

## ✓ Topic: Partial Freezing of MLMs for PoS Tagging - English

```
!pip -q install datasets transformers conllu torch
```

### ✓ Task 1: Dataset Preparation and Baseline Model Training

#### ✓ Loading Data

```
import random
import numpy as np
import torch

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

import requests, zipfile, io, os

# GitHub raw URL (direct to the ZIP)
url = "https://github.com/johnemekaeze/PoS-Tagging-MLMs-Partial-Freezing/raw/main/data/ud-treebanks-v2.16-subset.zip"

# Download and extract to a folder
response = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(response.content))
z.extractall("ud_subset") # this creates a folder with all sub-treebanks
```

```
import os

root_path = "ud_subset/ud-treebanks-v2.16-subset"
treebanks = sorted(os.listdir(root_path))
print("Available treebanks:")
treebanks
```

→ Available treebanks:

```
[ 'UD_English-Atis',
  'UD_English-CHILDES',
  'UD_English-CTeTex',
  'UD_English-ESLSpok',
  'UD_English-EWT',
  'UD_English-GENTLE',
  'UD_English-GUM',
  'UD_English-GUMReddit',
  'UD_English-LinES',
  'UD_English-ParTUT',
  'UD_Hausa-NorthernAutogramm',
  'UD_Hausa-SouthernAutogramm',
  'UD_Naija-NSC',
  'UD_Yoruba-YTB' ]
```

```
from conllu import parse_incr

def load_conllu_sentences(path):
    data = []
```

```
with open(path, "r", encoding="utf-8") as f:
    for tokenlist in parse_incr(f):
        tokens = [token["form"].lower() for token in tokenlist if type(token["id"]) == int]
        upos   = [token["upos"] for token in tokenlist if type(token["id"]) == int]
        data.append((tokens, upos))
return data

tb_name = "UD_English-EWT"
train_path = os.path.join(root_path, tb_name, "en_ewt-ud-train.conllu")
test_path  = os.path.join(root_path, tb_name, "en_ewt-ud-test.conllu")
dev_path   = os.path.join(root_path, tb_name, "en_ewt-ud-dev.conllu")

train_sentences = load_conllu_sentences(train_path)
test_sentences  = load_conllu_sentences(test_path)
dev_sentences   = load_conllu_sentences(dev_path)

# Example
print(f"{len(train_sentences)} sentences loaded.")
print("Tokens:", train_sentences[0][0])
print("UPOS  :", train_sentences[0][1])

→ 12544 sentences loaded.
Tokens: ['al', '-', 'zaman', ':', 'american', 'forces', 'killed', 'shaikh', 'abdullah', 'al', '-', 'ani', ',', 't
UPOS  : ['PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'ADJ', 'NOUN', 'VERB', 'PROPN', 'PROPN', 'PROPN', 'PUNCT', 'PROPN',
         'PUNCT', 'PROPN']

print(f"Number of training examples: {len(train_sentences)}")
print(f"Number of test examples: {len(test_sentences)}")
print(f"Number of dev examples: {len(dev_sentences)}")

→ Number of training examples: 12544
Number of test examples: 2077
Number of dev examples: 2001
```

## ✓ Tokenization

```
import random

# Set seed for reproducibility
random.seed(42)

train_subset = train_sentences #random.sample(train_sentences, int(0.2 * len(train_sentences)))
dev_subset   = dev_sentences #random.sample(dev_sentences,   int(0.2 * len(dev_sentences)))
test_subset  = test_sentences #random.sample(test_sentences, int(0.2 * len(test_sentences)))

# Get full set of UPOS tags from training split
all_tags = sorted({tag for _, tags in train_subset for tag in tags})
tag2id = {tag: i for i, tag in enumerate(all_tags)}
id2tag = {i: tag for tag, i in tag2id.items()}

from transformers import AutoTokenizer

# Load a multilingual DistilBERT tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-multilingual-cased")

def tokenize_and_align(example, label_all_tokens=False):
    tokens, labels = example
    tokenized = tokenizer(tokens,
                         is_split_into_words=True,
                         truncation=True,
```

```

        max_length=128)

word_ids = tokenized.word_ids()
aligned_labels = []
previous_word_idx = None

for word_idx in word_ids:
    if word_idx is None:
        aligned_labels.append(-100)
    elif word_idx != previous_word_idx:
        aligned_labels.append(tag2id[labels[word_idx]])
    else:
        aligned_labels.append(tag2id[labels[word_idx]] if label_all_tokens else -100)
    previous_word_idx = word_idx

tokenized["labels"] = aligned_labels
return tokenized

```

→ /usr/local/lib/python3.11/dist-packages/huggingface\_hub/utils/\_auth.py:94: UserWarning:  
The secret `HF\_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/t>)  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models or datasets.

```

warnings.warn(
tokenizer_config.json: 100%                                         49.0/49.0 [00:00<00:00, 3.13kB/s]
config.json: 100%                                         466/466 [00:00<00:00, 38.7kB/s]
vocab.txt: 100%                                         996k/996k [00:00<00:00, 4.78MB/s]
tokenizer.json: 100%                                         1.96M/1.96M [00:00<00:00, 6.89MB/s]

