GoodData

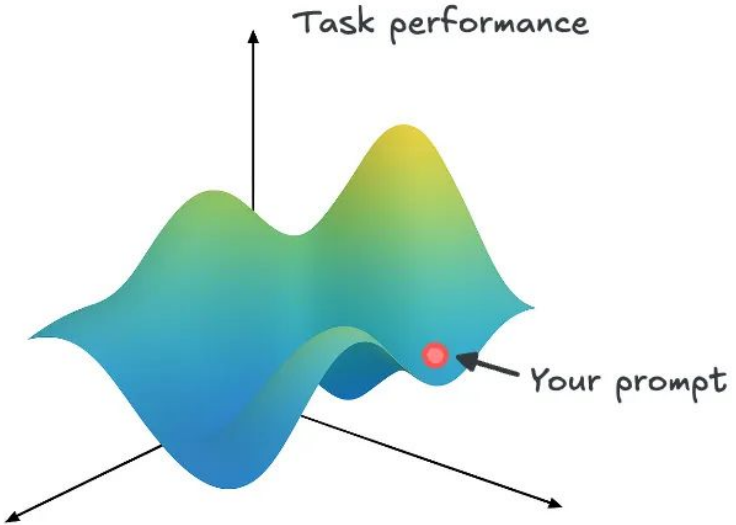
Using Evals For DSPy Optimization

Jakub Žovák

Prompt Optimization

Motivation

- Manual prompting can be brittle and unscalable
  - Akin to hand-tuning the weights for a classifier
- No use of data in prompt design
- Iterative development is unreliable (regressions)
- Compound AI system are hard to tune
- Some prompts can be surprisingly effective:
  - “Model’s proficiency in mathematical reasoning can be enhanced by the expression of an affinity for **Star Trek**”
    (The Unreasonable Effectiveness of Eccentric Automatic Prompts)

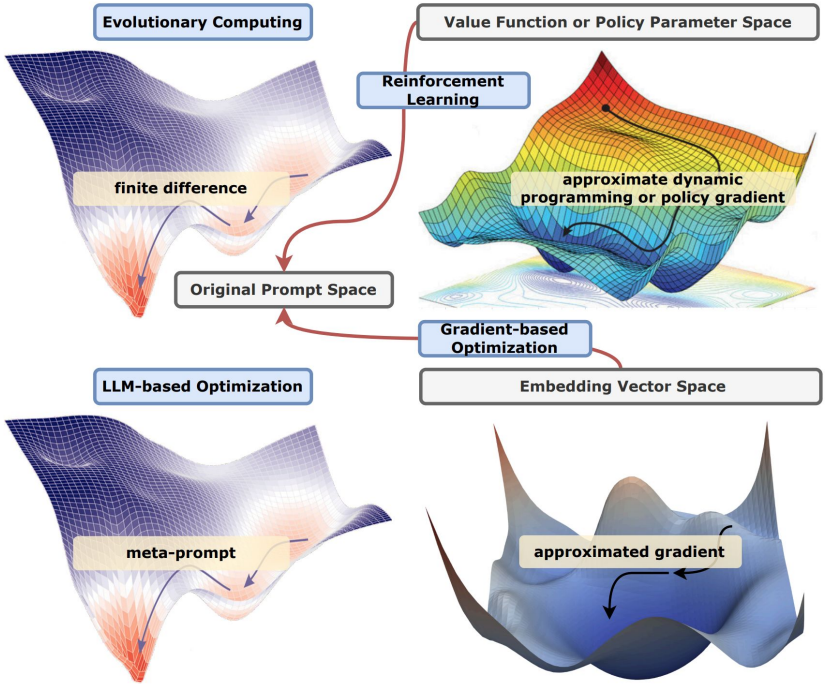


Learning DSPy: The power of good abstractions

Prompt Optimization

Overview

- **Data-driven prompt development**
- Prompt tuning similar to training a ML model
- Components to optimize
  - Instructions
  - Few-shot examples
- Optimization approaches
  - **Meta prompting**
  - Evolutionary
  - Gradient-based
  - Reinforcement Learning

