💡 Generative AI on Ray

About

Welcome to the Generative AI class at Ray Summit 2023! Incorporating multiple data modalities into AI applications is my personal favorite topic to follow in ML. I remember a time (only a few years ago!) when generative computer vision involved highly custom architectures and painstakingly slow inference, only in research labs. Today, we see text-to-image consumer products hitting the market, and they're changing the way people work, learn, and relate to one another.

In this class, we'll find ourselves on an abridged version of this journey from tinkering to production. Using Dreambooth and Stable Diffusion as a stand-in for any multimodal, deep learning model, we'll show how to fine-tune and deploy this in a robust pipeline. We're lucky to have such a diverse audience today, so we encourage healthy discussion, questions, and to learn from one another. Let's dive in!

Meet the team

Emmy Li	Kourosh Hakhamaneshi	Justin Yu
emmy@anyscale.com	kourosh@anyscale.com	justinvyu@anyscale.com

How to participate

Join live polls and engage in Q&A by going to **app.sli.do** and enter code `#Ray-GenAI`. You can also ask questions live by calling over an instructor.

How to access Anyscale

Anyscale, the company founded by the creators of Ray, simplifies the development of distributed applications for machine learning. Simply put, it's the best place to run Ray, and today, you'll have access to a provisioned cluster with zero set-up.

1	Log-in via **console.anyscale.com** with the credentials sent to your email.
2	Select "Workspaces" in the side panel and open your workspace.

3	Click on the Jupyter or VSCode icon to access the course materials.

Fine-tuning Stable Diffusion

1. Set-Up

📌 Note on GPU Memory

The base model, `CompVis/stable-diffusion-v1-4`, happens to fit on a single one of our GPUs. When confronted with limited GPU memory, you can first try workarounds like loading in the weights with float16 precision instead of the default float32. If that still isn't enough, Ray Train's TorchTrainer allows you to launch your DeepSpeed training across a distributed Ray cluster.

💡 Fun Fact

The Dreambooth paper is among a growing number of publications that explores "few-shot learning" for text-to-image models, allowing us to tailor pre-trained models to new tasks with minimal additional training data.

🧪 Try it yourself

1. For fine-tuning, we're downloading five photos of a dog to train the Stable Diffusion model, but these images could be of anything!
   a. Instead of using these stock images, replace them with your favorite object or person. You should only need 3-5 instances of the subject matter to achieve decent results.

2. Augmenting the Training Dataset

🔎 Inspect

1. Take a closer look at how we generate class images with Ray Data.
   a. Why do we use `flat_map`? Outside of this example, can you think of ML data processing workloads where you would choose to use `flat_map`?
   b. What happens when you remove the `.repartition(2)` step? Run this transformation again, and look at the Ray Dashboard to see the effect.
   c. The Ray Data ActorPoolStrategy specifies the autoscaling behavior of a Dataset transformation. Instead of a fixed-size pool, try setting a `min_size` and a `max_size` and observe the resulting behavior in the Ray Dashboard.

3. Creating the Training Dataset

🛣️ Detour Through Utilities

We've obscured the way that the training dataset is constructed from the main notebook to keep things simple, but it's worth a taking a peek!

1. There are several Ray Data APIs used in `get_train_dataset` like batch processing, preprocessors, repartitioning, and basic transformations like adding and dropping columns.
   a. Walk through each step and see if you can identify a pattern here that would fit into an existing ML project that you're working on.
   b. Monitor the Ray Dashboard as the dataset is built, especially the Actors and Metrics tabs.

4. Run Fine-Tuning

🔬 Focus on "Launch fine-tuning"

There's quite a bit to unpack in the training function. The thing to focus on, however, is that this training logic will be unchanged from your original workflow. The `train_loop_config` specifies hyperparameters and other arguments already in your workflow as well. Ray's Trainer wraps around this function and config to then scale it to the worker nodes in the Ray cluster. Notice that the dataset we feed in for fine-tuning is a Ray Dataset, which composes nicely with the Trainer to complete the end-to-end pipeline.

5. Test the Fine-Tuned Model

🧪 Try it yourself

1. Modify the prompts list to generate the same subject matter with different contexts, properties, views, and/or accessories. See if you can build a list of prompts that qualitatively probe the performance of the fine-tuned model.
   a. Can you spot weaknesses in the training dataset methodology?
2. Compare results with the base model.
   a. Generate images using the base model with a bit of prompt engineering. Compare the quality and relevance of the images generated by the fine-tuned model versus the base model.

Advanced Activities

1. Run fine-tuning again, but this time, interrupt the run somehow.
   a. How does Ray handle node failures?
   b. How does Ray Train's checkpointing work?
2. Cost-benefit analysis
   a. Discuss the financial and computational costs of how you choose to scale. Is scaling horizontally (adding more machines) more effective than scaling vertically (adding more power to a single machine) for your use case?