```

```

from datasets import Dataset

# Wrap into HuggingFace Datasets
train_dataset = Dataset.from_list([{"tokens": t, "upos": u} for t, u in train_subset])
dev_dataset = Dataset.from_list([{"tokens": t, "upos": u} for t, u in dev_subset])
test_dataset = Dataset.from_list([{"tokens": t, "upos": u} for t, u in test_subset])

# Tokenize and align
train_tok = train_dataset.map(lambda ex: tokenize_and_align((ex["tokens"], ex["upos"])),
                               batched=False)
dev_tok = dev_dataset.map(lambda ex: tokenize_and_align((ex["tokens"], ex["upos"])),
                           batched=False)
test_tok = test_dataset.map(lambda ex: tokenize_and_align((ex["tokens"], ex["upos"])),
                           batched=False)

```

→ Map: 100% 12544/12544 [00:04<00:00, 1683.18 examples/s]  
Map: 100% 2001/2001 [00:00<00:00, 3404.63 examples/s]  
Map: 100% 2077/2077 [00:00<00:00, 3347.78 examples/s]

```

print("Original Tokens:", train_subset[0][0])
print("Tokenized Version:", tokenizer.convert_ids_to_tokens(train_tok[0]["input_ids"]))
print("Labels:", [id2tag[label] if label != -100 else -100 for label in train_tok[0]["labels"]])

→ Original Tokens: ['al', '-', 'zaman', ':', 'american', 'forces', 'killed', 'shaikh', 'abdullah', 'al', '-', 'ani'
Tokenized Version: ['[CLS]', 'al', '-', 'zaman', ':', 'american', 'forces', 'killed', 'sh', '#aik', '#h', 'ab',
Labels: [-100, 'PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'ADJ', 'NOUN', 'VERB', 'PROPN', -100, -100, 'PROPN', -100, -10

```

## ✓ Fine-tuning a Distilled Model (Baseline)

```
from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
    Trainer,
)
import numpy as np
from datasets import Dataset
import warnings
warnings.filterwarnings("ignore")

# 2) Load tokenizer & model
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(tag2id),
    id2label={i: t for t, i in tag2id.items()},
    label2id=tag2id,
)
# 3) Data collator to pad and align labels
data_collator = DataCollatorForTokenClassification(tokenizer)

# 4) Define metrics function (here: token accuracy)
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=2)
    labels = p.label_ids
    # only consider non -100 labels
    mask = labels != -100
    acc = (preds[mask] == labels[mask]).astype(np.float32).mean().item()
    return {"accuracy": acc}

# 5) Training arguments
training_args = TrainingArguments(
    output_dir="./baseline_distilbert",
    eval_strategy="epoch",      # skip eval for the fastest baseline
    save_strategy="no",         # skip checkpointing
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
)
# 6) Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=dev_tok,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
# 7) Train!
trainer.train()
```

```

→ model.safetensors: 100% 542M/542M [00:10<00:00, 37.0MB/s]

Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was n
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter: .....
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: johneze (john-eze) to https://api.wandb.ai. Use `wandb login --relogin` to force r
Tracking run with wandb version 0.20.1
Run data is saved locally in /content/wandb/run-20250707_045737-8oqitinx
Syncing run /baseline\_distilbert to Weights & Biases \(docs\)
View project at https://wandb.ai/john-eze/huggingface
View run at https://wandb.ai/john-eze/huggingface/runs/8oqitinx
[3920/3920 08:22, Epoch 5/5]

```

Epoch	Training Loss	Validation Loss	Accuracy
1	0.117600	0.150702	0.956257
2	0.069300	0.143936	0.960393
3	0.037900	0.163399	0.960870
4	0.014600	0.179782	0.963733
5	0.010400	0.185904	0.963574

```
TrainOutput(global_step=3920, training_loss=0.06856364032671768, metrics={'train_runtime': 532.253,
'train_samples_per_second': 117.839, 'train_steps_per_second': 7.365, 'total_flos': 901072948530624.0,
'train_loss': 0.06856364032671768, 'epoch': 5.0})
```

```
# 8) Baseline evaluation on a small dev split
metrics = trainer.evaluate(eval_dataset=dev_tok)
print("Evaluation accuracy:", metrics["eval_accuracy"])
```

