

# Hugging Face + Ray AIR: Scaling Transformers

Jules S. Damji, Antoni Baum  
Anyscale, Ray Team  
Data Council 2023, Austin, TX



# A Quick Poll ...



# Who we are ...

## Jules S. Damji

- Dev Adv @ Anyscale, Databricks & Hortonworks
- SWE at
  - Sun Microsystems,
  - Netscape,
  - @Home
  - Opsware/LoudCloud,
  - VeriSign,



## Antoni Baum

- Software Engineer at Anyscale
- On Libraries Team
  - AIR, Train, Tune
- Open source enthusiast!





# anyscale

**Who we are:** Original creators of Ray, a unified framework for scalable computing

**What we do:** Scalable compute for AI/ML and Python

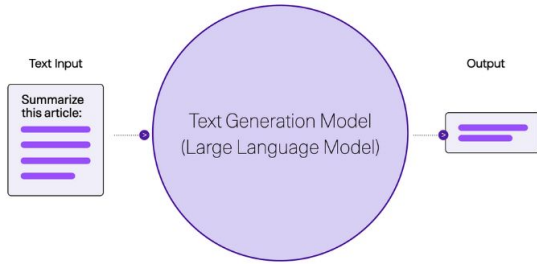
**Why we do it:** Scaling is a necessity, scaling is hard; make distributed computing easy and simple for everyone

# Agenda

1. State of ML and AI today ...
2. Hugging Face for cutting edge ML
3. Distributed training is a necessity
4. 🙌 + Ray AIR = easy distributed training
5. Deep Dive into Ray AIR Trainer
6. Demo

# State of ML and AI today ...

Text Generation: Software that generates coherent human language



Text generation models are a central pillar of Generative AI.

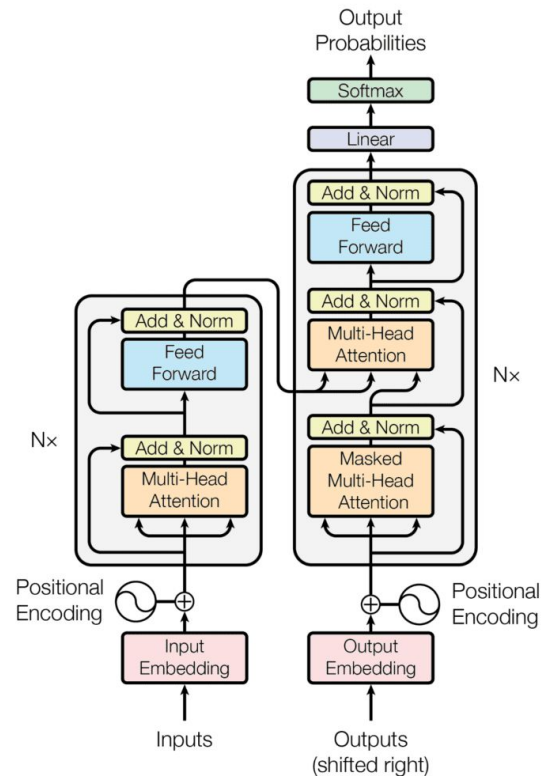


Image generation models create (often astounding) images guided by text prompts.

- Deep Learning + Transformers are the SOTA
- Impressive foundational & LLM models (e.g., GPT-3, DALL-E, ChaptGPT, Stability, etc)
- Generative AI
  - Text classification, sentiment analysis
  - Toxicity, entity recognition, language translation,
  - Sentence completion, text-2-image generation, q & a etc

# What are Transformers?

- [Attention Is All You Need](#) (circa 2017, Vaswani et al)
- Deep Neural Networks
  - Encoders
  - Attention heads
  - Decoders
  - Attention heads
- Final layers
  - Linear & Softmax





# Transformers for SOTA ML/AI

## Simple, robust and powerful

- Library for Python developers
- Provides an opinionated, high level API
- Mostly focused on NLP
- Multiple LLM models (GPT, BERT, etc.)
- Huge community & social focus

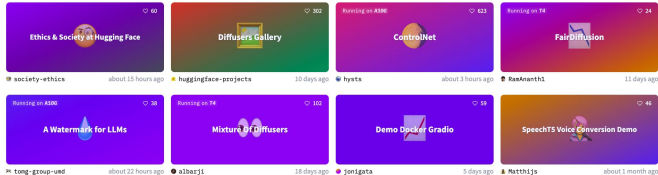


Spaces

Discover amazing ML apps made by the community!

Search Spaces

Spaces of the week



Search models, datasets, users...

Models Datasets Spaces Docs Solutions Pricing

Filter Tasks by name

Multimodal

- Feature Extraction
- Text-to-Image
- Image-to-Text
- Visual Question Answering
- Document Question Answering
- Graph Machine Learning

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Image-to-Image
- Unconditional Image Generation
- Video Classification
- Zero-Shot Image Classification

Natural Language Processing

- Text Classification
- Text Classification
- Table Question Answering
- Question Answering
- Zero-Shot Classification
- Translation
- Summarization
- Conversational
- Text Generation
- Text2Text Generation
- Fill-Mask
- Sentence Similarity

Models 150,049

Filter by name

Model Name	Updated	Downloads
<b>bert-base-uncased</b>	Updated Nov 16, 2022	50.6M
<b>gpt2</b>	Updated Dec 16, 2022	10.7M
<b>15-base</b>	Updated Jan 24	10.5M
<b>distilbert-base-uncased</b>	Updated Nov 16, 2022	9.89M
<b>pytorch-lstm</b>	Updated Oct 27, 2021	8.11M
<b>xln-roberta-large</b>	Updated Jan 27, 2022	7.98M
<b>roberta-base</b>	Updated 8 days ago	6.83M
<b>ConoVis/stable-diffusion-v1-4</b>		
<b>emilysentner/Bio_ClinicalBERT</b>	Updated Feb 27, 2022	33.3M
<b>xln-roberta-base</b>	Updated Nov 26, 2022	33.2M
<b>openai/clip-vit-large-patch14</b>	Updated Oct 4, 2022	30.1M
<b>StanfordAIMI/stanford-deidentifier-base</b>	Updated Nov 23, 2022	9.79M
<b>bert-base-cased</b>	Updated Nov 16, 2022	8.44M
<b>microsoft/layoutlmv3-base</b>	Updated Dec 13, 2022	7.38M
<b>albert-base-v2</b>	Updated Aug 26, 2022	4.83M
<b>openai/clip-vit-base-patch32</b>		

