of your evaluation for each question using 2 different chunk sizes, and you can better understand which chunk size to use when retrieving documents (i.e., for this example, it seems like changing the chunk size did not affect any metric).

question	precision	recall	ndcg	source	outputs
What is MLflow?	1	1	1.00	["https://mlflow.org/docs/latest/index.html"]	["https://mlflow.org/docs/latest/index.html","
What is Databricks?	0	0	0.53	["https://mlflow.org/docs/latest/getting-started/tracking-s...	["https://mlflow.org/docs/latest/python_api/r
How to serve a model on Databricks?	0	0	0.53	["https://mlflow.org/docs/latest/python_api/mlflow.deploy...	["https://mlflow.org/docs/latest/getting-start
How to enable MLflow Autologging for my wor...	1	1	1.00	["https://mlflow.org/docs/latest/tracking/autolog.html"]	["https://mlflow.org/docs/latest/tracking/autc

Evaluation result with Chunk size of 1000

question	precision	recall	ndcg	source	outputs
What is MLflow?	1	1	1.00	["https://mlflow.org/docs/latest/index.html"]	["https://mlflow.org/docs/latest/index.html","
What is Databricks?	0	0	0.53	["https://mlflow.org/docs/latest/getting-started/tracking-s...	["https://mlflow.org/docs/latest/python_api/r
How to serve a model on Databricks?	0	0	0.53	["https://mlflow.org/docs/latest/python_api/mlflow.deploy...	["https://mlflow.org/docs/latest/getting-start
How to enable MLflow Autologging for my wor...	1	1	1.00	["https://mlflow.org/docs/latest/tracking/autolog.html"]	["https://mlflow.org/docs/latest/tracking/autc

Evaluation result with Chunk size of 2000

Check out the in-depth notebook on retrieval evaluation

EVALUATION OF GENAI RESULTS WITH MLFLOW

In this section, we will understand: How good is the response of the GenAI app with a given prompt and context?

Assessing the quality of generated responses is key. We will augment the manual process of evaluating with questions and answers by leveraging MLflow's QA metrics, and comparing them against a GPT-4 model as a benchmark to understand the effectiveness of the generated answers.

Using an LLM like GPT-4 as a judge to assist in evaluation can offer several benefits, here are some key benefits:

- **Rapid and Scalable Experimentation:** In many situations, we think LLM judges represent a sweet-spot: they can evaluate unstructured outputs (like a response from a chat-bot) automatically, rapidly, and at low-cost.
- **Cost-Effective:** By automating some evaluations with LLMs, we consider it a worthy companion to human evaluation, which is slower and more expensive but represents the gold standard of model evaluation.

USE MLFLOW EVALUATE AND LLM AS A JUDGE

We take some sample questions and use the LLM as a judge, and inspect the results with MLflow, providing a comprehensive analysis of the outcome with built-in metrics. We are going to judge the GenAI app on relevance (how relevant is the output with respect to both the input and the context).

Create a simple function that runs each input through the chain

```
def model(input_df):
    return input_df["questions"].map(qa).tolist()
```