→ [251/251 00:02]  
 Dev accuracy: 0.9635741710662842

## ▼ Task 2: Model Adjustment and Partial Freezing

### ▼ Layer Freezing

```
from transformers import AutoModelForTokenClassification

def freeze_layers(model, freeze_strategy="first_k", k=2):
    """
    Freeze layers of a DistilBERT model based on the given strategy.
    """

    Args:
```

```
    model: An instance of AutoModelForTokenClassification based on DistilBERT.
    freeze_strategy: Strategy to freeze layers. Options:
        - "all_encoder": Freeze all encoder layers.
        - "first_k": Freeze the first k encoder layers.
        - "last_k": Freeze the last k encoder layers.
        - "alternating": Freeze alternating layers (even-indexed).
    k: Number of layers to freeze for "first_k" or "last_k" strategies.
```

```
    """
```

```

layers = model.distilbert.transformer.layer

if freeze_strategy == "all_encoder":
    for layer in layers:
        for param in layer.parameters():
            param.requires_grad = False

elif freeze_strategy == "first_k":
    for i, layer in enumerate(layers):
        if i < k:
            for param in layer.parameters():
                param.requires_grad = False

elif freeze_strategy == "alternating":
    for i, layer in enumerate(layers):
        if i % 2 == 0:
            for param in layer.parameters():
                param.requires_grad = False
    else:
        raise ValueError(f"Unknown freeze_strategy: {freeze_strategy}")

model.distilbert.transformer.layer

→ ModuleList(
    (0-5): 6 x TransformerBlock(
        (attention): DistilBertSdpAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    )
)
)

len(model.distilbert.transformer.layer)

→ 6

```

## ▼ Freeze all encoder layers

```

from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
    Trainer,
)
import numpy as np
from datasets import Dataset
import warnings
warnings.filterwarnings("ignore")

```

```

# 2) Load tokenizer & model
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(tag2id),
    id2label={i: t for t, i in tag2id.items()},
    label2id=tag2id,
)

# 3) Apply partial-freeze
freeze_layers(model, freeze_strategy="all_encoder")

# 4) Prepare data collator
data_collator = DataCollatorForTokenClassification(tokenizer)

# 5) Metrics function
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=2)
    labels = p.label_ids
    mask = labels != -100
    acc = (preds[mask] == labels[mask]).astype(np.float32).mean().item()
    return {"accuracy": acc}

# 6) Training arguments
training_args = TrainingArguments(
    output_dir="./frozen_first_2_distilbert",
    eval_strategy="epoch",
    save_strategy="no",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
)
# 7) Initialize Trainer with the already-frozen model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=dev_tok,           # now you can also evaluate each epoch
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
# 8) Train!
trainer.train()

```

→ Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-base. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[3920/3920 05:51, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.746900	0.662659	0.848729
2	0.395300	0.382548	0.896170
3	0.298000	0.317304	0.909015
4	0.260400	0.291540	0.914264
5	0.242000	0.284033	0.915855

TrainOutput(global\_step=3920, training\_loss=0.5731881867866127, metrics={'train\_runtime': 351.35, 'train\_samples\_per\_second': 178.511, 'train\_steps\_per\_second': 11.157, 'total\_flos': 901072948530624.0, 'train\_loss': 0.5731881867866127, 'epoch': 5.0})

# 8) Baseline evaluation on a small dev split  
metrics = trainer.evaluate(eval\_dataset=dev\_tok)  
print("Evaluation accuracy:", metrics["eval\_accuracy"])

→ [251/251 00:02]

Evaluation accuracy: 0.9158547520637512

## ▼ Freeze first 2 layers

```
from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
    Trainer,
)
import numpy as np
from datasets import Dataset
import warnings
warnings.filterwarnings("ignore")

# 2) Load tokenizer & model
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(tag2id),
    id2label={i: t for t, i in tag2id.items()},
    label2id=tag2id,
)
# 3) Apply partial-freeze
freeze_layers(model, freeze_strategy="first_k", k=2)

# 4) Prepare data collator
data_collator = DataCollatorForTokenClassification(tokenizer)

# 5) Metrics function
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=2)
    labels = p.label_ids
    mask = labels != -100
```

```

acc = (preds[mask] == labels[mask]).astype(np.float32).mean().item()
return {"accuracy": acc}

# 6) Training arguments
training_args = TrainingArguments(
    output_dir='./frozen_first_2_distilbert',
    eval_strategy="epoch",
    save_strategy="no",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
)

# 7) Initialize Trainer with the already-frozen model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=dev_tok,           # now you can also evaluate each epoch
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
# 8) Train!
trainer.train()

```

→ Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-base-uncased. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[3920/3920 07:27, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.125400	0.150468	0.955184
2	0.081300	0.142893	0.960711
3	0.046500	0.153406	0.961268
4	0.023300	0.166981	0.962858
5	0.016700	0.172846	0.962978

TrainOutput(global\_step=3920, training\_loss=0.07826632197794257, metrics={'train\_runtime': 447.5689, 'train\_samples\_per\_second': 140.135, 'train\_steps\_per\_second': 8.758, 'total\_flos': 901072948530624.0, 'train\_loss': 0.07826632197794257, 'epoch': 5.0})

```

# 8) Baseline evaluation on a small dev split
metrics = trainer.evaluate(eval_dataset=dev_tok)
print("Evaluation accuracy:", metrics["eval_accuracy"])