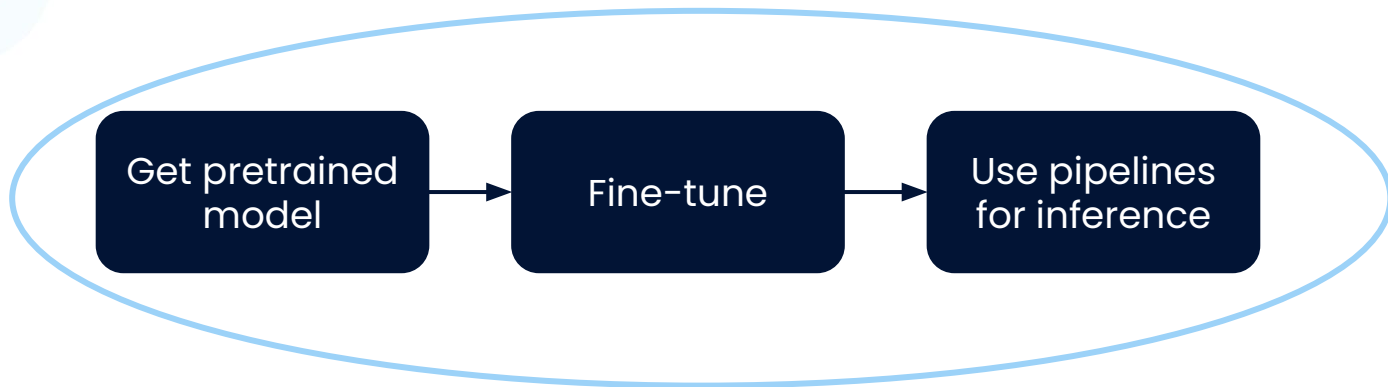




# How 🤗 makes ML/AI easier ...

- Abstract DL complexities with simple flow
- Increase developer velocity
- Huge 🤗 Hub to choose from

Simple flow & abstraction



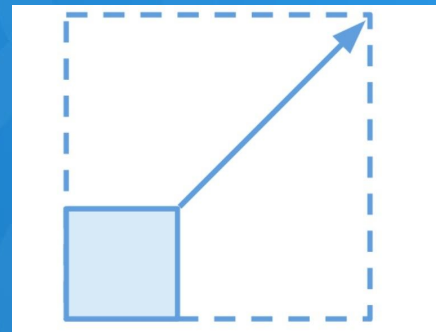
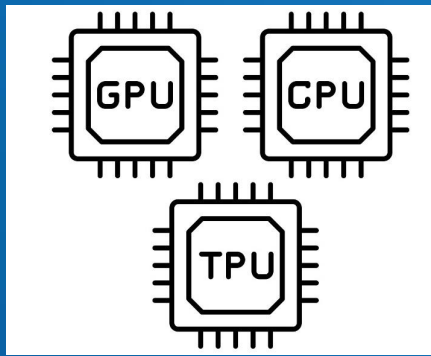
# How 🙌 makes ML/AI easier ...

- Abstract DL complexities with simple flow
- Increase developer velocity
- Huge 🙌 Hub to choose from

```
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer
from datasets import load_dataset
```

```
dataset = load_dataset("yelp_review_full")
train_dataset, eval_dataset = dataset["train"], dataset["test"]
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)
training_args = TrainingArguments(f"{model_checkpoint}-yelp", evaluation_strategy="epoch")
trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, eval_dataset=eval_dataset)
trainer.train()
```