✏️ Your notes

Serving Stable Diffusion

1. Introduction to Ray Serve

📚 Glossary

Ray Serve: A scalable and flexible model-serving library built on top of the Ray distributed computing framework, designed to simplify the deployment and management of machine learning models and other services.

Ray Cluster: A set of interconnected nodes managed by the Ray framework to distribute and parallelize computation tasks and data across multiple machines.

Node: A Ray node is a physical or virtual machine that can run one or multiple Ray processes to execute tasks.

Deployment: A user-defined, versioned unit of code, that can contain ML models or business logic and is encapsulated as a Python class or function. It can be horizontally scaled and accessed via HTTP or Python APIs.

Replica: An individual instance of a deployment, running as a separate Ray Actor, that handles incoming requests and can be autoscaled to adjust to the volume of incoming traffic.

ServeHandle: A ServeHandle is a reference to a bound deployment that allows for programmatic interaction with the deployment. This allows multiple independent deployments to call into each other and facilitates flexible and complex model composition where bound deployments can reference other bound deployments.

Application: An application is composed of one or more deployments and can be accessed via HTTP routes or Python handles, working together to provide a specific service or functionality.

💡 Fun Fact

Ray Serve supports model multiplexing, which is the ability to host multiple machine learning models within a single deployment or service. This allows a single endpoint to serve requests for multiple models (up to thousands). A great example of this is the “Ray Docs AI” LLM application from this year’s keynote!

This approach simplifies infrastructure and resource management, as you don't need to deploy a new service for each model, and can also improve resource utilization by sharing resources like memory and CPU among multiple models.

2. How to Create a Basic Service

🔬 Focus on "`.bind()`"

The `.bind()` method is a crucial step in Ray Serve. It not only binds arguments to the constructor but also facilitates the creation of "ServeHandles," enabling inter-deployment communication.

💡 Fun Fact

While this was just a simple service, these same components from Ray Serve are what allows companies like Samsara to save 50% on total annual inference costs, Ant Financial to build a 4000 GPU online inference cluster, and Anyscale to launch LLM Endpoints for 50% cheaper that GPT 3.5. And it's now generally available!

3. Deploying Dreambooth

📣 Discussion Questions

1. How does the `APIIngress` class serve as the entry point for all API requests?
2. Discuss the role of FastAPI in this configuration.

4. Making Requests to the Endpoint

🏃 Exercises

Many more companies are putting ML into production than ever before. This means that a solid observability story is critical so that you can sleep at night while your service is running. These exercises are aimed at exploring how to monitor your online inference application.

1. Prompt variations
    a. Try changing the `prompt` and `image_size`. How does this affect the quality of the generated images? How does this affect metrics like latency?
2. Client-side Load Balancing
    a. Modify the existing code to generate a large number of requests to the Dreambooth service.
    b. Open the Ray Dashboard and look at the "Serve" tab. Look through the logs, metrics, and autoscaling behavior for your service.
    c. Open Grafana for additional metrics and advanced dashboard visualization.
3. Resource constraints

a. Try running the Stable Diffusion deployment with different resource allocations (`num_cpus`, `num_gpus`). Observe the changes and/or any error messages with your experiment in the Ray Dashboard.

✏️ **Your notes**

Advanced Activities

Autoscaling

Try to change the `autoscaling_config` defined in the Stable Diffusion deployment to further specify resource management. You can:

1. Specify the number of CPUs, GPUs, and custom resources.
2. Fix the number of replicas.
3. Check out Resource Management in Ray Serve for more details.
4. After each change, go to the Ray Dashboard to see active Serve deployments.

Deployment Composition

We can compose multiple deployments together. You already experimented with this by binding an Ingress deployment with the StableDifusion deployment. See if you can add in another deployment. Some ideas include:

1. Add in a filter that screens out grayscale and/or censored images.
2. Prompt engineer before the image gets generated to encourage a diversity of results.
3. Use the base model for prompts unrelated to the subject matter and the fine-tuned model for unique subject matter prompts.

✏️ **Your notes**

Feedback survey

We're so glad to have you along in our classroom. To improve our content and presentation for the next batch of future learners, we would love to hear how your experience was.

Please visit **bit.ly/ray-summit-feedback** to let us know how today went!

More resources

Self-Paced Ray & Anyscale Education	Access the best course materials from Ray Summit and get a sneak preview of technical content releases before they become public at **training.anyscale.com**
Docs	Your one-stop-shop for all things Ray at **docs.ray.io**
Blogs	Read our latest news and findings at **anyscale.com/blog**
YouTube	Watch a curated set of tutorials at **youtube.com/anyscale**
Anyscale	Interested in a managed Ray service? Go to **anyscale.com/sign-up**