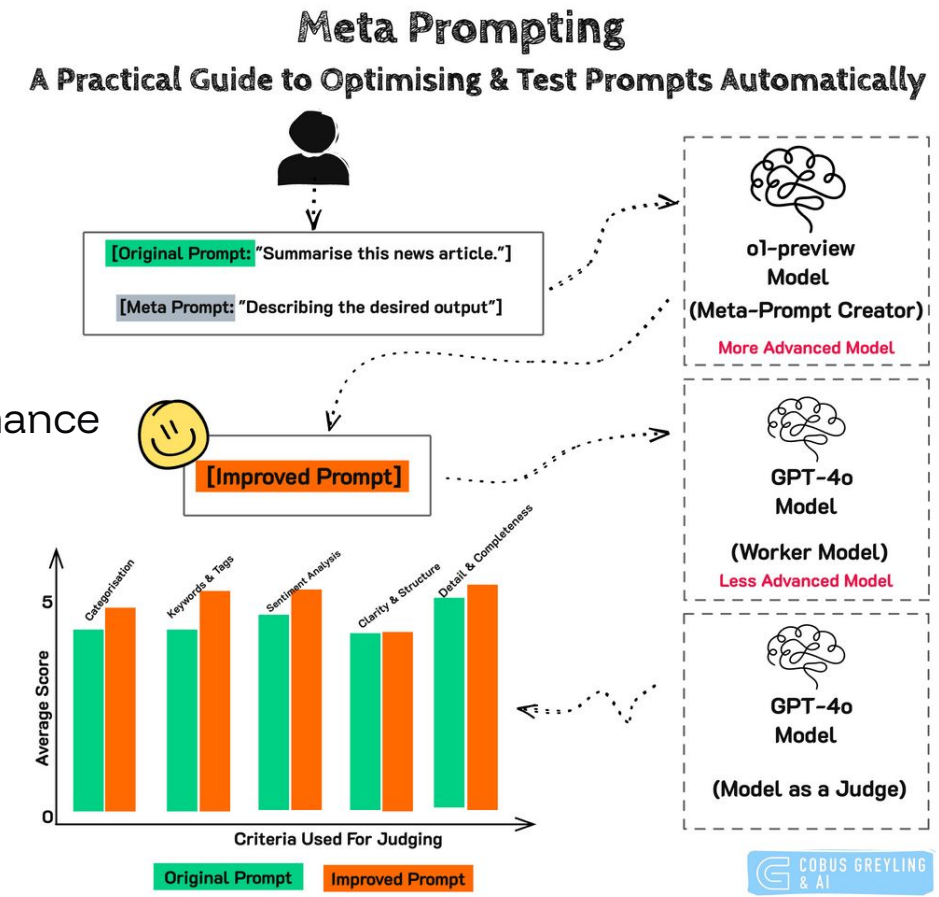


A Survey of Automatic Prompt Engineering: An Optimization Perspective; Li

Prompt Optimization

Meta Prompting

- Let AI design the prompts
- **Meta prompts generate task prompts**
- Meta prompting enhances LLM performance
  (Meta-Prompting: Enhancing Language Models; Suzgun)
- Applications
  - Prompt linting & formatting
  - Optimization
  - Cross-model support

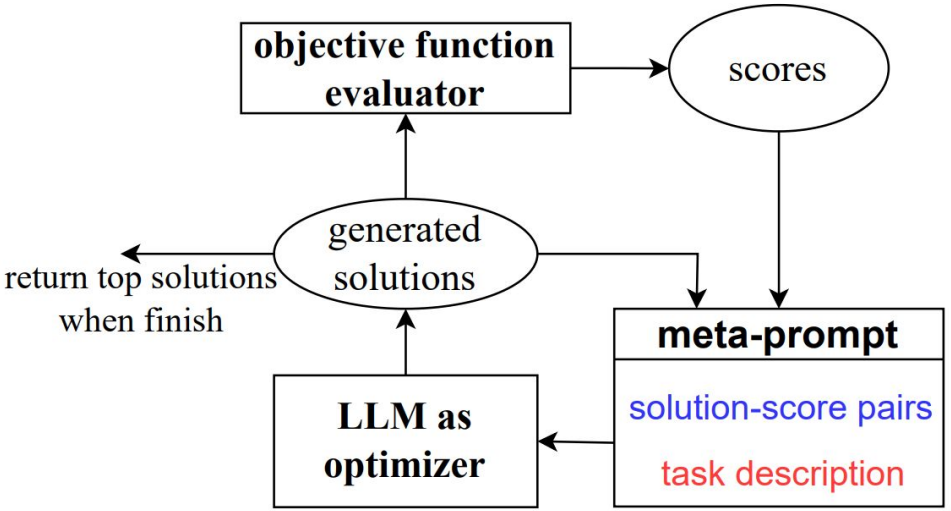


Meta Prompting: A Practical Guide to Optimising Prompts Automatically

Prompt Optimization

OPRO

- Optimization by PROmpting (OPRO)
  - Simplest instruction optimizer
  - No few-shot examples selection
  - Everything passed to meta prompt
- More advanced strategies can be used
  - Chain-of-Thought
  - Reflection
  - Beam Search
- Can be combined with heuristic search methods