```

→ [251/251 00:02]  
Evaluation accuracy: 0.9629777073860168

## ▼ Freeze first 4 layers

```

from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
)

```

```

    Trainer,
)
import numpy as np
from datasets import Dataset
import warnings
warnings.filterwarnings("ignore")

# 2) Load tokenizer & model
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(tag2id),
    id2label={i: t for t, i in tag2id.items()},
    label2id=tag2id,
)
# 3) Apply partial-freeze
freeze_layers(model, freeze_strategy="first_k", k=4)

# 4) Prepare data collator
data_collator = DataCollatorForTokenClassification(tokenizer)

# 5) Metrics function
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=2)
    labels = p.label_ids
    mask = labels != -100
    acc = (preds[mask] == labels[mask]).astype(np.float32).mean().item()
    return {"accuracy": acc}

# 6) Training arguments
training_args = TrainingArguments(
    output_dir="./frozen_first_4_distilbert",
    eval_strategy="epoch",
    save_strategy="no",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
)
# 7) Initialize Trainer with the already-frozen model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=dev_tok,           # now you can also evaluate each epoch
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
# 8) Train!
trainer.train()

```

Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-base. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[3920/3920 06:39, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.136000	0.160417	0.952161
2	0.095600	0.145956	0.957172
3	0.062100	0.149088	0.959359
4	0.045600	0.150120	0.961626
5	0.036000	0.154047	0.961228

```
TrainOutput(global_step=3920, training_loss=0.10372949417452423, metrics={'train_runtime': 400.0542, 'train_samples_per_second': 156.779, 'train_steps_per_second': 9.799, 'total_flos': 901072948530624.0, 'train_loss': 0.10372949417452423, 'epoch': 5.0})
```

```
# 8) Baseline evaluation on a small dev split
dev_hf = dev_tok
metrics = trainer.evaluate(eval_dataset=dev_hf)
print("Evaluation accuracy:", metrics["eval_accuracy"])
```

[251/251 00:02]

Evaluation accuracy: 0.9612279534339905

## Alternate freezing strategy

```
from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
    Trainer,
)
import numpy as np
from datasets import Dataset
import warnings
warnings.filterwarnings("ignore")

# Load tokenizer & model
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(tag2id),
    id2label={i: t for t, i in tag2id.items()},
    label2id=tag2id,
)

# Apply partial-freeze
freeze_layers(model, freeze_strategy="alternating")

# Prepare data collator
data_collator = DataCollatorForTokenClassification(tokenizer)

# Metrics function
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=2)
    labels = p.label_ids
```

```

mask = labels != -100
acc = (preds[mask] == labels[mask]).astype(np.float32).mean().item()
return {"accuracy": acc}

# Training arguments
training_args = TrainingArguments(
    output_dir="./frozen_alt_distilbert",
    eval_strategy="epoch",
    save_strategy="no",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
)

# Initialize Trainer with the already-frozen model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=dev_tok,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
# Train
trainer.train()

```

→ Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-base-uncased. You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[3920/3920 07:03, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.124800	0.150780	0.955502
2	0.081000	0.141362	0.959439
3	0.051800	0.152201	0.960154
4	0.029200	0.163982	0.962421
5	0.019500	0.166284	0.962381

TrainOutput(global\_step=3920, training\_loss=0.08599146607579017, metrics={'train\_runtime': 423.6397, 'train\_samples\_per\_second': 148.05, 'train\_steps\_per\_second': 9.253, 'total\_flos': 901072948530624.0, 'train\_loss': 0.08599146607579017, 'epoch': 5.0})

# 8) Baseline evaluation on a small dev split  
dev\_hf = dev\_tok  
metrics = trainer.evaluate(eval\_dataset=dev\_hf)  
print("Evaluation accuracy:", metrics["eval\_accuracy"])

→ [251/251 00:02]

Evaluation accuracy: 0.9623811841011047

## ▼ Task 3: Analysis and Comparison

### ▼ Analysis of Parameters

```

import torch
from transformers import AutoModelForTokenClassification
import pandas as pd

# Define freeze function
def freeze_layers(model, strategy, k=0):
    # Unfreeze all first
    for param in model.parameters():
        param.requires_grad = True
    layers = model.distilbert.transformer.layer
    if strategy == "all_encoder":
        for layer in layers:
            for param in layer.parameters():
                param.requires_grad = False
    elif strategy == "first_k":
        for i, layer in enumerate(layers):
            if i < k:
                for param in layer.parameters():
                    param.requires_grad = False
    elif strategy == "alternating":
        for i, layer in enumerate(layers):
            if i % 2 == 0:
                for param in layer.parameters():
                    param.requires_grad = False

# Load base model name
model_name = "distilbert-base-multilingual-cased"
strategies = [
    ("Baseline", None, 0),
    ("Freeze All", "all_encoder", 0),
    ("Freeze First 2", "first_k", 2),
    ("Freeze First 4", "first_k", 4),
    ("Alternating Freeze", "alternating", 0),
]
results = []

for name, strat, k in strategies:
    model = AutoModelForTokenClassification.from_pretrained(
        model_name, num_labels=10 # dummy num_labels
    )
    if strat:
        freeze_layers(model, strat, k)
    total_params = sum(p.numel() for p in model.parameters())
    trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    results.append({
        "Strategy": name,
        "Total Params": total_params,
        "Trainable Params": trainable_params,
        "Trainable (%)": trainable_params / total_params * 100
    })
df_params = pd.DataFrame(results)

# Display the DataFrame to the user
display(df_params)