# SOTA models need loads of compute!

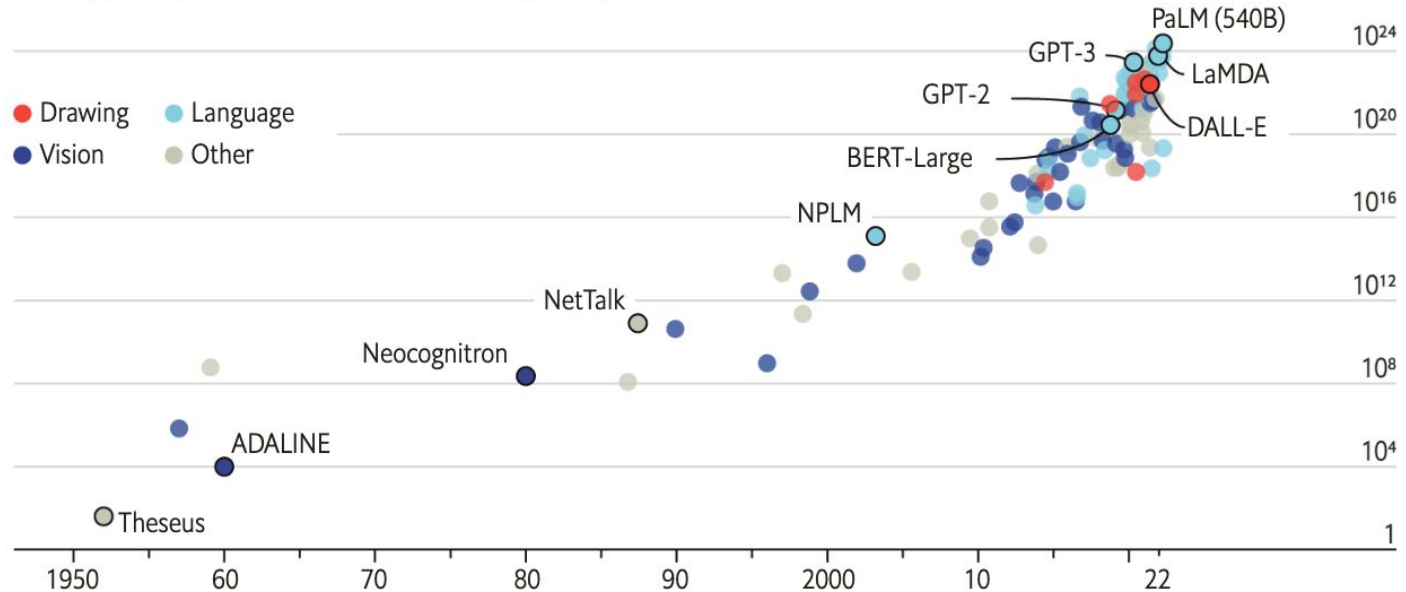


# Blessings of scale ...

## The blessings of scale

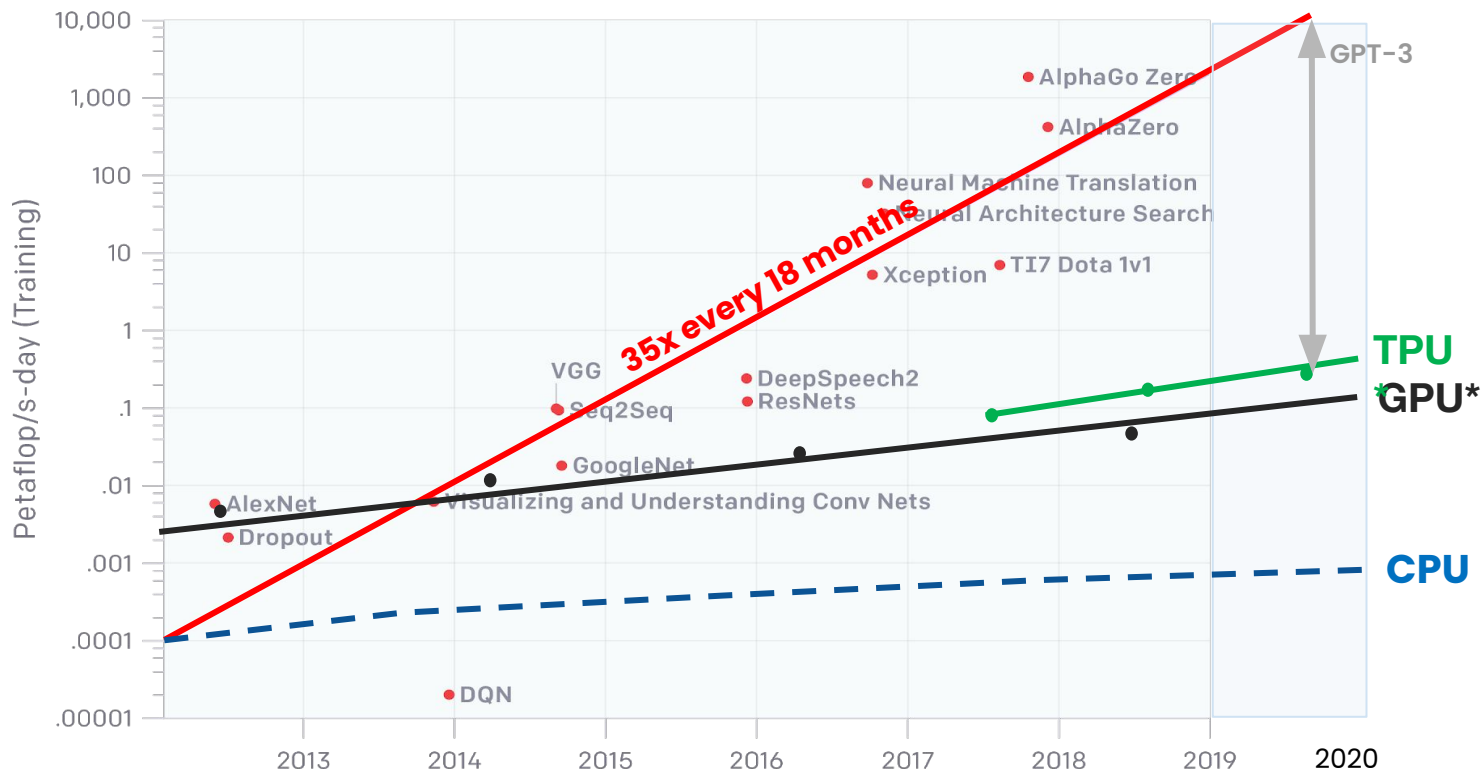
AI training runs, estimated computing resources used

Floating-point operations, selected systems, by type, log scale

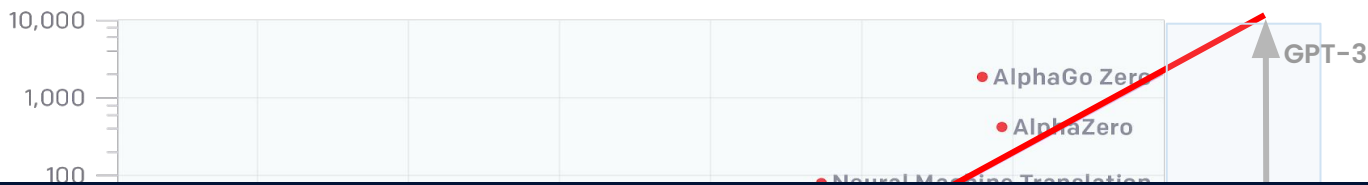


Sources: "Compute trends across three eras of machine learning", by J. Sevilla et al., arXiv, 2022; Our World in Data

