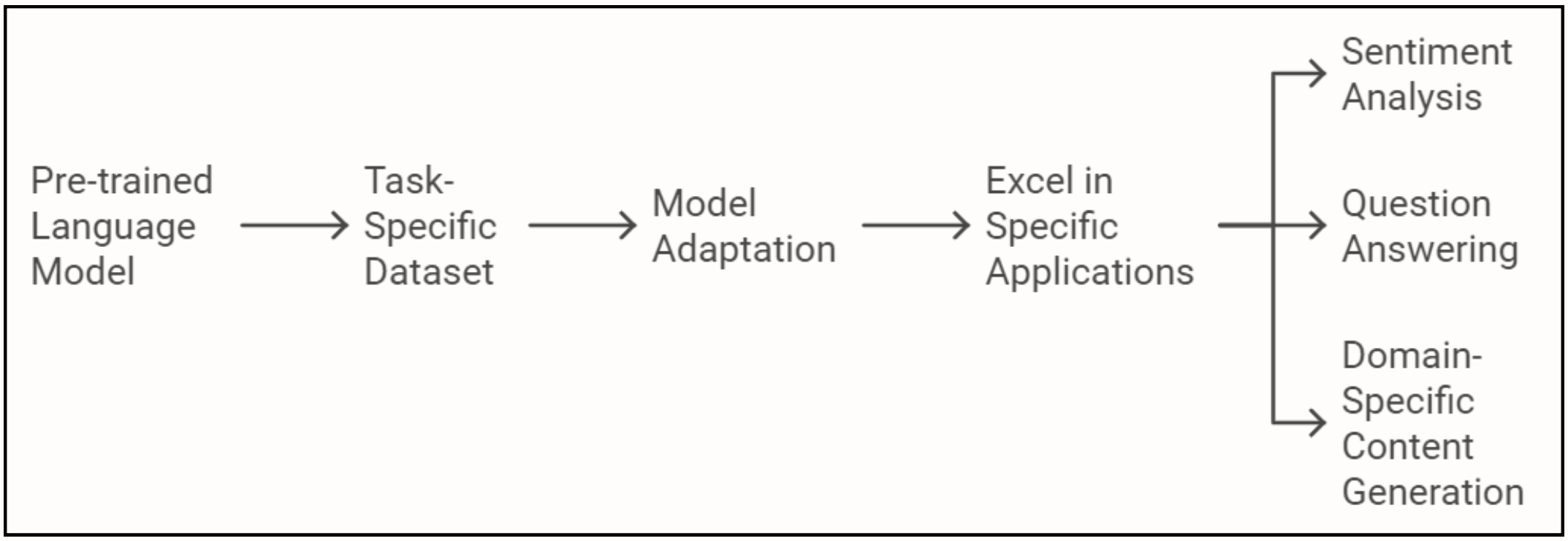