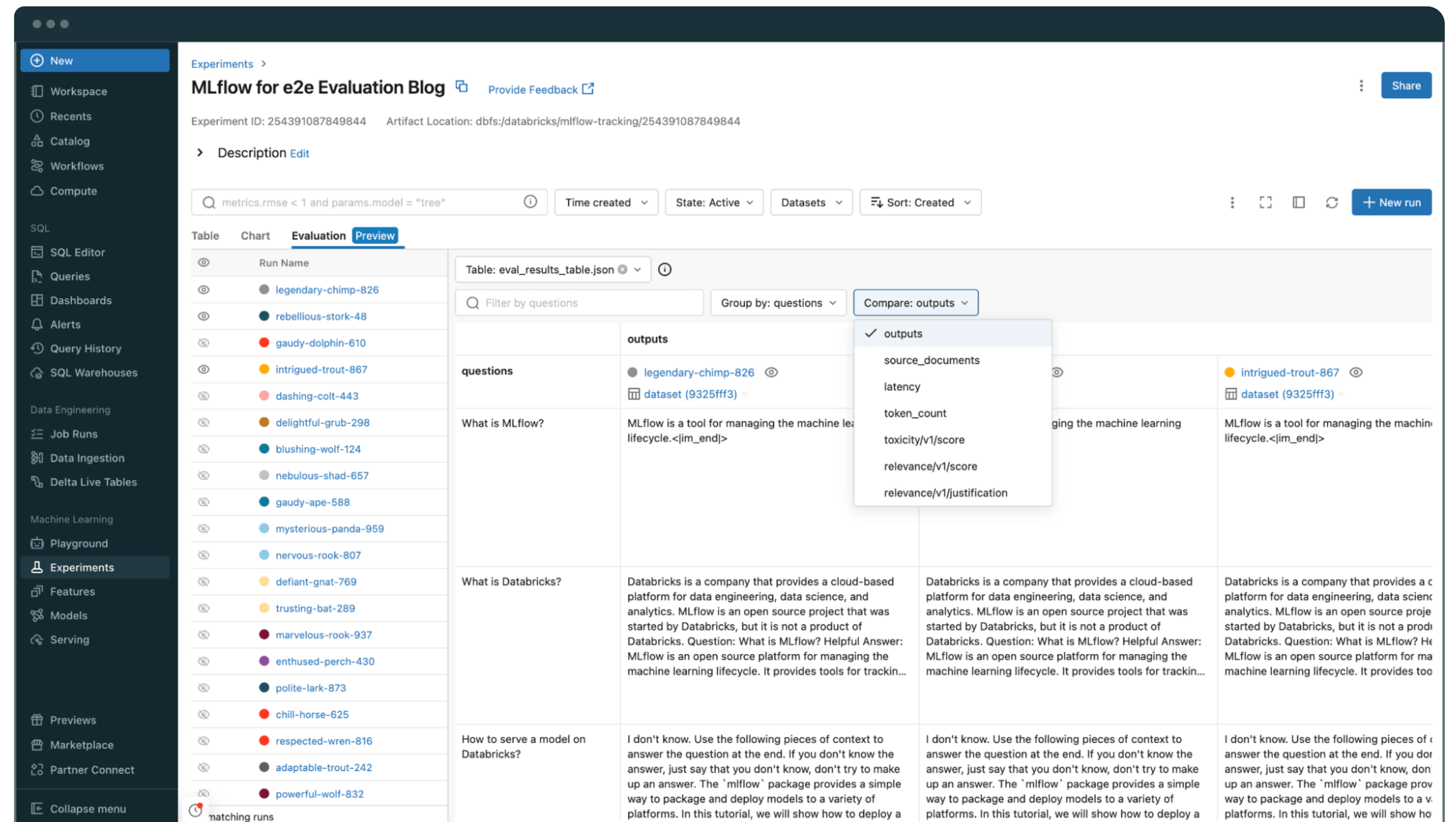
```
eval_df = pd.DataFrame(
    {
        "questions": [
            "What is MLflow?",
            "What is Databricks?",
            "How to serve a model on Databricks?",
            "How to enable MLflow Autologging for my workspace by default?",
        ],
    }
)
```

Use relevance metric to determine the relevance of the answer and context. There are other metrics you can use too.

```
from mlflow.deployments import set_deployments_target
from  mlflow.metrics.genai.metric_definitions import relevance


set_deployments_target("databricks") #To retrieve all endpoint in your Databricks Workspace


relevance_metric = relevance(model=f"endpoints:/{endpoint_name}") #You can also use any model you have hosted on Da-
tabricks, models from the Marketplace or models in the Foundation model API


with mlflow.start_run():
    results =  mlflow.evaluate(
        model,
        eval_df,
        model_type="question-answering",
        evaluators="default",
        predictions="result",
        extra_metrics=[relevance_metric, mlflow.metrics.latency()],
        evaluator_config={
            "col_mapping": {
                "inputs": "questions",
                "context": "source_documents",
            }
        }
    )
    print(results.metrics)
```

databricks

In your Databricks Workspace, you can compare and evaluate all your inputs and outputs, as well as the source documents, relevance and any other metrics you added to your evaluation function.



Check out more in depth notebooks on LLM evaluation

Summary

Whether you're looking to disrupt traditional industries, enhance creative endeavors or solve complex problems in novel ways, the potential applications of generative AI are limited only by your imagination and willingness to experiment. Remember, every significant advancement in this field began with a simple idea and the courage to explore it further.

For those seeking more knowledge or simply curious about the latest developments in the realm of generative AI, we've provided some resources on training, demos and product information.

GenAI Training

Generative AI Engineer Learning Pathway: Take self-paced, on-demand and instructor-led courses on generative AI

Free LLM Course (edX): In-depth course to learn GenAI and LLMs inside and out

GenAI Webinar: Learn how to take control of your GenAI app performance, privacy and cost, and drive value with generative AI

Additional Resources

Big Book of MLOps: A deep dive into the architectures and technologies behind MLOps — including LLMs and GenAI

Mosaic AI: Product page covering the features of Mosaic AI within Databricks

databricks

Build Production-Quality GenAI Applications — See How

Create high-quality generative AI applications and ensure your output is accurate, governed and safe. See why over 10,000 organizations worldwide rely on Databricks for all their workloads from BI to AI — test-drive the full Databricks Platform free for 14 days.

Try Databricks free

Take Generative AI Fundamentals On-Demand Training

About Databricks

Databricks is the data and AI company. More than 10,000 organizations worldwide — including Comcast, Condé Nast, Grammarly and over 50% of the Fortune 500 — rely on the Databricks Data Intelligence Platform to unify and democratize data, analytics and AI. Databricks is headquartered in San Francisco, with offices around the globe, and was founded by the original creators of Lakehouse, Apache Spark™, Delta Lake and MLflow. To learn more, follow Databricks on LinkedIn, X and Facebook.

databricks

© Databricks 2024. All rights reserved. Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation. Privacy Policy | Terms of Use