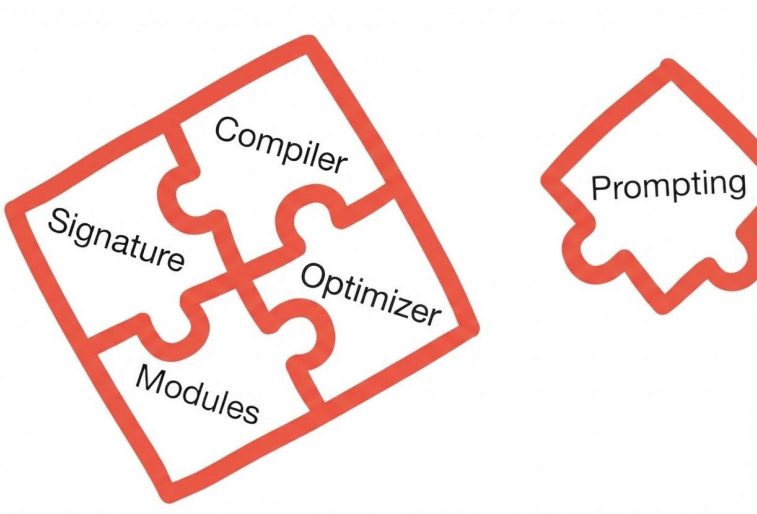


LARGE LANGUAGE MODELS AS OPTIMIZERS

DSPy

Declarative Self-improving Python

- **Optimization library inspired by PyTorch**
- Declarative definition of prompts through programs
- Developed at Stanford in series of papers starting from 2022
- Adopted by industry: Databricks, AXA, Replit
- Has support in GenAI ecosystem
  - Arize Phoenix, MLFlow, Langfuse
- **Can be used without optimizer**
  - Useful abstractions and primitives

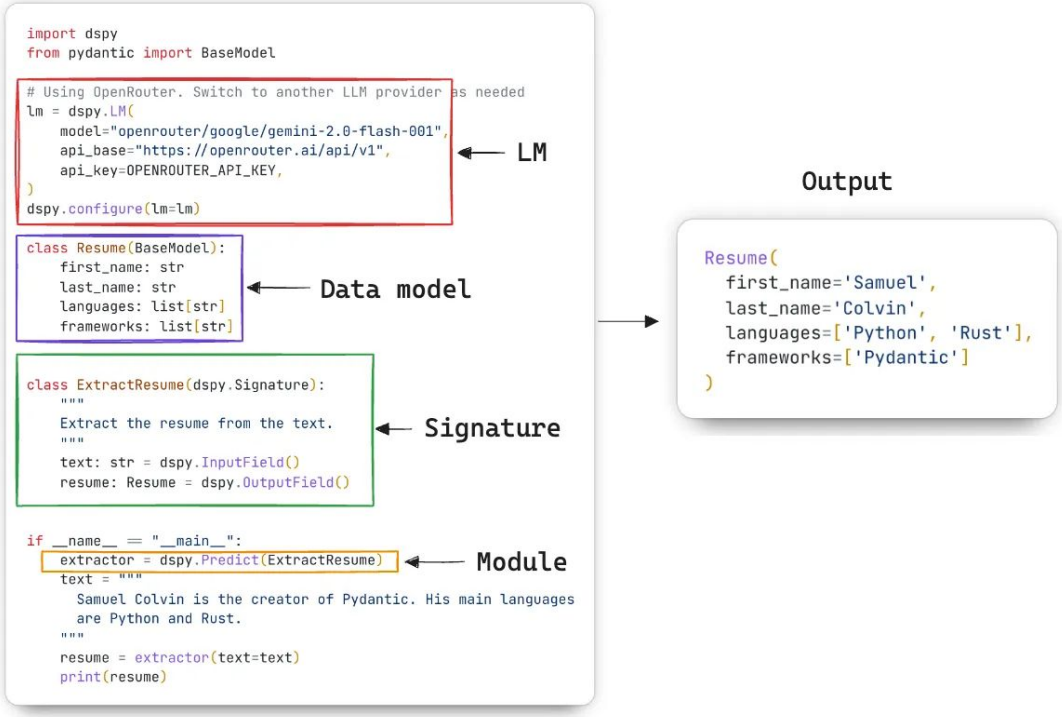


Intro to DSPy
(AI Modified)