```

```
→ /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:  
The secret `HF_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/t)  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models or datasets.  
    warnings.warn(  
config.json: 100%                                         466/466 [00:00<00:00, 21.9kB/s]  
  
model.safetensors: 100%                                     542M/542M [00:11<00:00, 83.3MB/s]  
  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert-bas  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

	Strategy	Total Params	Trainable Params	Trainable (%)	
0	Baseline	134741770	134741770	100.000000	
1	Freeze All	134741770	92214538	68.437974	
2	Freeze First 2	134741770	120566026	89.479325	
3	Freeze First 4	134741770	106390282	78.958650	
4	Alternating Freeze	134741770	113478154	84.218987	

Next steps: [Generate code with df\\_params](#) [View recommended plots](#) [New interactive sheet](#)

## Model Performance Across Freezing Strategies

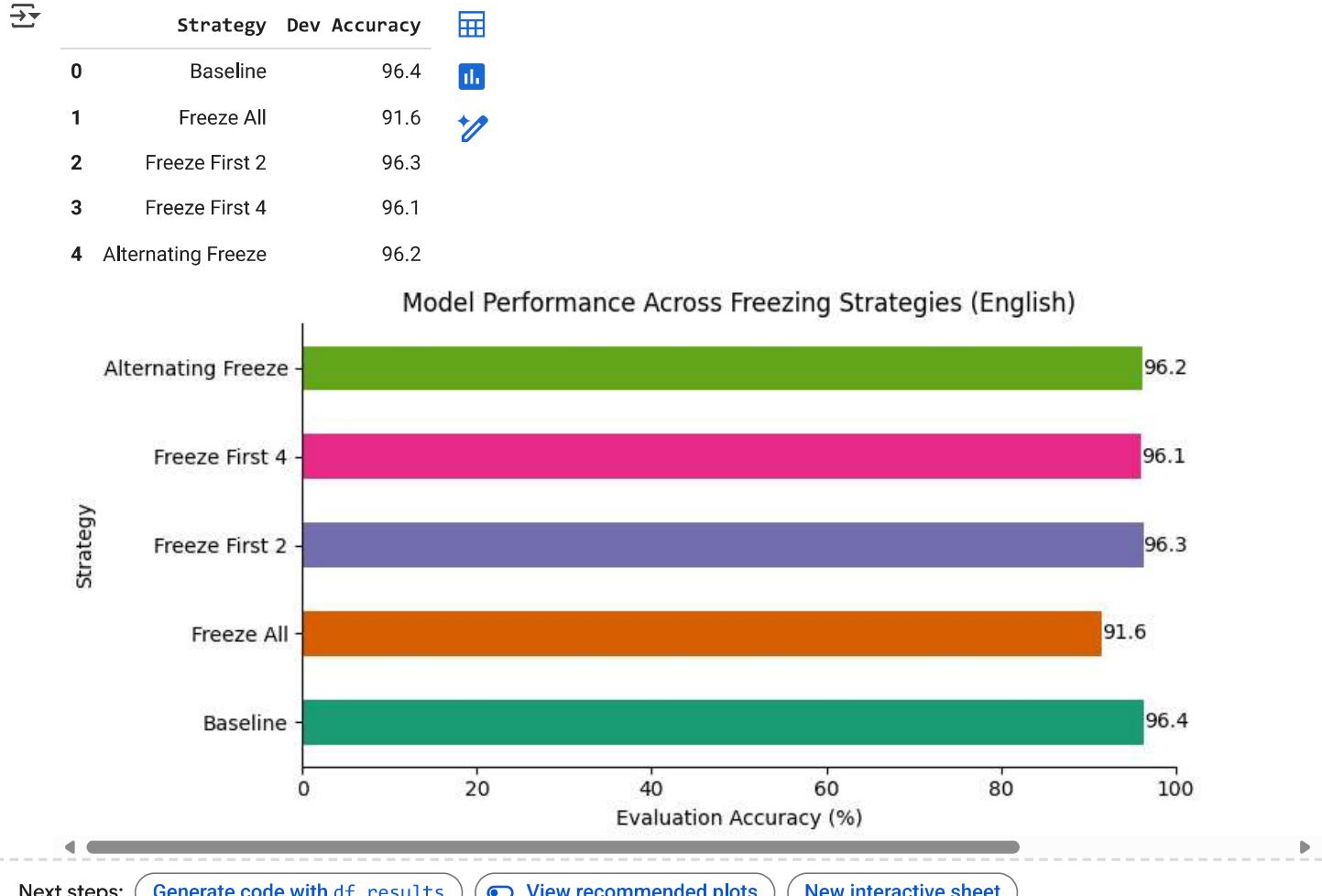
```
from matplotlib import pyplot as plt  
import seaborn as sns  
import pandas as pd  
  
# Example results - replace with your actual evaluation metrics  
data = {  
    "Strategy": ["Baseline", "Freeze All", "Freeze First 2", "Freeze First 4", "Alternating Freeze"],  
    "Dev Accuracy": [96.4, 91.6, 96.3, 96.1, 96.2]  
}  
df_results = pd.DataFrame(data)  
  