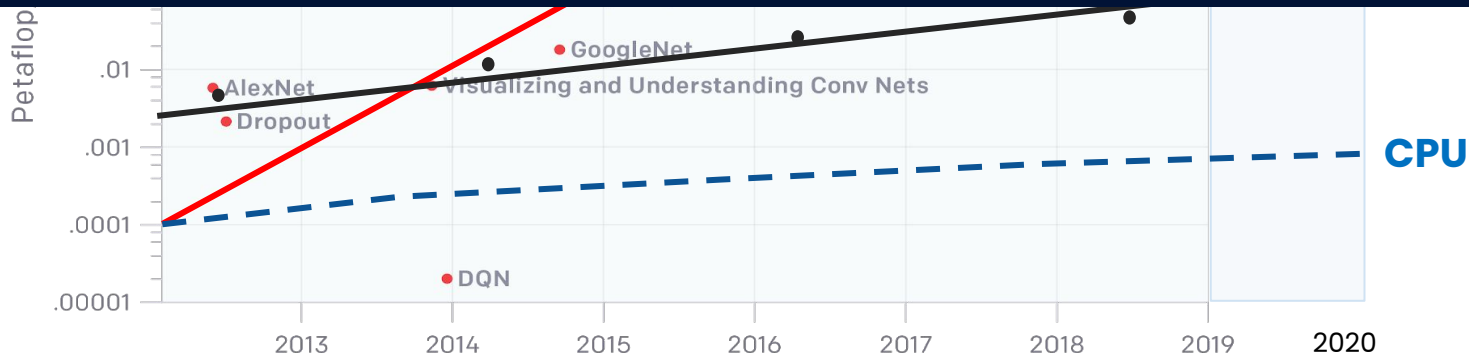
# Compute – supply demand problem



# Specialized hardware is not enough



**No way out but to distribute!**



# We have to go distributed

## New problems!

- Slow Developer velocity
- Managing complex infrastructure
- Keeping end-to-end ML pipelines scalable

# Solution is Ray AIR

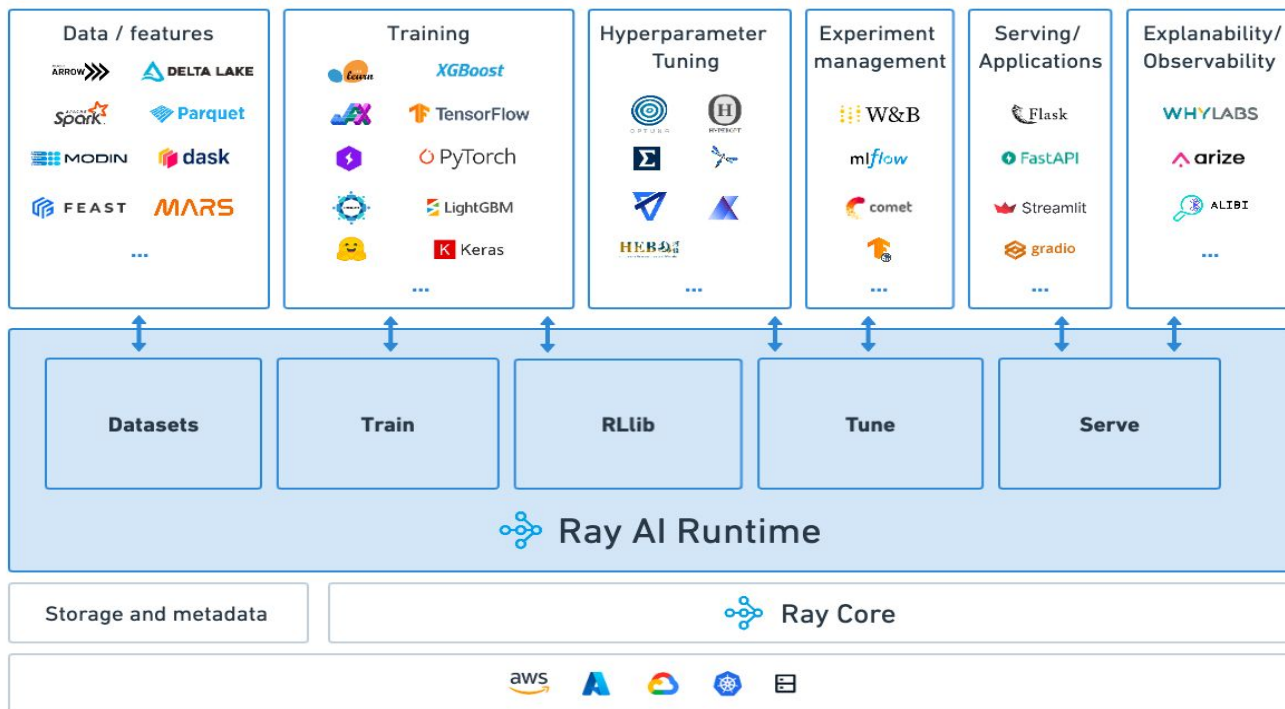
**Ray AI Runtime (AIR)** is a scalable, unified toolkit for both data scientists and software engineers.

Ray AIR provides a flexible, pythonic framework for each step of the ML workflow.





# The Ray AI Runtime (Ray AIR)



# When to use Ray AIR

Scale a single type of workload

Scale end-to-end ML applications

Run ecosystem libraries using a unified API

Build a custom ML platform

# Why use Ray & Ray AIR

Efficient data layer and distributed object store

Robust scheduling and resource management

Python-based API

**Build a top Ray:**

- Compute strata Infrastructure
- Addresses challenges of distributed computing!