DSPy

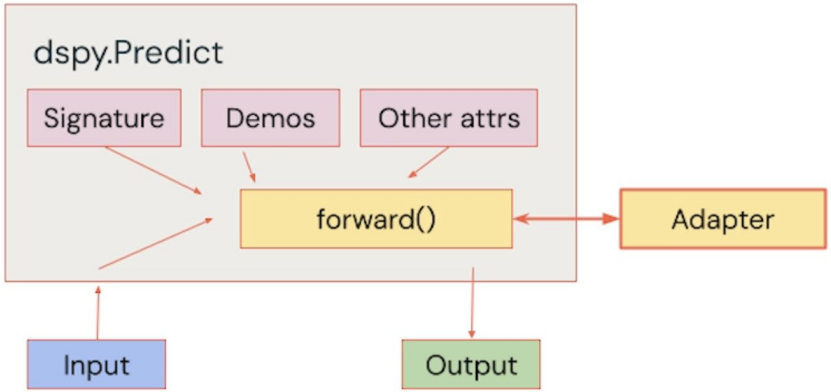
Architecture Components

- **Signatures**
  - Instruction specification
  - LLM input output definition
- **Modules**
  - A core unit of DSPy program
  - Composable
- **Adapters**
  - Interface (middleware) for LLMs
- **Optimizers**
  - Tune the DSPy program



Learning DSPy: Understanding the internals

DSPy
Module



DSPy: Build and Optimize Agentic Apps

- **A core unit** of DSPy programs
- Encapsulates the logic for interacting with LLMs
  - Can contain **arbitrary Python code**
  - **Composable**
- Abstracts various prompting techniques
  - Chain of thought
  - Reflection
  - ReAct

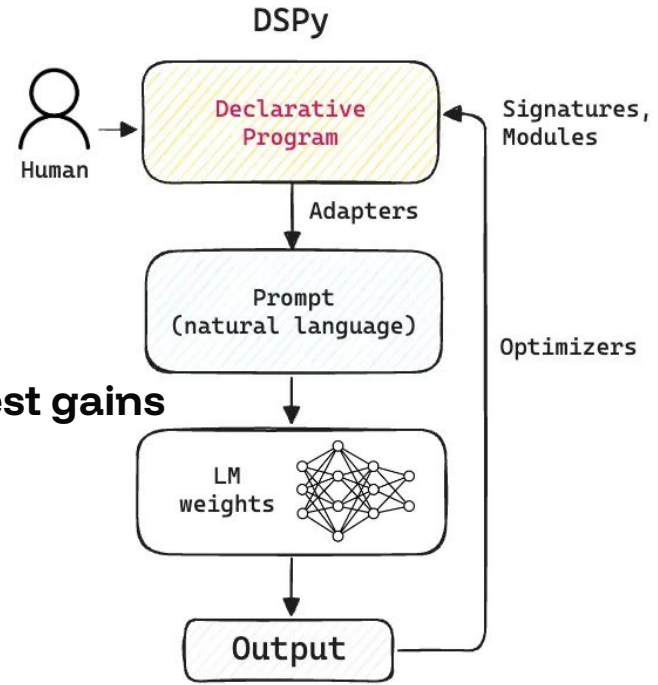
```python
class Hop(dspy.Module):
    def __init__(self, num_docs=10, num_hops=4):
        self.num_docs, self.num_hops = num_docs, num_hops
        self.generate_query = dspy.ChainOfThought('claim, notes -> query')
        self.append_notes = dspy.ChainOfThought('claim, notes, context -> new_notes: list[str], titles: l

    def forward(self, claim: str) -> list[str]:
        notes = []
        titles = []

        for _ in range(self.num_hops):
            query = self.generate_query(claim=claim, notes=notes).query
            context = search(query, k=self.num_docs)
            prediction = self.append_notes(claim=claim, notes=notes, context=context)
            notes.extend(prediction.new_notes)
            titles.extend(prediction.titles)

        return dspy.Prediction(notes=notes, titles=list(set(titles)))
```