# Display the DataFrame to the user  
display(df_results)  
  
# Create horizontal bar chart with Dark2 palette  
palette = sns.color_palette("Dark2", n_colors=len(df_results))  
plt.figure(figsize=(8, 4))  
ax = df_results.set_index('Strategy')['Dev Accuracy'].plot(  
    kind='barh',  
    color=palette  
)  
  
# Annotate each bar with its accuracy value  
for i, (strategy, accuracy) in enumerate(zip(df_results['Strategy'], df_results['Dev Accuracy'])):  
    ax.text(accuracy + 0.005, i, f"{accuracy:.1f}", va='center')  
  
# Remove top and right spines
```

```

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.xlabel("Evaluation Accuracy (%)")
plt.title("Model Performance Across Freezing Strategies (English)")
plt.xlim(0, 100.0) # ensure space for labels on the right
plt.tight_layout()
plt.show()

```



## Epoch Convergence Curve

```

import matplotlib.pyplot as plt

# Example: histories is a dict mapping strategy → list of accuracies per epoch
histories = {
    "Baseline": [95.6, 96.0, 96.1, 96.4, 96.4],
    "Freeze All": [84.9, 89.6, 90.9, 91.4, 91.6],
    "Freeze First 2": [95.5, 96.1, 96.1, 96.3, 96.3],
    "Freeze First 4": [95.2, 95.7, 95.9, 96.2, 96.1],
    "Alternating Freeze": [95.6, 95.9, 96.0, 96.2, 96.2],
}

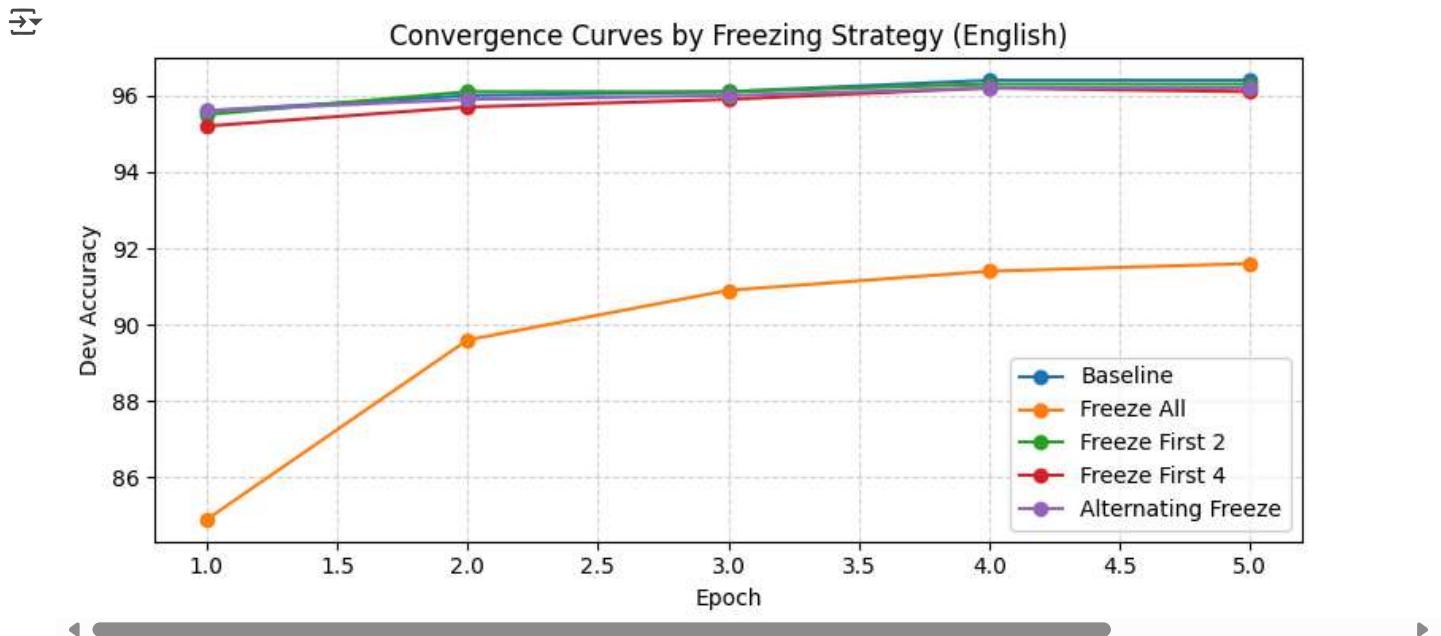
plt.figure(figsize=(8,4))
for strat, accs in histories.items():
    plt.plot(range(1, len(accs)+1), accs, marker='o', label=strat)
plt.xlabel("Epoch")
plt.ylabel("Dev Accuracy")

```

```

plt.title("Convergence Curves by Freezing Strategy (English)")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

```



## ▼ Accuracy vs Model Size Trade-off

```

import matplotlib.pyplot as plt
import seaborn as sns

# Example data – replace with your actual numbers
param_counts = {
    "Baseline": 135,
    "Freeze All": 92,
    "Freeze First 2": 121,
    "Freeze First 4": 106,
    "Alternating Freeze": 113,
}
accuracies = {
    "Baseline": 96.4,
    "Freeze All": 91.6,
    "Freeze First 2": 96.3,
    "Freeze First 4": 96.1,
    "Alternating Freeze": 96.2,
}