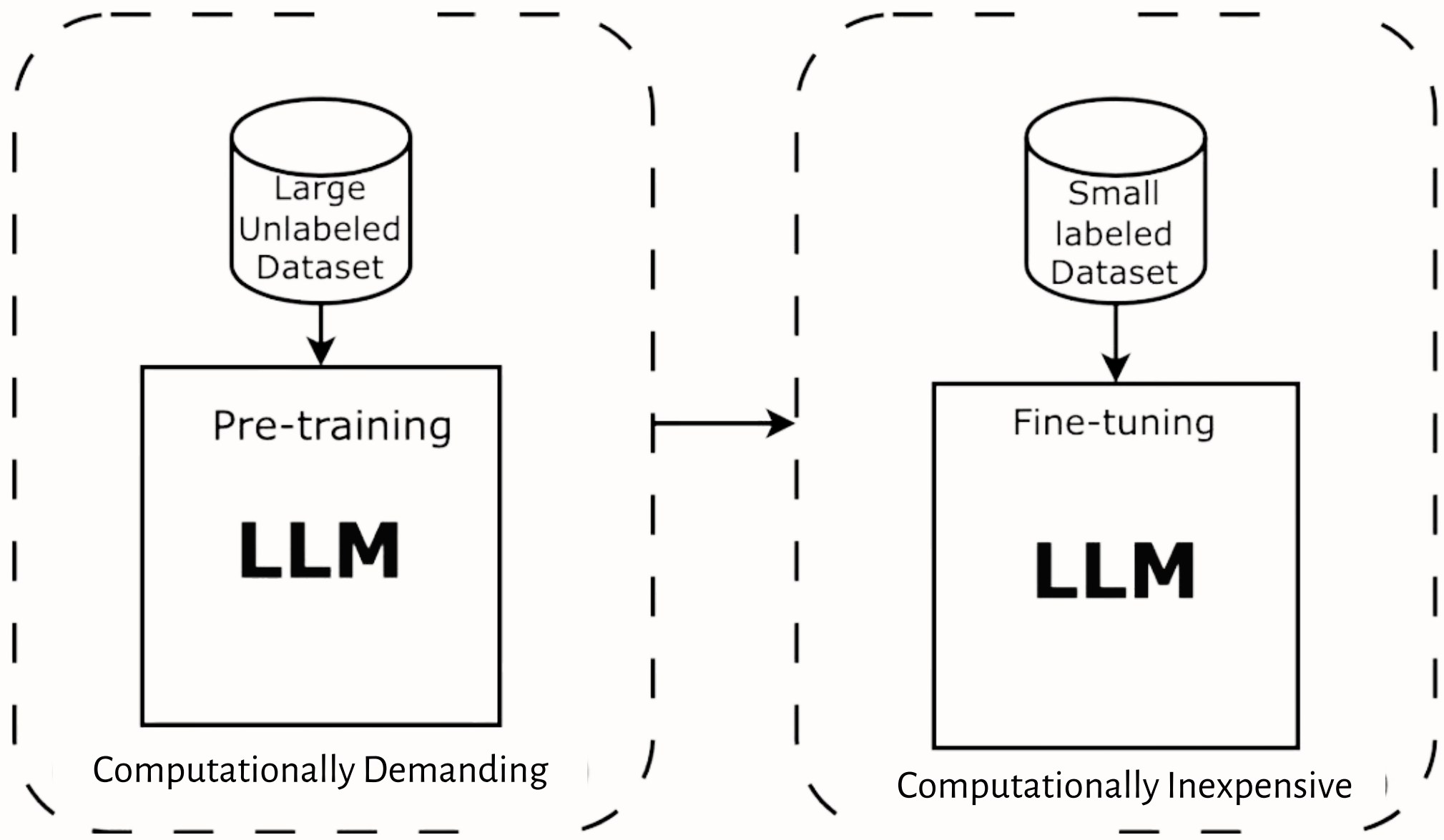
A BEGINNER'S GUIDE TO FINE-TUNING LLMS