DSPy Modules Documentation

DSPy

Optimizers

- Tunable parameters in DSPy
  - Prompt instructions
  - Few-shot examples
  - Underlying LLM itself (LORA)
- **Optimizing few-shot examples often provides greatest gains** (Optimizing Instructions and Demonstrations)
- Successful optimization requires **appropriate evals**
- Prominent Optimizers
  - Few-shot Examples
    - BootstrapFewShotWithRandomSearch
  - Instruction Optimization
    - GEPA - Instruction tuning through reflective prompt mutation

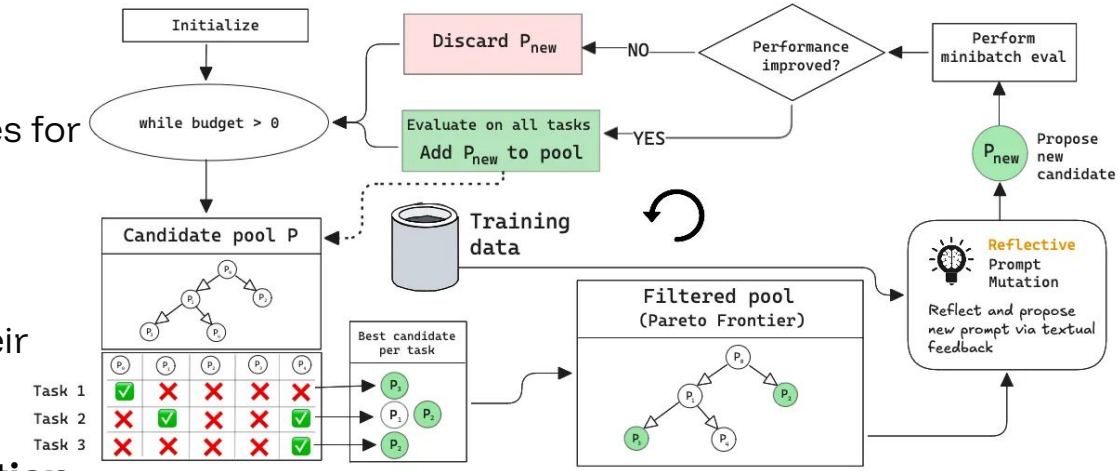


Learning DSPy: The power of good abstractions

DSPy

Genetic-Pareto Reflective Prompt Evolution (GEPA)

- **Reflective prompting**
  - Reflect on the execution traces for each module
- **Evolutionary approach**
  - Mutate prompts and track their lineage
- **Pareto-Based candidate selection**
  - Avoid stucking in local optimum by maintaining multiple candidates
- **Text feedback**
  - Specify feedback instruction yourself

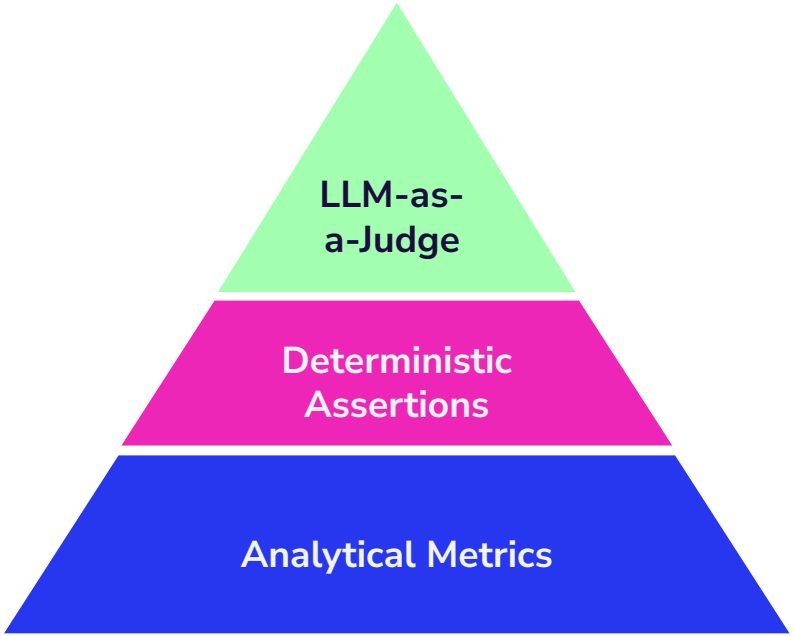


GEPA: REFLECTIVE PROMPT EVOLUTION CAN OUTPERFORM REINFORCEMENT LEARNING
Diagram redrawn by Rao

