Scaling LLM Evaluation with DSPy

Abstract

As LLM adoption moves from "chatbot" prototypes to production pipelines, the brittleness of manual prompt engineering becomes a critical bottleneck. This whitepaper details our transition from "vibe-based" evaluation to a rigorous, programmatic optimization loop using **DSPy**. We present two diverse case studies: **Structured Clinical Extraction** (achieving 99% accuracy) and **Multi-Rubric Essay Grading** (achieving 0.92 human-alignment correlation). By implementing **Assertion-Driven Feedback** and leveraging the `MIPROv2` optimizer, we demonstrate how to treat prompts as optimized programs rather than static strings.

---

1. The Problem: The "Prompt Engineering" Treadmill

In legacy architectures, engineers are trapped in a loop of manual tuning.

- **The Workflow:** Write a prompt, test on 5 examples, see a failure, tweak the prompt (e.g., "IMPORTANT: output valid JSON only!"), and re-test.
- **The Result:** 2,000-token "mega-prompts" that function as "Black Boxes"—brittle, costly, and impossible to debug.
- **The Metric:** "Vibes." If the output looks good to the engineer, it ships.

We hit a wall when scaling to high-stakes demands: 100,000+ clinical documents and real-time student assessment. The error rates were unacceptable, and manual fixes introduced regressions.

2. Case Study A: Structured Clinical Extraction

Our first challenge was extracting structured patient vitals and medication histories from unstructured clinical notes.

2.1 The Challenge: Reliability at Scale

LLMs struggle with strict schema adherence. A prompt asking for JSON often leaks commentary ("Here is the JSON: ...") or hallucinates fields.

2.2 The DSPy Solution

We defined a typed signature to strictly enforce the I/O contract:

```
class ClinicalExtraction(dspy.Signature):
    """Extract patient vitals and medication history from clinical notes."""

    clinical_note = dspy.InputField(desc="The doctor's unstructured notes.")
    patient_vitals = dspy.OutputField(desc="JSON object containing BP, HR, Temp.")
    medications = dspy.OutputField(desc="List of medications with dosage and
frequency.")
```

2.3 Assertions as Guardrails

To reach 99% accuracy, we implemented **DSPy Assertions** to enforce constraints at runtime:

1. **Format Check:** `dspy.Assert(is_valid_json(output), "Output must be valid JSON")`
2. **Schema Validation:** `dspy.Assert(validate_schema(output), "Missing required fields: BP, HR")`
3. **Hallucination Check:** `dspy.Suggest(extract in clinical_note, "Extracted value must appear in source text")`

Result: Zero-shot pass rates jumped from 68% to 92% immediately. Using the `MIPROv2` optimizer to tune the instructions pushed this to **99.1%**.

---

3. Case Study B: Grading with Multi-Rubric Standards

Our second challenge was automating the grading of undergraduate history essays based on a complex 5-point rubric (Thesis, Evidence, Analysis, Structure, Mechanics).

3.1 The Challenge: Subjectivity and Grade Inflation

LLMs are inherently "nice." Without strict grounding, they tend to give scores of 4/5 or 5/5 ("Grade Inflation"). They also struggle to justify *why* a grade was given, often providing generic feedback.

3.2 The DSPy Solution: Chain of Thought + Rubric Assertions

We modeled the grading process not as a single number generation, but as a reasoned debate.

```
class AssessRubric(dspy.Signature):
    """Assess a specific rubric criterion for an essay."""

    essay_text = dspy.InputField()
    rubric_criterion = dspy.InputField(desc="e.g. 'Evidence: Usage of primary
sources'")
    max_score = dspy.InputField(desc="e.g. 5")

    reasoning = dspy.OutputField(desc="Chain of thought explaining the score.")
    cited_evidence = dspy.OutputField(desc="Direct quotes from text supporting the
score.")
    score = dspy.OutputField(desc="Integer score.")
```

3.3 Semantic Assertions

We used assertions to force the model to prove its work *before* assigning a score.

- **Evidence Citation:** `dspy.Assert(len(cited_evidence) > 0, "You must quote the essay key points to justify the grade.")`
- **Logic Consistency:** `dspy.Suggest(score <= 3 if "no primary sources" in reasoning, "If no primary sources are found, score for Evidence must be 3 or lower.")`

This "suggest" assertion acts as a soft constraint, nudging the model to align its score with its own reasoning reasoning.

3.4 Optimization with Human Alignment Metric

We created a "Golden Set" of 50 essays evaluated by lead professors. We defined a custom DSPy metric: **RubricAlignment**.

```
def rubric_alignment_metric(example, pred, trace=None):
    # Calculate Root Mean Square Error between AI score and Professor score
    score_diff = abs(example.gold_score - pred.score)
    # Check if reasoning mentions key flaws identified by Professor
    reasoning_match = fuzz.partial_ratio(example.gold_feedback, pred.reasoning) >
80

    return (score_diff == 0) and reasoning_match
```

Using `dspy.MIPROv2` against this metric allowed the system to learn the *nuance* of the professors' grading style, effectively "fine-tuning" the prompts to be stricter or more lenient to match the human baseline.

---

4. The Optimization Loop: Replacing Human Tuning

The common thread in both case studies is **MIPROv2 (Many-Instruction PRompt Optimizer)**.

Instead of manually tweaking prompts, we compile the program:

```
optimizer = dspy.MIPROv2(metric=custom_metric)
optimized_program = optimizer.compile(
    student=module,
    trainset=golden_dataset,
    max_bootstrapped_demos=5,
    max_labeled_demos=5
)
```

What the Optimizer Does:

1. **Instruction Search:** Generates 50+ variations of "persona" (e.g., "Act as a harsh critic", "Act as a supportive tutor").
2. **Few-Shot Selection:** Selects the mathematically optimal set of examples that differentiate edge cases (e.g., distinguishing a "3/5" from a "4/5").
3. **Ensemble Scoring:** Evaluates combinations to maximize the custom metric.

5. Results & Impact

Clinical Extraction

Metric	Manual Prompting	DSPy + MIPROv2
Accuracy	85% (Unstable)	**99.1%**
Token Cost	$4.50 / 1k docs	**$2.10**

Essay Grading

Metric	GPT-4 (Vanilla)	DSPy Optimized
Human Correlation	0.65 (High Variance)	**0.92 (Strong)**
Feedback Specificity	Generic ("Good job!")	**Grounded (Cites specific lines)**
Grade Inflation	+1.5 points on avg	**+0.1 points (Aligned)**

6. Conclusion

Scaling LLM evaluation requires abandoning the art of "prompt whispering" for the science of **AI Systems Engineering**. Whether dealing with rigid schemas or subjective rubrics, DSPy provides the primitives—Signatures, Assertions, and Optimizers—to build reliable, self-improving systems.

Resources

- DSPy GitHub Repository
- LM Assertions Paper