Bhavishya Pandit

WHAT IS FINE-TUNING



Fine-tuning a large language model (LLM) means taking a pre-trained model and adjusting it using specific, smaller datasets to improve its performance for a particular task. This helps the model become more specialized while retaining its original knowledge.



Bhavishya Pandit

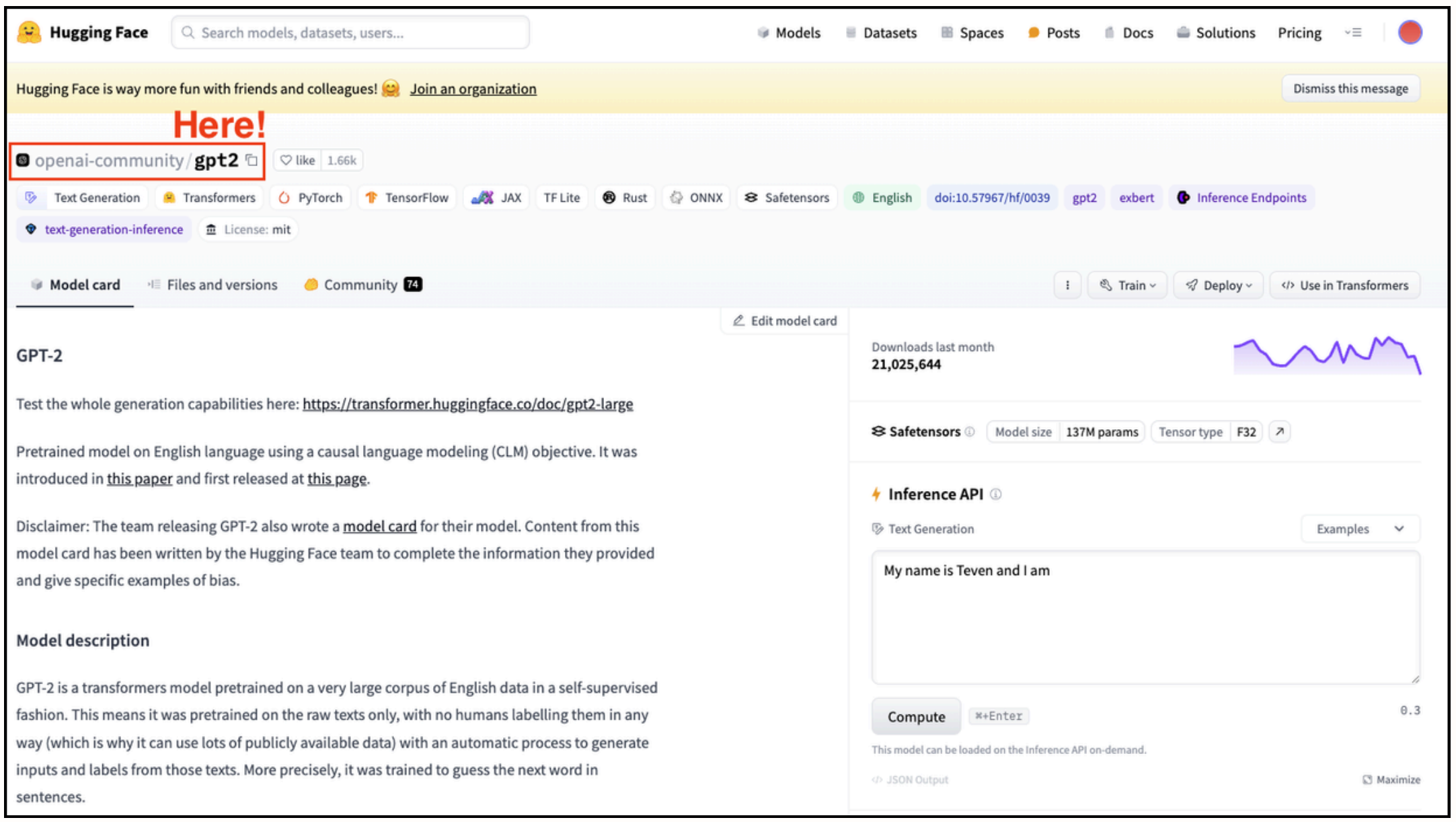
DIFFERENT TYPES OF FINE-TUNING TECHNIQUES

- **Task-Specific Fine-Tuning**: Adapting the model to perform a particular task, such as summarization using a dataset tailored for that task.
- **Domain-Specific Fine-Tuning**: Training the model on data from a specific domain like to improve its performance in those areas.
- **Few-Shot Fine-Tuning**: Fine-tuning with a small amount of task-specific data to help the model perform well even when data availability is limited.
- **Supervised Fine-Tuning**: Using labeled datasets to guide the model in generating more accurate outputs for specific tasks, making it more reliable and consistent.
- **Low-Rank Adaptation (LoRA)**: Adding smaller, trainable parameters while keeping the main model frozen, making fine-tuning more efficient.

Bhavishya Pandit

A STEP-BY-STEP GUIDE TO FINE-TUNING A LLM

Choose a pre-trained model and a dataset



Credits: DataCamp

Step 1: Install necessary packages

```
!pip install datasets
!pip install pandas
!pip install transformers
!pip install evaluate
!pip install numpy
!pip install datasets pandas transformers evaluate numpy
!pip install torch
```

Bhavishya Pandit

Step 2: Load the data to use

```
from datasets import load_dataset

dataset = load_dataset("mteb/tweet_sentiment_extraction")
df = pd.DataFrame(dataset['train'])
```

Step 3: Tokenizer

```
from transformers import GPT2Tokenizer

# Loading the dataset to train our model
dataset = load_dataset("mteb/tweet_sentiment_extraction")

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

To improve our processing requirements, we can create a smaller subset of the full dataset to fine-tune our model. The training set will be used to fine-tune our model, while the testing set will be used to evaluate it.

