

- Read
  - Write
  - Glob
  - Grep
- 

# DSPy BootstrapFinetune Optimizer

## Goal

Distill a DSPy program into fine-tuned model weights for efficient production deployment.

## When to Use

- You have a working DSPy program with a large model
- Need to reduce inference costs
- Want faster responses (smaller model)
- Deploying to resource-constrained environments

## Inputs

Input	Type	Description
<code>program</code>	<code>dspy.Module</code>	Teacher program to distill
<code>trainset</code>	<code>list[dspy.Example]</code>	Training examples
<code>metric</code>	<code>callable</code>	Validation metric (optional)
<code>train_kwargs</code>	<code>dict</code>	Training hyperparameters

## Outputs

Output	Type	Description
<code>finetuned_program</code>	<code>dspy.Module</code>	Program with fine-tuned weights
<code>model_path</code>	<code>str</code>	Path to saved model

## Workflow

### Phase 1: Prepare Teacher Program

```
import dspy

# Configure with strong teacher model
dspy.configure(lm=dspy.LM("openai/gpt-4o"))

class TeacherQA(dspy.Module):
    def __init__(self):
        self.cot = dspy.ChainOfThought("question -> answer")

    def forward(self, question):
        return self.cot(question=question)
```

### Phase 2: Generate Training Traces

BootstrapFinetune automatically generates traces from the teacher:

```
optimizer = dspy.BootstrapFinetune(
    metric=lambda gold, pred, trace=None: gold.answer.lower() in pred.answer.lower()
)
```

### Phase 3: Fine-tune Student Model

```
finetuned = optimizer.compile(
    TeacherQA(),
    trainset=trainset,
    train_kwarg={

        'learning_rate': 5e-5,
        'num_train_epochs': 3,
        'per_device_train_batch_size': 4,
        'warmup_ratio': 0.1
    }
)
```

### Phase 4: Deploy

```
# Save the fine-tuned model
finetuned.save("finetuned_qa_model")

# Load and use
loaded = TeacherQA()
loaded.load("finetuned_qa_model")
result = loaded(question="What is machine learning?")
```

## Production Example

---

```

import dspy
from dspy.evaluate import Evaluate
import logging
import os

logger = logging.getLogger(__name__)

class ClassificationSignature(dspy.Signature):
    """Classify text into categories."""
    text: str = dspy.InputField()
    label: str = dspy.OutputField(desc="Category: positive, negative, neutral")

class TextClassifier(dspy.Module):
    def __init__(self):
        self.classify = dspy.Predict(ClassificationSignature)

    def forward(self, text):
        return self.classify(text=text)

def classification_metric(gold, pred, trace=None):
    """Exact label match."""
    gold_label = gold.label.lower().strip()
    pred_label = pred.label.lower().strip() if pred.label else ""
    return gold_label == pred_label

def finetune_classifier(trainset, devset, output_dir="."):
    """Full fine-tuning pipeline."""

    # Configure teacher (strong model)
    dspy.configure(lm=dspy.LM("openai/gpt-4o"))

    teacher = TextClassifier()

    # Evaluate teacher
    evaluator = Evaluate(devset=devset, metric=classification_metric, num_threads=8)
    teacher_score = evaluator(teacher)
    logger.info(f"Teacher score: {teacher_score:.2%}")

    # Fine-tune
    optimizer = dspy.BootstrapFinetune(
        metric=classification_metric
    )

    finetuned = optimizer.compile(
        teacher,
        trainset=trainset,
        train_kwargs={
            'learning_rate': 2e-5,
            'num_train_epochs': 3,
            'per_device_train_batch_size': 8,
            'gradient_accumulation_steps': 2,
            'warmup_ratio': 0.1,
            'weight_decay': 0.01,
            'logging_steps': 10,
            'save_strategy': 'epoch',
            'output_dir': output_dir
        }
    )

    # Evaluate fine-tuned model
    student_score = evaluator(finetuned)
    logger.info(f"Student score: {student_score:.2%}")

```

```

# Save
finetuned.save(os.path.join(output_dir, "final_model"))

return {
    "teacher_score": teacher_score,
    "student_score": student_score,
    "model_path": output_dir
}

# For RAG fine-tuning
class RAGClassifier(dspy.Module):
    """RAG pipeline that can be fine-tuned."""

    def __init__(self, num_passages=3):
        self.retrieve = dspy.Retrieve(k=num_passages)
        self.classify = dspy.ChainOfThought("context, text -> label")

    def forward(self, text):
        context = self.retrieve(text).passages
        return self.classify(context=context, text=text)

def finetune_rag_classifier(trainset, devset):
    """Fine-tune a RAG-based classifier."""

    # Configure retriever and LM
    colbert = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')
    dspy.configure(
        lm=dspy.LM("openai/gpt-4o"),
        rm=colbert
    )

    rag = RAGClassifier()

    optimizer = dspy.BootstrapFinetune(
        metric=classification_metric
    )

    finetuned = optimizer.compile(
        rag,
        trainset=trainset,
        train_kwargs={
            'learning_rate': 1e-5,
            'num_train_epochs': 5
        }
    )

    return finetuned

```

## Training Arguments Reference

---

Argument	Description	Typical Value
learning_rate	Learning rate	1e-5 to 5e-5
num_train_epochs	Training epochs	3-5
per_device_train_batch_size	Batch size	4-16
gradient_accumulation_steps	Gradient accumulation	2-8
warmup_ratio	Warmup proportion	0.1
weight_decay	L2 regularization	0.01
max_grad_norm	Gradient clipping	1.0

## Best Practices

---

1. **Strong teacher** - Use GPT-4 or Claude as teacher
2. **Quality data** - Teacher traces are only as good as training examples
3. **Validate improvement** - Compare student to teacher on held-out set
4. **Start with more epochs** - Fine-tuning often needs 3-5 epochs
5. **Monitor overfitting** - Track validation loss during training

## Limitations

---

- Requires access to model weights (not API-only models)
- Training requires GPU resources
- Student may not match teacher quality on all inputs
- Fine-tuning takes hours/days depending on data size
- Model size reduction may cause capability loss