

- Read
- Write
- Glob
- Grep

# DSPy MIPROv2 Optimizer

## Goal

Jointly optimize instructions and few-shot demonstrations using Bayesian Optimization for maximum performance.

## When to Use

- You have **200+ training examples**
- You can afford longer optimization runs (40+ trials)
- You need state-of-the-art performance
- Both instructions and demos need tuning

## Inputs

Input	Type	Description
<code>program</code>	<code>dspy.Module</code>	Program to optimize
<code>trainset</code>	<code>list[dspy.Example]</code>	200+ training examples
<code>metric</code>	<code>callable</code>	Evaluation function
<code>auto</code>	<code>str</code>	“light”, “medium”, or “heavy”
<code>num_trials</code>	<code>int</code>	Optimization trials (40+)

## Outputs

Output	Type	Description
<code>compiled_program</code>	<code>dspy.Module</code>	Fully optimized program

## Workflow

### Three-Stage Process

1. **Bootstrap** - Generate candidate demonstrations

2. **Propose** - Create grounded instruction candidates
3. **Search** - Bayesian optimization over combinations

## Phase 1: Setup

```
import dspy

dspy.configure(lm=dspy.LM("openai/gpt-4o-mini"))
```

## Phase 2: Define Program

```
class RAGAgent(dspy.Module):
    def __init__(self):
        self.retrieve = dspy.Retrieve(k=3)
        self.generate = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = self.retrieve(question).passages
        return self.generate(context=context, question=question)
```

## Phase 3: Optimize

```
optimizer = dspy.MIPROv2(
    metric=dspy.evaluate.answer_exact_match,
    auto="medium", # Balanced optimization
    num_threads=24
)

compiled = optimizer.compile(RAGAgent(), trainset=trainset)
```

## Auto Presets

Preset	Trials	Use Case
"light"	~10	Quick iteration
"medium"	~40	Production optimization
"heavy"	~100+	Maximum performance

## Production Example

---

```

import dspy
from dspy.evaluate import Evaluate
import json
import logging

logger = logging.getLogger(__name__)

class ReActAgent(dspy.Module):
    def __init__(self, tools):
        self.react = dspy.ReAct("question -> answer", tools=tools)

    def forward(self, question):
        return self.react(question=question)

def search_tool(query: str) -> list[str]:
    """Search knowledge base."""
    results = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')(query,
k=3)
    return [r['text'] for r in results]

def optimize_agent(trainset, devset):
    """Full MIPROv2 optimization pipeline."""

    agent = ReActAgent(tools=[search_tool])

    # Baseline evaluation
    evaluator = Evaluate(
        devset=devset,
        metric=dspy.evaluate.answer_exact_match,
        num_threads=8
    )
    baseline = evaluator(agent)
    logger.info(f"Baseline: {baseline:.2%}")

    # MIPROv2 optimization
    optimizer = dspy.MIPROv2(
        metric=dspy.evaluate.answer_exact_match,
        auto="medium",
        num_threads=24,
        # Custom settings
        num_candidates=15,
        max_bootstrapped_demos=4,
        max_labeled_demos=8
    )

    compiled = optimizer.compile(agent, trainset=trainset)
    optimized = evaluator(compiled)
    logger.info(f"Optimized: {optimized:.2%}")

    # Save with metadata
    compiled.save("agent_mipro.json")

    metadata = {
        "baseline_score": baseline,
        "optimized_score": optimized,
        "improvement": optimized - baseline,
        "num_train": len(trainset),
        "num_dev": len(devset)
    }

    with open("optimization_metadata.json", "w") as f:
        json.dump(metadata, f, indent=2)

```

```
return compiled, metadata
```

## Instruction-Only Mode

---

```
# Disable demos for pure instruction optimization
optimizer = dspy.MIPROv2(
    metric=metric,
    auto="medium",
    max_bootstrapped_demos=0,
    max_labeled_demos=0
)
```

## Best Practices

---

1. **Data quantity matters** - 200+ examples for best results
2. **Use auto presets** - Start with “medium”, adjust based on results
3. **Parallel threads** - Use `num_threads=24` or higher if available
4. **Monitor costs** - Track API usage during optimization
5. **Save intermediate** - Bayesian search saves progress

## Limitations

---

- High computational cost (many LLM calls)
- Requires substantial training data
- Optimization time: hours for “heavy” preset
- Memory intensive for large candidate sets