# Combining Hugging Face and Ray AIR



# Ray AIR's 🤗 Trainer: API

```
def trainer_init_per_worker(train_dataset, eval_dataset, **config):  
    # HF code goes here  
    return transformers.Trainer(...)  
  
scaling_config = ScalingConfig(num_workers=3, use_gpu=True)  
  
trainer = HuggingFaceTrainer(  
    trainer_init_per_worker=trainer_init_per_worker,  
    scaling_config=scaling_config,  
    datasets={"train": ray_train_ds, "evaluation": ray_evaluation_ds},  
)  
  
result = trainer.fit()
```

1. Use existing HF code in a function
2. Provide ScalingConfig & other Ray AIR configs if needed
3. Initialize the HuggingFaceTrainer with Ray Datasets
4. Fit the trainer
5. Inspect the results

# Ray AIR's 🤗 Trainer: API

```
def trainer_init_per_worker(train_dataset, eval_dataset, **config):  
    # HF code goes here  
    return transformers.Trainer(...)  
  
scaling_config = ScalingConfig(num_workers=3, use_gpu=True)  
  
trainer = HuggingFaceTrainer(  
    trainer_init_per_worker=trainer_init_per_worker,  
    scaling_config=scaling_config,  
    datasets={"train": ray_train_ds, "evaluation": ray_evaluation_ds},  
)  
  
result = trainer.fit()
```

1. Use existing HF code in a function
2. Provide ScalingConfig & other Ray AIR configs if needed
3. Initialize the HuggingFaceTrainer with Ray Datasets
4. Fit the trainer
5. Inspect the results

# Ray AIR's 🤗 Trainer: API

```
def trainer_init_per_worker(train_dataset, eval_dataset, **config):  
    # HF code goes here  
    return transformers.Trainer(...)  
  
scaling_config = ScalingConfig(num_workers=3, use_gpu=True)  
  
trainer = HuggingFaceTrainer(  
    trainer_init_per_worker=trainer_init_per_worker,  
    scaling_config=scaling_config,  
    datasets={"train": ray_train_ds, "evaluation": ray_evaluation_ds},  
)  
  
result = trainer.fit()
```

1. Use existing HF code in a function
2. Provide ScalingConfig & other Ray AIR configs if needed
3. Initialize the HuggingFaceTrainer with Ray Datasets
4. Fit the trainer
5. Inspect the results

# Ray AIR's 🤗 Trainer: API

```
def trainer_init_per_worker(train_dataset, eval_dataset, **config):  
    # HF code goes here  
    return transformers.Trainer(...)  
  
scaling_config = ScalingConfig(num_workers=3, use_gpu=True)  
  
trainer = HuggingFaceTrainer(  
    trainer_init_per_worker=trainer_init_per_worker,  
    scaling_config=scaling_config,  
    datasets={"train": ray_train_ds, "evaluation": ray_evaluation_ds},  
)  
  
result = trainer.fit()
```

1. Use existing HF code in a function
2. Provide `ScalingConfig` & other Ray AIR configs if needed
3. Initialize the `HuggingFaceTrainer` with Ray Datasets
4. Fit the trainer
5. Inspect the results



# Ray AIR's 🤗 Trainer: API

```
def trainer_init_per_worker(train_dataset, eval_dataset, **config):  
    # HF code goes here  
    return transformers.Trainer(...)  
  
scaling_config = ScalingConfig(num_workers=3, use_gpu=True)  
  
trainer = HuggingFaceTrainer(  
    trainer_init_per_worker=trainer_init_per_worker,  
    scaling_config=scaling_config,  
    datasets={"train": ray_train_ds, "evaluation": ray_evaluation_ds},  
)  
  
result = trainer.fit()
```

1. Use existing HF code in a function
2. Provide `ScalingConfig` & other Ray AIR configs if needed
3. Initialize the `HuggingFaceTrainer` with Ray Datasets
4. Fit the trainer
5. Inspect the results

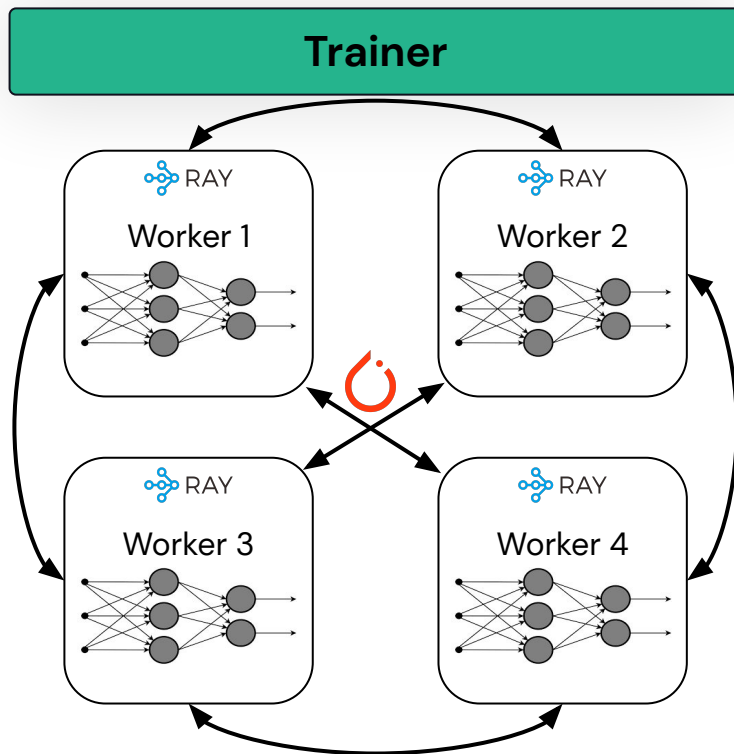
# Ray AIR's 🤗 Trainer Implementation

# Ray AIR's 🤗 Trainer: Implementation

- Distributed Data Parallel/FSDP training on a Ray Cluster
  - Takes advantage of PyTorch DDP & Hugging Face support for it
- Runs user-defined Hugging Face code without any changes
- Automatically converts Ray Datasets to format expected by Hugging Face
- Built-in logging & monitoring
- Upcoming: Separate AccelerateTrainer for lower level code with 🤗 Accelerate