```
small_train_dataset =
tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
small_eval_dataset =
tokenized_datasets["test"].shuffle(seed=42).select(range(1000))
```

Step 4: Initialize our base model

```
from transformers import GPT2ForSequenceClassification

model = GPT2ForSequenceClassification.from_pretrained("gpt2", num_labels=3)
```

Step 5: Evaluate method

```
import evaluate

metric = evaluate.load("accuracy")
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

Step 6: Fine-tune using the Trainer Method

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="test_trainer",
    #evaluation_strategy="epoch",
    per_device_train_batch_size=1,  # Reduce batch size here
    per_device_eval_batch_size=1,    # Optionally, reduce for evaluation as well
    gradient_accumulation_steps=4
    )

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,

)
trainer.train()
```

After training, evaluate the model's performance on a validation or test set. Again, the trainer class already contains an evaluate method that takes care of this.

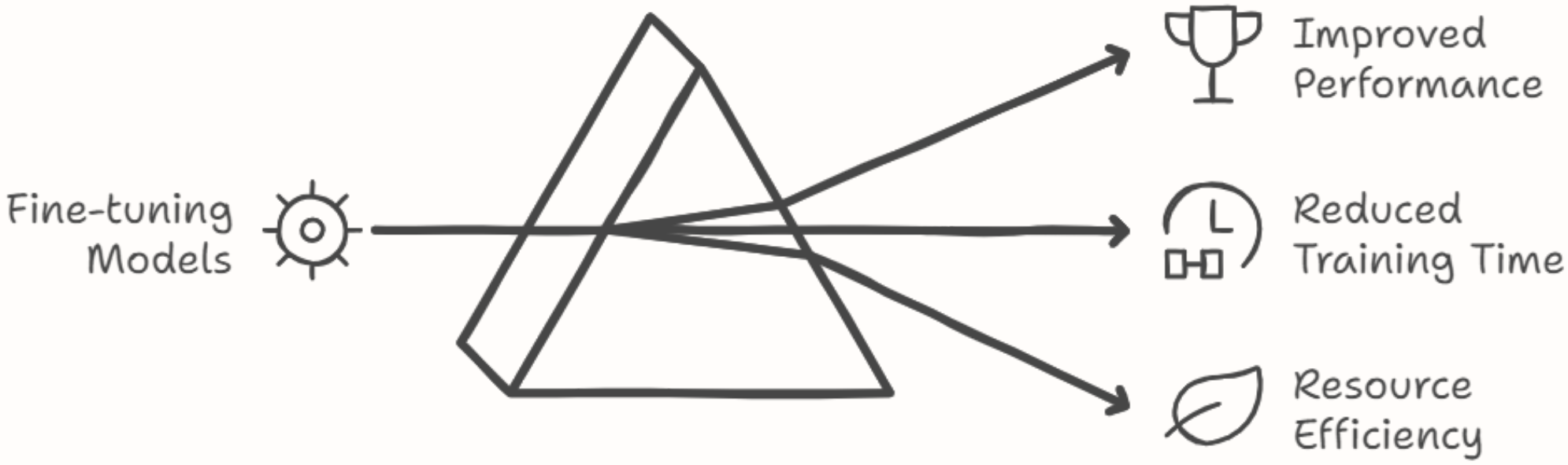
```
import evaluate


trainer.evaluate()
```

Code Credits: DataCamp

Bhavishya Pandit

BENEFITS OF FINE-TUNING



- **Improved Performance:** Fine-tuned models typically outperform their base counterparts on specific tasks, leading to higher accuracy and better user satisfaction.
- **Reduced Training Time:** Fine-tuning a pre-trained model is generally faster than training a model from scratch, as the model starts with a strong foundation of language understanding.
- **Resource Efficiency:** Fine-tuning requires less computational power and data compared to training a new model, making it a more accessible option for many organizations.

Bhavishya Pandit

Follow for more AI/ML posts

SAVE

LIKE

SHARE