# Create a consistent color palette
palette = sns.color_palette("Dark2", n_colors=len(param_counts))

plt.figure(figsize=(8, 4))
for (strategy, count), color in zip(param_counts.items(), palette):
    plt.scatter(count, accuracies[strategy], s=100, color=color, label=strategy)

plt.xlabel("Trainable Parameters (Millions)")
plt.ylabel("Eval Accuracy (%)")
plt.title("Accuracy vs. Model Size by Freezing Strategy (English)")
plt.grid(True, linestyle='--', alpha=0.5)

# Add legend instead of inline labels

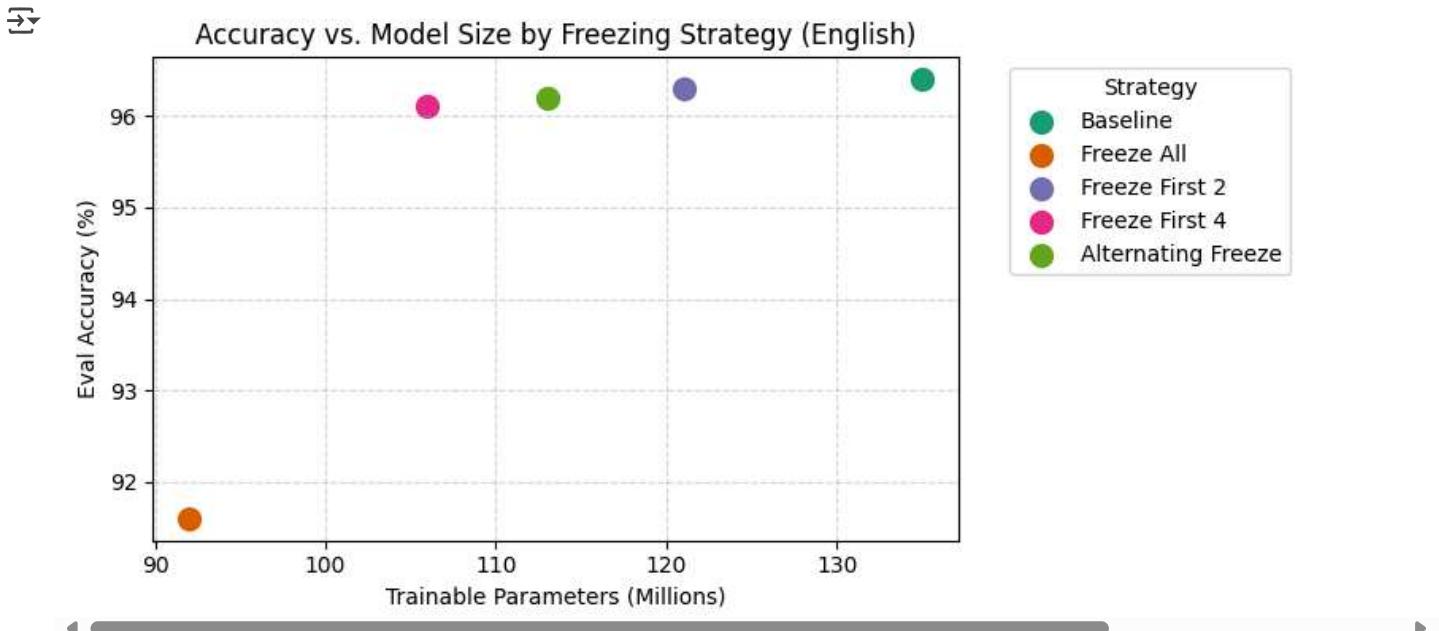
```

```

plt.legend(title="Strategy", bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()

```



## ▼ Training Time Saving

```

from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd

# Example training times - replace with your actual timing metrics
data = {
    "Strategy": ["Baseline", "Freeze All", "Freeze First 2", "Freeze First 4", "Alternating Freeze"],
    "Training Time (seconds)": [532, 351, 447, 400, 423]
}
df_times = pd.DataFrame(data)

# Display the DataFrame to the user
display(df_times)

# Create horizontal bar chart with Dark2 palette
palette = sns.color_palette("Dark2", n_colors=len(df_times))
plt.figure(figsize=(8, 4))
ax = df_times.set_index('Strategy')['Training Time (seconds)'].plot(
    kind='barh',
    color=palette
)