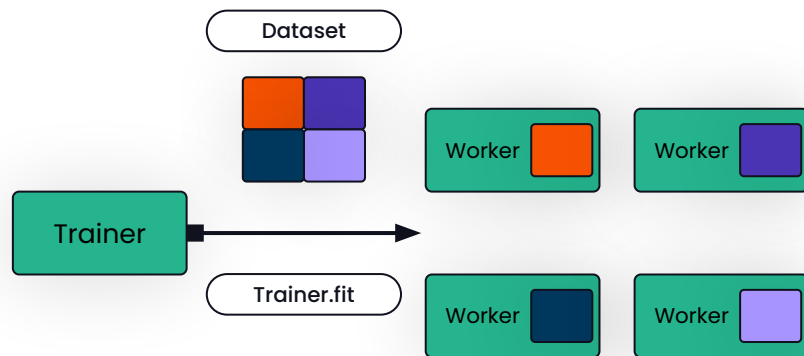
# Ray AIR's 🤗 Trainer: Parallelization

- PyTorch DDP on a Ray Cluster
  - FSDP, DeepSpeed are also supported
- Abstracts away infrastructure
- Supports both CPU and GPU workers



# Ray AIR's 🤗 Trainer: Data ingest

- Ray AIR uses Ray Datasets as a common data format
- Easily read from disk/cloud, or from other formats
- Fully distributed
- Can handle data too big to fit on one node or even the entire cluster



# Ray AIR's 🤗 Trainer: Preprocessors

- Ray AIR provides out-of-box preprocessors for common ML tasks
- You can also write your own UDFs to map-apply
- Automatically applied during training and inference

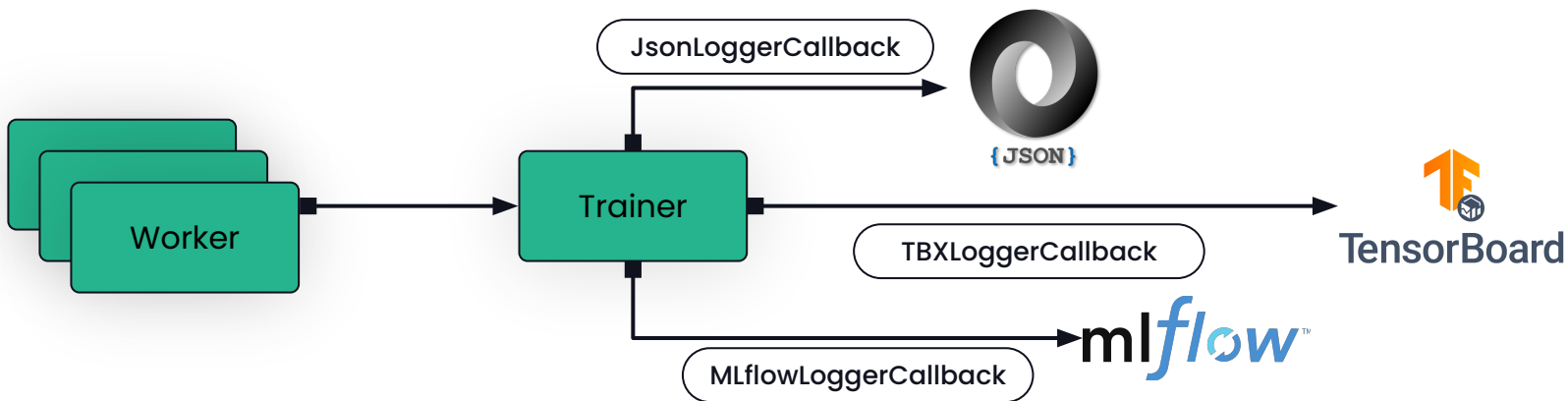
```
from ray.train.huggingface import HuggingFaceTrainer
from ray.data.preprocessors import BatchMapper
from transformers import AutoTokenizer

def tokenize_function(df):
    tokenizer = AutoTokenizer.from_pretrained("sgugger/gpt2-like-tokenizer")
    return tokenizer(df["text"])

batch_tokenizer = BatchMapper(tokenize_function)
trainer = HuggingFaceTrainer(
    trainer_init_per_worker=train_function,
    scaling_config=ScalingConfig(num_workers=num_workers, use_gpu=use_gpu),
    datasets={"train": ray_train, "evaluation": ray_validation},
    preprocessor=batch_tokenizer,
)
```

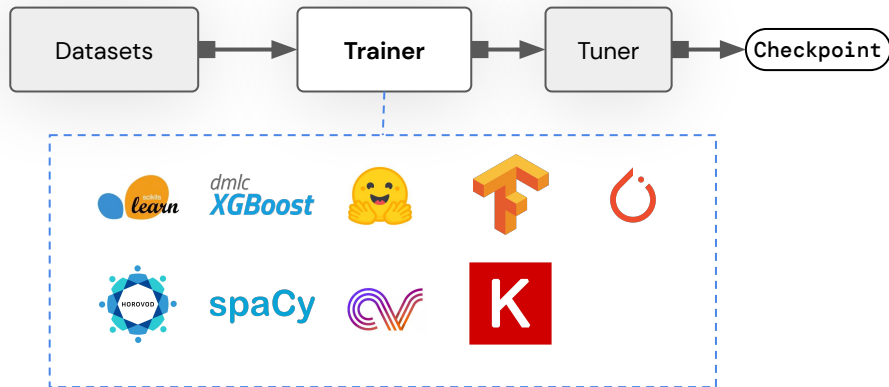
# Ray AIR's 🤗 Trainer: Logging & Monitoring

- All Hugging Face metrics are reported every epoch
- Use Ray AIR callbacks for Tensorboard, MLflow, Weights & Biases, Comet, etc.
- Inspect Result after training