Evals

Usage Pyramid

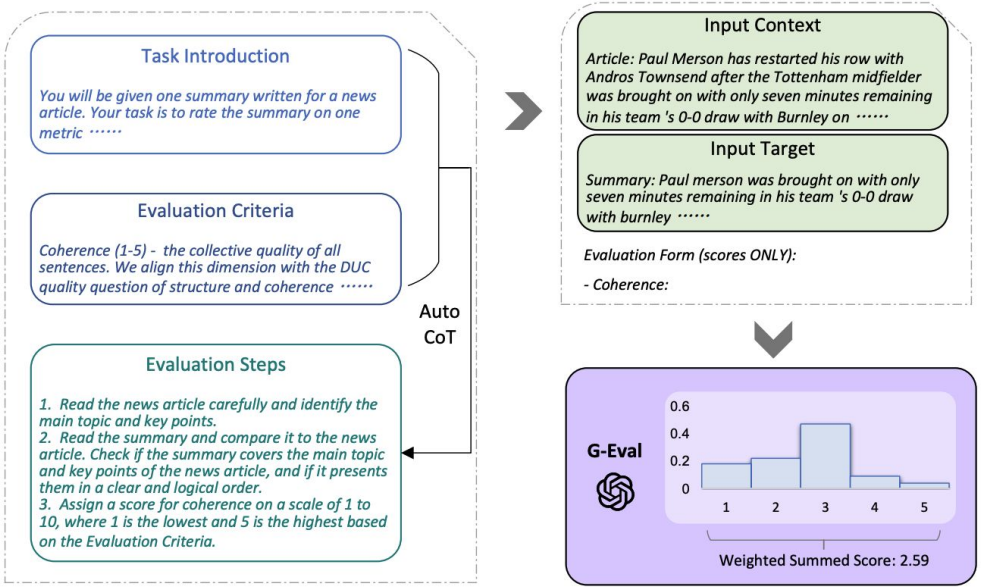
- **LLM/Agent-as-Judge**
  - Use sparingly
  - G-Eval - faithfulness, answer relevancy …
  - Optimize open-ended tasks
- **Deterministic Assertions**
  - Use case specific
  - Programmatic evals, regex, …
  - Optimize classification, info. extraction …
- **Analytical Metrics**
  - Globally applicable numerical metrics
  - Perplexity, pass@k, NLP metrics …
  - Not a good proxy for optimization



Evals

G-Eval (LLM-as-a-Judge)

- LLMs are unreliable at producing real-number scores
- G-Eval **improves score extraction** by
  1. Chain-of-Thought
  2. Form-filling Paradigm Scoring
  3. **Weighting By Token Probabilities**
- Reference Implementation DeepEval
  - Rubrics Support
  - Multi-turn Evaluation
  - Predefined Metrics
  - CoT Caching