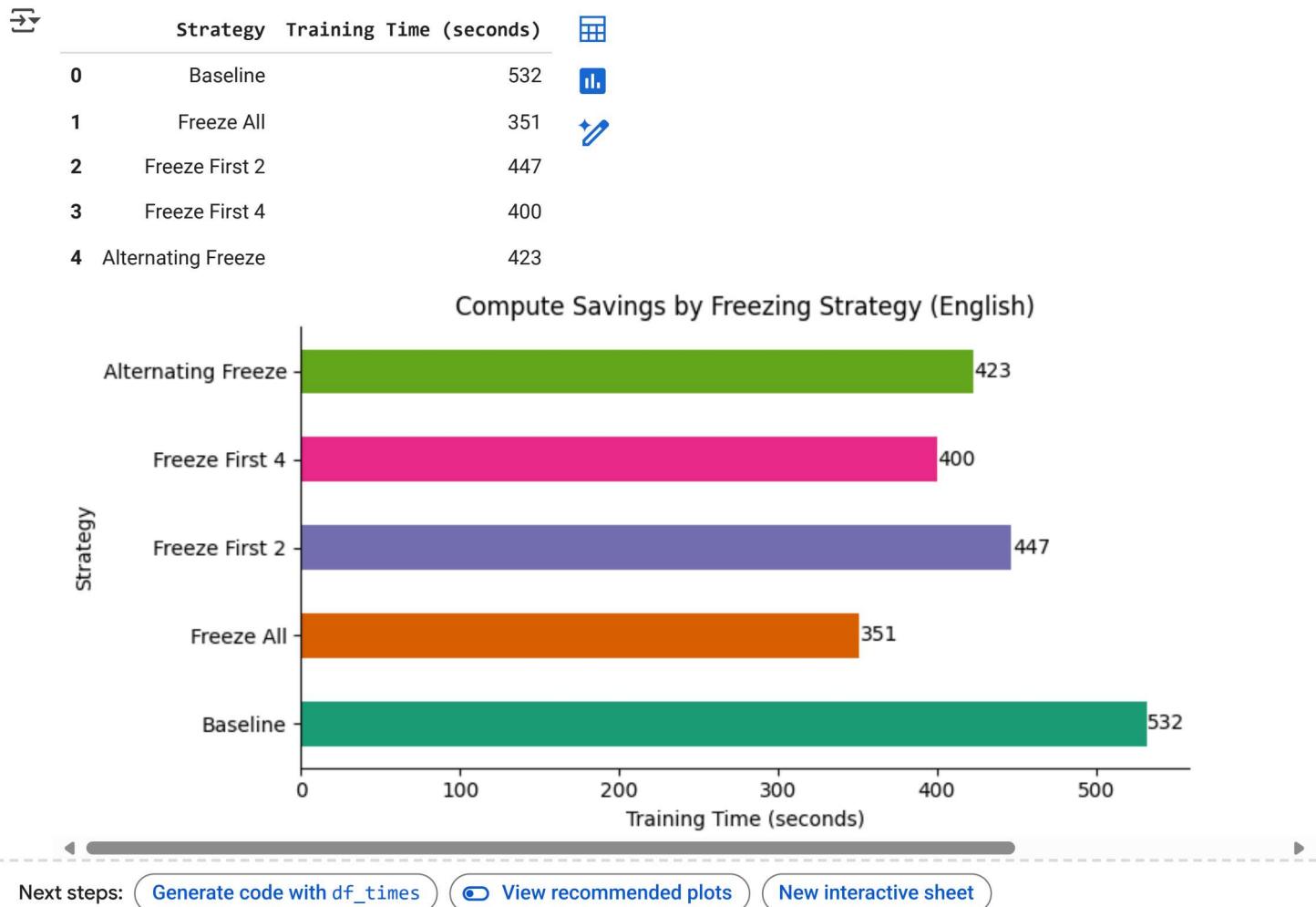
# Annotate each bar with its training time value
for i, (strategy, time_val) in enumerate(zip(df_times['Strategy'], df_times['Training Time (seconds)'])):
    ax.text(time_val + 0.5, i, f"{time_val:.0f}", va='center')

# Remove top and right spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.xlabel("Training Time (seconds)")
plt.title("Compute Savings by Freezing Strategy (English)")

```

```
plt.tight_layout()  
plt.show()
```



## Result Summary

```
import pandas as pd  
  
# 1) Raw experiment metrics  
data = {  
    "Strategy": [  
        "Baseline",  
        "Freeze All",  
        "Freeze First 2",  
        "Freeze First 4",  
        "Alternating Freeze"  
    ],  
    "Dev Accuracy (%)": [  
        96.4, # Baseline  
        91.6, # Freeze All  
        96.3, # Freeze First 2  
        96.1, # Freeze First 4  
        96.2 # Alternating Freeze  
    ],  
    "Trainable Params (M)": [  
        134, # Baseline  
        92, # Freeze All  
        120, # Freeze First 2  
        106, # Freeze First 4  
        113 # Alternating Freeze
```

```
],
    "Training Time (s)": [
        532, # Baseline
        351, # Freeze All
        447, # Freeze First 2
        400, # Freeze First 4
        423 # Alternating Freeze
    ]
}

# 2) Build DataFrame
df = pd.DataFrame(data)

# 3) Compute savings (%) relative to baseline time
baseline_time = df.loc[df["Strategy"] == "Baseline", "Training Time (s)"].iloc[0]
df["Compute Savings (%)"] = (
    (baseline_time - df["Training Time (s)"]) / baseline_time * 100
).round(1)

# 4) Rearrange columns for readability
df = df[[
    "Strategy",
    "Dev Accuracy (%)",
    "Trainable Params (M)",
    "Training Time (s)",
    "Compute Savings (%)"
]]

```