# Ray AIR's 🤗 Trainer: Checkpointing

- Automatic, configurable checkpointing
- Resume training from Checkpoint object
- Enables spot instance usage
- Use the Checkpoint for inference & serving







# training workflow

```
dataset = load_dataset("yelp_review_full")
train_dataset, eval_dataset = dataset["train"], dataset["test"]
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased",
num_labels=5)
training_args = TrainingArguments(f"{model_checkpoint}-yelp",
evaluation_strategy="epoch")
trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset,
eval_dataset=eval_dataset)
trainer.train()
```



# training workflow, distributed with Ray AIR

```
dataset = load_dataset("yelp_review_full")
train_dataset, eval_dataset = dataset["train"], dataset["test"]
def trainer_init_per_worker(train_dataset, eval_dataset, **config):
    model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)
    training_args = TrainingArguments(f"{model_checkpoint}-yelp", evaluation_strategy="epoch")
    trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, eval_dataset=eval_dataset)
    return trainer
```



# training workflow, distributed with Ray AIR

```
dataset = load_dataset("yelp_review_full")
train_dataset, eval_dataset = dataset["train"], dataset["test"]
def trainer_init_per_worker(train_dataset, eval_dataset, **config):
    model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)
    training_args = TrainingArguments(f"{model_checkpoint}-yelp", evaluation_strategy="epoch")
    trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, eval_dataset=eval_dataset)
    return trainer
trainer = HuggingFaceTrainer(
    trainer_init_per_worker=trainer_init_per_worker,
    scaling_config=ScalingConfig(num_workers=3, use_gpu=True),
    datasets={"train": ray.data.from_huggingface(train_dataset), "evaluation": ray.data.from_huggingface(eval_dataset)},
)
```



# training workflow, distributed with Ray AIR

```
dataset = load_dataset("yelp_review_full")
train_dataset, eval_dataset = dataset["train"], dataset["test"]
def trainer_init_per_worker(train_dataset, eval_dataset, **config):
    model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)
    training_args = TrainingArguments(f"{model_checkpoint}-yelp", evaluation_strategy="epoch")
    trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, eval_dataset=eval_dataset)
    return trainer
trainer = HuggingFaceTrainer(
    trainer_init_per_worker=trainer_init_per_worker,
    scaling_config=ScalingConfig(num_workers=3, use_gpu=True),
    datasets={"train": ray.data.from_huggingface(train_dataset), "evaluation": ray.data.from_huggingface(eval_dataset)},
)
result = trainer.fit()
```