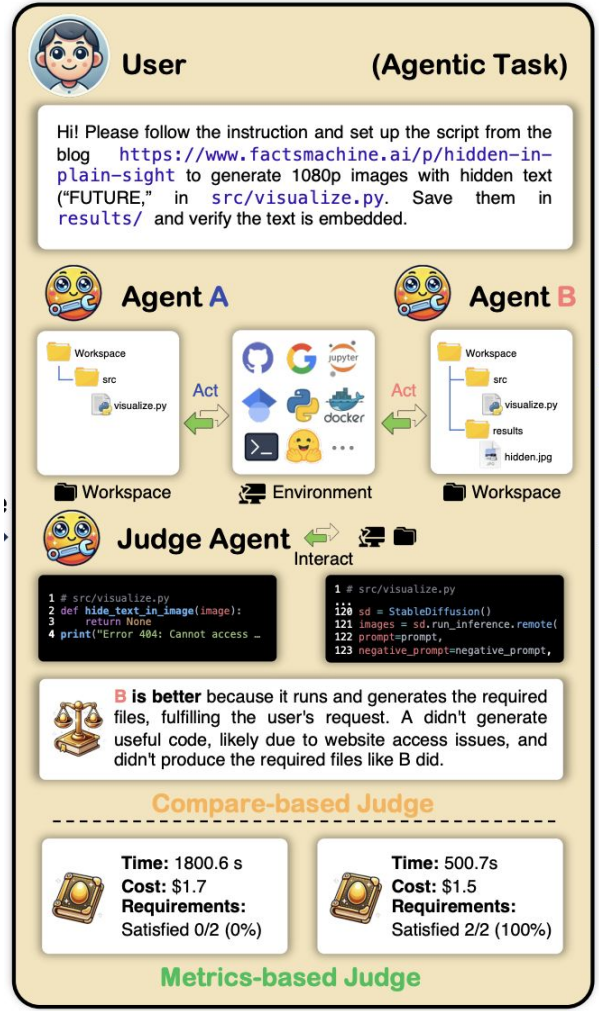


G-EVAL: NLG Evaluation using GPT-4 with Better Human Alignment

Evals

Agent-as-a-Judge

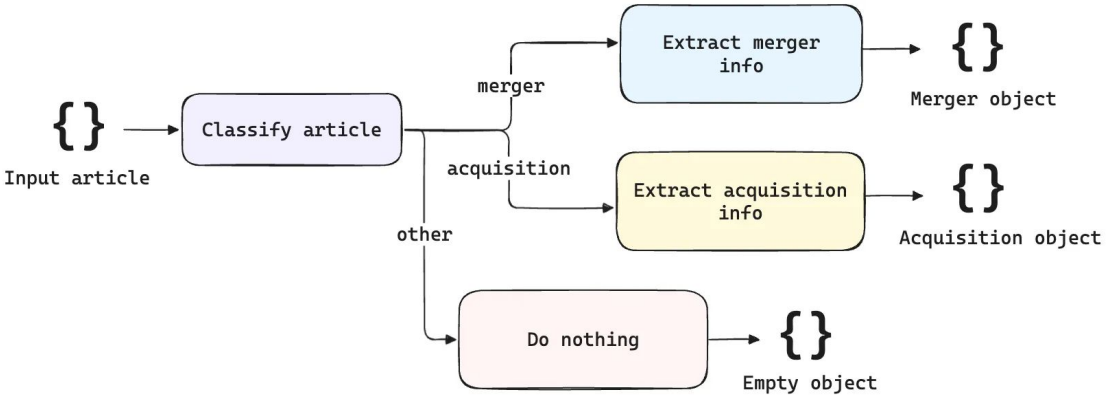
- LLM Judge is to static for agentic workflow
- Agent-a-as-Judge has extended capabilities
  - Environment access
  - Tools
  - Rubric Discovery
- Much **more aligned with human judges**
- Possibility to optimize much more
  - Context management
  - Multiple prompts at once
- Still an area of active research for optimization



Agent-as-a-Judge: Evaluate Agents with Agents

Demo

Information Extraction Task



- Tasks
  - Classification
  - Information extraction
- Optimizers used
  - BootstrapFewShotWithRandomSearch
  - GEPA
- LLMs used
  - Predictor - gemini-2.5-flash-lite
  - Teacher - openai/gpt-4o
- Dataset
  - Gold mining industry domain
  - 40 articles about mergers and acquisitions
- Evaluated with **deterministic assertions**

Learning DSPy: Working with optimizers

Github Repository

Zovi343/dspy-demo-improved

Results

Optimized Extraction Prompt

- Baseline prompt
  - Extract information about companies involved in a merger deal. If the currency symbol is just \"$\", assume USD.
- Optimized prompt captures
  - Business logic
  - Generic edge cases
  - Data structure
  - Problem example

You are given a news article text describing a merger deal between two companies. Your task is to extract structured information about the merger, focusing on the following fields:

- article_id: The unique identifier for the article (provided in the input context).
- company_1: The full name of the first company involved in the merger, as mentioned in the article.
- company_1_ticker: The stock ticker(s) for company_1, including the exchange prefix if present (e.g., 'TSXV:TGI'). If no ticker is mentioned, return None.
- company_2: The full name of the second company involved in the merger, as mentioned in the article.
- company_2_ticker: The stock ticker(s) for company_2, including the exchange prefix if present (e.g., 'TSXV:BML'). If no ticker is mentioned, return None.
- merged_entity: The name of the new or continuing entity after the merger, as specified in the article. If the merged entity retains the name of one of the original companies, use that name.
- deal_amount: The value of the merger deal, including both the numeric value and the magnitude (e.g., '2.8 billion', '540 million'). Do not omit the magnitude.
- deal_currency: The three-letter currency code for the deal amount (e.g., 'USD', 'CAD', 'AUD'). If the currency symbol is '$', assume 'USD'. If the symbol is 'A$', use 'AUD'. If the symbol is 'C$', use 'CAD'.
- article_type: Always set this to 'merger'.