training workflow, distributed with Ray AIR

**Ray Datasets  
ingest**

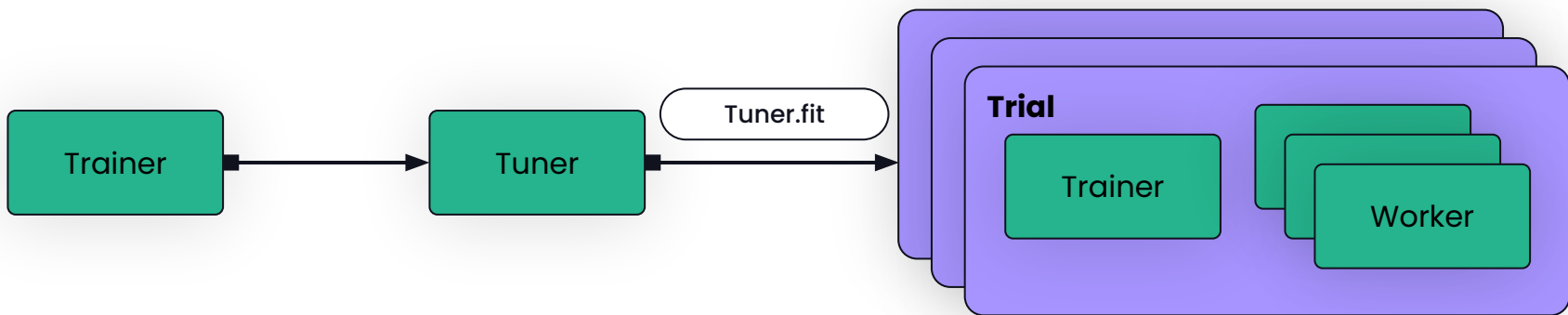
**Use existing  
 code**

**Integrate  
with the rest  
of Ray AIR**

# Hyperparameter tuning with Ray AIR

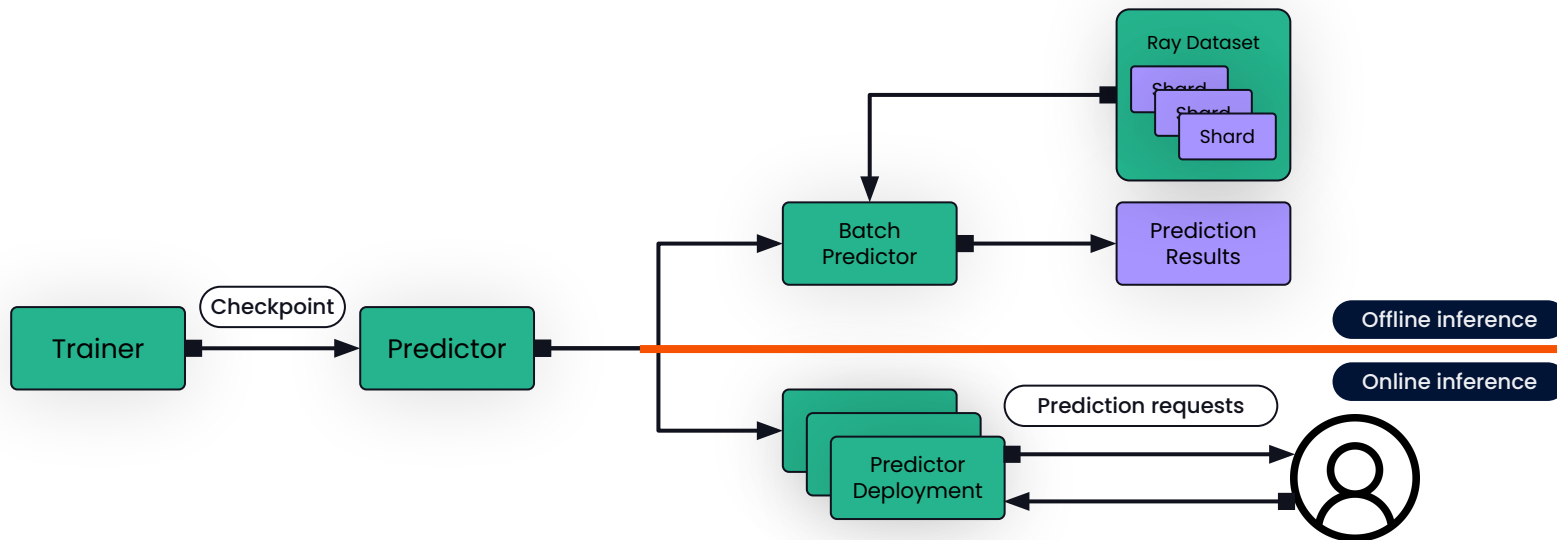
Launch a SOTA distributed hyperparam search in 2 lines of code!

```
trainer = HuggingFaceTrainer(...)  
tuner = Tuner(trainer, param_space={"batch_size": tune.grid_search([1, 2, 3])})  
results = tuner.fit()
```



# Inference & Serving with Ray AIR

- Pass the Checkpoint obtained after the end of the training to a HuggingFacePredictor for scalable offline & online inference!
- Uses 🙌 Pipelines under the hood
  - Get same output as with vanilla 😊, but in Ray Dataset format



# Inference & Serving with Ray AIR

- Pass the Checkpoint obtained after the end of the training to a HuggingFacePredictor for scalable offline & online inference!
- Uses 🙌 Pipelines under the hood
  - Get same output as with vanilla 😊, but in Ray Dataset format

```
tokenizer = AutoTokenizer.from_pretrained("sgugger/gpt2-like-tokenizer")
prompt = ["My text: Complete me..."]
predictor = BatchPredictor.from_checkpoint(
    results.checkpoint,
    HuggingFacePredictor,
    task="text-generation",
    tokenizer=tokenizer,
)
data = ray.data.from_pandas(pd.DataFrame(prompt, columns=["prompt"]))
prediction = predictor.predict(data, num_gpus_per_worker=1)
```



# Demo

hf-gpu-gpt-j-2x8-4

gptj\_deepspeed\_fine\_tuning

Metrics

+

session-sedlspnpy16naa5lm9kf2cmi2y.i.anyscaleuserdata-staging.com/vscode?folder=%2Fhome%2Fray%2Fdefault

gptj\_deepspeed\_fine\_tuning.ipynb M X

```
doc > source > ray-air > examples > gptj_deepspeed_fine_tuning.ipynb > #! pip install "datasets" "evaluate" "accelerate">=0.16.0" "transformers">=4.26.0" "torch">=1.12.0" "deepspeed"
```

base (Python 3.8.13)

## GPT-J-6B Fine-Tuning with Ray AIR and DeepSpeed

In this example, we will showcase how to use the Ray AIR for **GPT-J fine-tuning**. GPT-J is a GPT-2-like causal language model trained on the Pile dataset. This particular model has 6 billion parameters. For more information on GPT-J, click [here](#).

We will use Ray AIR (with the 🍌 Transformers integration) and a pretrained model from Hugging Face hub. Note that you can easily adapt this example to use other similar models.

This example focuses more on the performance and distributed computing aspects of Ray AIR. If you are looking for a more beginner friendly introduction to Ray AIR 🍌 Transformers integration, see `{doc}this example </ray-air/examples/huggingface_text_classification>`.

It is highly recommended to read [Ray AIR Key Concepts](#) and [Ray Data Key Concepts](#) before starting this example.

In order to run this example, make sure your Ray cluster has access to at least one GPU with 16 or more GBs of memory. The amount of memory needed will depend on the model. This notebook is being tested with 16 g4dn.4xlarge instances.

In this notebook, we will:

1. Set up Ray
2. Load the dataset
3. Preprocess the dataset with Ray AIR
4. Run the training with Ray AIR
5. Generate text from prompt with Ray AIR

Uncomment and run the following line in order to install all the necessary dependencies (this notebook is being tested with `transformers==4.26.0`):

```
1 #! pip install "datasets" "evaluate" "accelerate">=0.16.0" "transformers">=4.26.0" "torch">=1.12.0" "deepspeed"
```

Python

```
1 import numpy as np
2 import pandas as pd
3 import os
```

Python

master 0 0

Jupyter Server: local Layout: U.S.



# Try it out now!

[https://docs.ray.io/en/master/ray-air/examples/gptj\\_deepspeed\\_fine\\_tuning.html](https://docs.ray.io/en/master/ray-air/examples/gptj_deepspeed_fine_tuning.html)



# Ray Summit 2023



SPEAKERS

TRAINING

SPONSORS

WHO ATTENDS

REGISTER NOW

## THE PLACE FOR EVERYTHING RAY

San Francisco Marriott Marquis | September 18-20

AI is moving fast. Get in front of what's next at Ray Summit 2023. Join the global Ray community in San Francisco for keynotes, Ray deep dives, lightning talks and more exploring the future of machine learning and scalable AI.

REGISTRATION IS OPEN



<https://bit.ly/raysummit2023>



# Thank you!

## Q & A

Jules S. Damji, [jules@anyscale.com](mailto:jules@anyscale.com) @2twitme

Antoni Baum, [antoni@anyscale.com](mailto:antoni@anyscale.com),