Guidelines:

- Extract company names and tickers exactly as they appear in the article. If a ticker is given with an exchange prefix (e.g., 'TSXV:TGI'), include the prefix. If only the ticker symbol is given, include it as is.
- For deal_amount, always include the magnitude (e.g., 'billion', 'million') as stated in the article. Do not return only the numeric value.
- For deal_currency, map currency symbols to their respective codes as follows: '$' → 'USD', 'A$' → 'AUD', 'C$' → 'CAD'.
- If any field is not mentioned in the article, return None for that field.
- Output your answer as a single line in the following format: article_id= company_1='<company_1>' company_1_ticker= <company_1_ticker> company_2='<company_2>' company_2_ticker=<company_2_ticker> merged_entity='<merged_entity>' deal_amount='<deal_amount>' deal_currency='<deal_currency>' article_type='merger'
- Ensure all string values are enclosed in single quotes, and list tickers as Python lists (e.g., ['TSXV:TGI']) or None.

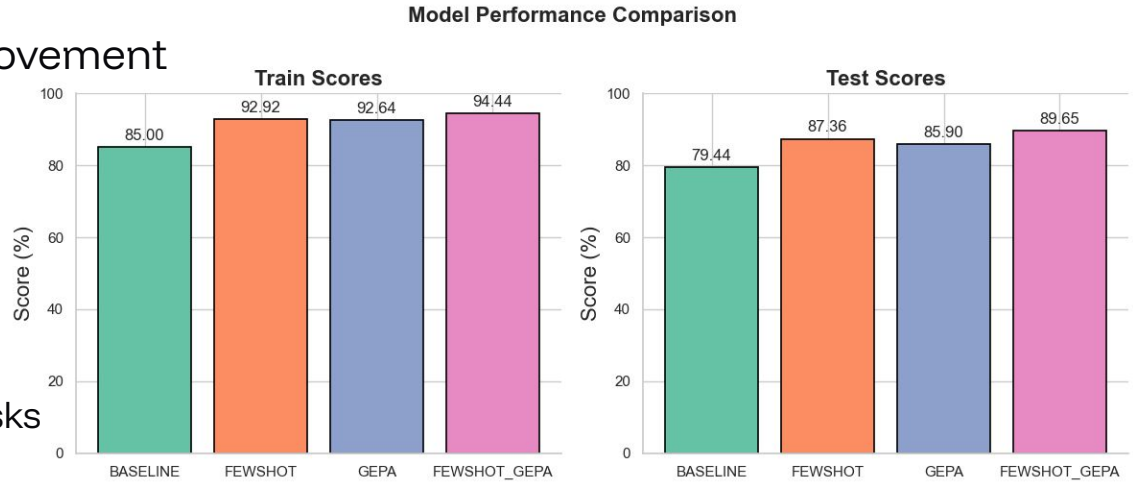
Example: If the article says "Tundra Gold Inc. (TSXV:TGI) and Boreal Mining Ltd. (TSXV:BML) have entered a definitive agreement to merge in a C$540 million all-stock transaction. The combined company, to be named Northern Tundra Resources...", your output should be:

> article_id=39 company_1='Tundra Gold Inc.' company_1_ticker=['TSXV:TGI'] company_2='Boreal Mining Ltd.' company_2_ticker= ['TSXV:BML'] merged_entity='Northern Tundra Resources' deal_amount='540 million' deal_currency='CAD' article_type='merger'

Results

Conclusion

- **Few-shot boost the performance the most**
- **GEPA captured domain specific instruction**
- Few-shot + GEPA = ~10% improvement
- DSPy benefits grow with
  - Domain complexity
  - Dataset size
- DSPy is not a silver bullet
  - Not necessary for simpler tasks
  - Steep learning curve



Thank you for your attention!

- Any questions?
- We are hiring
  - AI Architect
    - Skilled in AI/ML
    - Also proficient in SWE
- Feel free to connect with me on LinkedIn: