



Agentic AI Foundation

AGNTCon + MCPCon China

Where the **agentic stack** is being built.

How We Built Reliable Self Evolving Agent Skills

Subhro Das | ML Engineer @Mattoboard

About me

Suvrakamal Das

(Subhro)

ML Engineer

I love working at the intersection of AI and hardware.



Open Source

PyTorch, vLLM



Conferences

PyTorch Conference, PyCon USA, PyCon JP, etc



Papers

SciPy 2024, 2026

What This Talk Is NOT About

- ✗ Teaching you how to build agents
- ✗ Creating hype that agents solve all your problems
- ✗ Making you believe that evals & testing are dead
- ✗ Relying completely on a coding agent without domain knowledge

Table of Contents

- 01 DSPy and benefits
- 02 Chain of thought and ReAct using DSPy
- 03 GEPA
- 04 Evals
- 05 Observability
- 06 Reflection and feedback loops
- 07 Agent skills



Agentic AI Foundation

AGNTCon + MCPCon
China

The Anatomy of a GPT-5 prompt

Act as an expert travel guide focused on recommending lesser-known, unique outdoor hikes within two hours of San Francisco.

- Begin with a concise checklist (3-7 bullets) of steps you will follow to complete this task, focusing on conceptual planning rather than details.
- Identify and present the top 3 medium-length hikes (not among the most popular) within a two-hour drive from San Francisco.
- Ensure each hike selected offers a unique adventure due to its scenery, remoteness, or distinctive qualities.

- Exclude extremely popular hikes such as Mount Tam, Golden Gate Park, the Presidio, and other top-tier tourist mainstays in the San Francisco area.

- Prioritize accuracy: Hike names must match official listings (e.g., AllTrails), and all time and distance estimates should be realistic and reliable.
- Highlight what makes each hike an outstanding adventure in a concise summary.

- Internally vet all suggested hikes to guarantee they are real, under-the-radar, and fit the specified parameters before responding.
- Cross-check hike names and details with reliable outdoor hiking sources.
- Optimize for clarity, concise presentation, and practical value.

- Return the results as a properly formatted Markdown table with these columns:

```
| --- | --- | --- | --- | --- |  
| [Hike 1 name] | [Address] | [X:X] | [XX] | [X:XX] | [Summary] |  
| [Hike 2 name] | [Address] | [X:X] | [XX] | [X:XX] | [Summary] |  
| [Hike 3 name] | [Address] | [X:X] | [XX] | [X:XX] | [Summary] |
```

- Task is complete when three verified, unique medium-length hikes, excluding overly popular options, re returned in the specified format, and validation has confirmed full compliance with all requirements.

Role

Task

Context

Reasoning

Output
formatStop
conditions

The Reality of Prompt-Based Systems

While prompts are great to start, scaling them in production introduces severe hurdles:

EXHAUSTIVE

High Complexity & Size

Prompts must be massive and highly detailed to get consistent results.

FRAGILE

Non-Portable Across Models

They break immediately when migrating from one LM to another.

COSTLY

High Maintenance Overhead

Manual updates and fine-tuning quickly become expensive & slow.

UNRELIABLE

Broken Production Outputs

Edge cases in production constantly violate expected JSON structures.

Where the **agentic stack** is being built.

DSPy — Declarative Self-Improving Python

Declarative AI Programming

Open-source framework emphasizing programming over manual prompting to build modular AI software.

Eliminate Prompt Guesswork

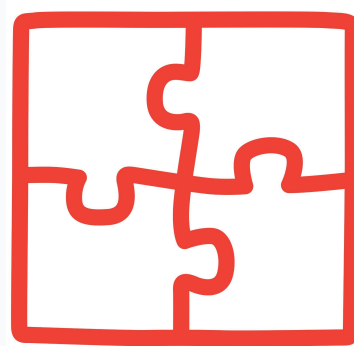
DSPy systematically handles the underlying prompts and "how" mechanics so you can focus purely on the "what".

Python-Native & Robust

Fully equipped with Python type hints, built-in Pydantic validation, and support for multiple LLMs via a LiteLLM backend.

Self-Improving Optimizers

The real charmer: advanced optimizers that automatically fine-tune prompts based on your specific datasets and metrics.



DSPy

Signatures — From Prompt to Program



Agentic AI Foundation

AGNTCon + MCPCon
China

dspy_signatures.py

```
dspy_signatures.py Python

import dspy
lm = dspy.LM("ollama_chat/llama3.2:1b", api_base="http://localhost:11434", api_key="")
dspy.configure(lm=lm)

lm("What is India known for?")

"""
India is globally renowned for its rich cultural heritage, profound spiritual traditions, and incredible diversity. It is celebrated worldwide as the birthplace of yoga and Ayurveda, the origin of major world religions like Hinduism and Buddhism, and home to iconic architectural marvels like the Taj Mahal.
"""
```

1. Declarative Python Signature

Define the input and output fields programmatically. DSPy translates this clean structure into optimized prompts under the hood.

dspy_101.py

```
dspy_101.py Python

print(lm.history[-1]['prompt'])

"""
Given the fields 'question', produce the fields 'response'.

---

Follow the following format.

Question: ${question}
Response: ${response}

---

Question: What is India known for?
Response:
"""
```

2. Compiled Prompt & Execution

The framework compiles your signature into structured prompt instructions and automatically handles formatting and parsing.

Efficient Signatures



Agentic AI Foundation

AGNTCon + MCPCon
China

dspy_101.py

Python

```
from typing import Literal

class Classify(dspy.Signature):
    """Classify sentiment of a given sentence."""

    sentence: str = dspy.InputField()
    sentiment: Literal["positive", "negative", "neutral"] = dspy.OutputField()
    confidence: float = dspy.OutputField()

classify = dspy.Predict(Classify)
classify(sentence="This book was super fun to read, though not the last chapter.")

"""
Prediction(
  sentiment='Sentence: This book was super fun to read, though not the last
chapter.\nSentiment: Positive',
  confidence='0.85'
)
"""
```

CoT(Chain of Thought), ReAct agents with DSPy



Agentic AI Foundation

AGENTCon + MCPCon
China

dspy_101.py

Python

```
# Define a module (ChainOfThought) and assign it a signature (return an answer, given a question).
qa = dspy.ChainOfThought('question -> answer')

# Run with the default LM configured with `dspy.configure` above.
response = qa(question="How many floors are in the castle David Gregory inherited?")
print(response.answer)
# 4

lm.inspect_history()
"""
Given the fields `question`, produce the fields `answer`.

---

Follow the following format.

Question: ${question}
Reasoning: Let's think step by step in order to ${produce the answer}. We ...
Answer: ${answer}

---

Question: How many floors are in the castle David Gregory inherited?
Reasoning: Let's think step by step in order to Reasoning: Let's think step by step in order
to determine how many floors are in the castle David Gregory inherited. We know from
historical records that David Gregory inherited Kinnauld Castle in Scotland. This castle, also
known as Castle Huntly, is a 15th-century structure. While exact architectural details can
vary, most reliable sources describe Kinnauld Castle as having four main floors (including the
ground floor and three upper levels), consistent with typical tower-house design of the
period.

Answer: 4
"""
```

dspy_react.py

Python

```
def evaluate_math(expression: str):
    return dspy.PythonInterpreter({}).execute(expression)

def search_wikipedia(query: str):
    results = dspy.ColBERTv2(url="http://20.102.90.50:2017/wiki17_abstracts")(query, k=3)
    return [x["text"] for x in results]

react = dspy.ReAct("question -> answer: float", tools=[evaluate_math, search_wikipedia])

pred = react(question="What is 9362158 divided by the year of birth of David Gregory of
Kinnauld castle?")
print(pred.answer)

# 5761.328

lm.inspect_history(n=1)
"""
Your input fields are:
1. `question` (str):
2. `trajectory` (str):
Your output fields are:
1. `reasoning` (str):
2. `answer` (float):
All interactions will be structured in the following way, with the appropriate values filled
in.

[[ ## question ## ]]
{question}

[[ ## trajectory ## ]]
{trajectory}

[[ ## reasoning ## ]]
{reasoning}

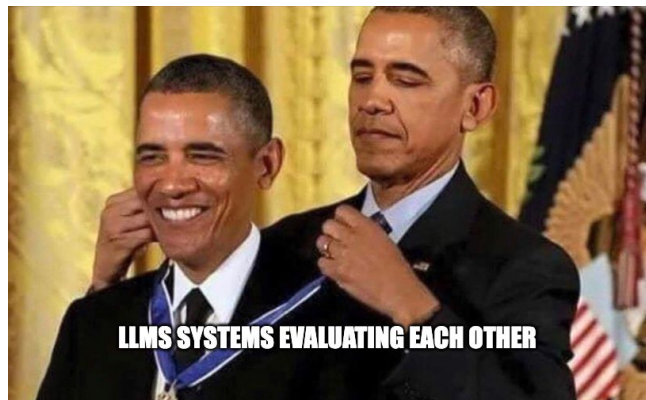
[[ ## answer ## ]]
{answer}          # note: the value you produce must be a single float value

[[ ## completed ## ]]
In adhering to this structure, your objective is:
    Given the fields `question`, produce the fields `answer`.
"""
```

Is Evals All You Need?

There are many ways to define your metric, but most use one or more of these patterns:

- **Labeled data comparisons:** compare the prediction against a “gold” output, usually labeled by a human expert
- **Rule-based checks:** evaluate with code that checks verifiable properties
- **LLM as judge:** even with reasoning/thinking mode, this can carry bias

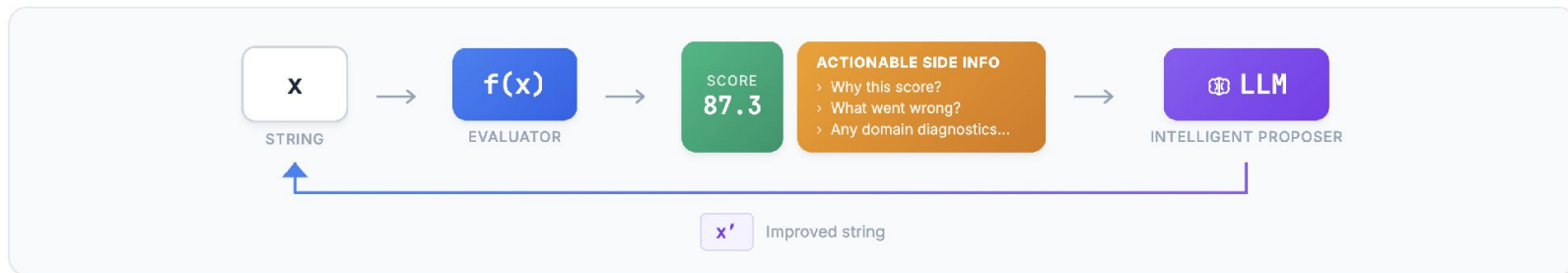


Optimize Anything with GEPA

optimize_anything Universal Interface for Text Optimization

maximize $f(x: \text{str}) \rightarrow \text{score}$

Any optimization problem can be expressed as maximizing evaluator f over strings by serializing artifacts as text.
The evaluator returns a score plus **Actionable Side Information (ASI)** — domain context explaining **why**.



EXAMPLE INSTANTIATIONS



Code Optimization

x : CUDA kernel code
 f : -runtime
ASI: Compiler errors, profiler traces...



Numeric Optimization

x : Params (str) or solver code
 f : -objective
ASI: Gradients, violations, logs...



Prompt / Agent Harness

x : Prompt / agent harness code
 f : Accuracy, token cost
ASI: Reasoning traces, feedback...



Policy Optimization

x : Scheduling policy
 f : -cost (efficiency)
ASI: Job traces, SLA violations...

GEPA - the game changer

The optimized prompt that GEPA generates for AIME, which achieves improves GPT-4.1 Mini's performance from **46.6% to 56.6%**, an improvement of 10% on AIME 2025. Note the details captured in the prompts in just 2 iterations of GEPA. GEPA can be thought of as precomputing some reasoning (during optimization) to come up with a good plan for future task instances.

Seed Prompt for Second-Hop of Multi-Hop QA System

Given the fields question, summary_1, produce the fields query.

GEPA's Optimized Prompt for Second-Hop of Multi-Hop QA System, GPT-4.1 Mini

You will be given two input fields: question and summary_1. Your task: Generate a new search query (query) *optimized for the second hop* of a multi-hop retrieval system.

- The original user question is typically complex and requires information from multiple documents to answer.
- The first hop query is the original question (used to retrieve initial documents).
- Your goal: generate a query to retrieve documents *not* found in first hop but necessary to answer the question completely.

Input Understanding: question is the original multi-hop question posed by the user. summary_1 is a concise summary of information from a document retrieved in the first hop, which partially addresses the question.

Purpose and Context:

- Your generated query aims to find the *missing pieces* of information needed to fully answer the question. ...
- The query must retrieve relevant documents *NOT* found in first hop ... for final answer extraction.

Key Observations and Lessons:

- First-hop documents often cover one entity or aspect.
- Remaining relevant documents often involve connected or higher-level concepts mentioned in summary_1 but not explicitly asked in the original question. The query should target these *missing*, but logically linked, documents.
- Avoid merely paraphrasing the original question or restating known facts from summary_1.
- Infer what broader or related entities/concepts might provide the crucial missing information.
- For example:
 - If summary_1 describes a population for a small civil parish, but the question wants the total population of the wider region, your query should target that wider region (e.g., "Madeira archipelago population in 2011").
 - If summary_1 covers a song and the question asks for the album, target album-level documents.

How to Build the Query:

- Identify entities or topics mentioned in summary_1 that are related but different from first-hop documents.
- Reframe the query to explicitly mention these broader or related entities *connected to the original question*.
- Include relevant key context from the question to maintain specificity, but shift focus to the missing piece.
- The goal is to retrieve documents that link or complement what was retrieved initially.

Practical Strategy:

- Read the summary_1 carefully to spot references to bigger contexts or other entities not covered in the first hop.
- Ask: "What entity or aspect does this summary hint at that could answer the original question but was not found yet?"
- Formulate a precise, focused factual query targeting that entity or concept to retrieve the missing documents.

Output:

- Produce query as a clear, concise question or keyword phrase designed for efficient retrieval of second-hop documents.
- Ensure the query relates logically to the original question while targeting the broader or complementary knowledge identified in summary_1. ... Do not include the original question or simply rephrase it. Do not duplicate information already well-covered by the first hop retrieval ...

Observability



Agent AI Foundation

AGNTCon + MCPCon
China



Tracing

Capture and debug LLM interactions and agent...



Evaluation

Measure and compare LLM quality with built-in and custo...



Prompts

Version control and manage prompts with aliases across...



AI Gateway

Unified API for multiple LLM providers with rate limiting.

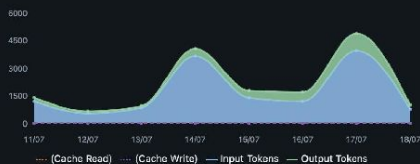


Model Training

Track experiments with parameters, metrics, and...

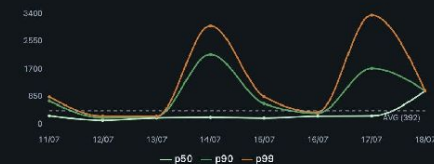
Token Usage

16.48K (13.55K input, 2.92K output)



Tokens per Trace

392 avg per trace



Traces

42



Cost Breakdown

\$0.06687 Total Cost

Model Provider

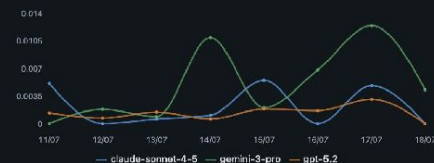


■ claude-sonnet-4-5 25%
■ gemini-3-pro 59%
■ gpt-5.2 15%

Cost Over Time

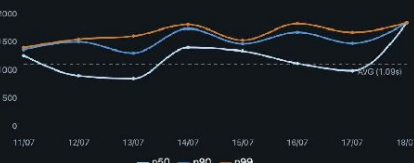
\$0.06687 Total Cost

Model Provider All models



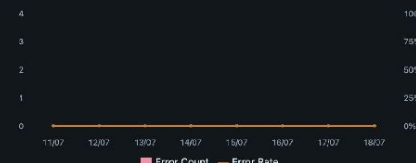
Latency

1.09s



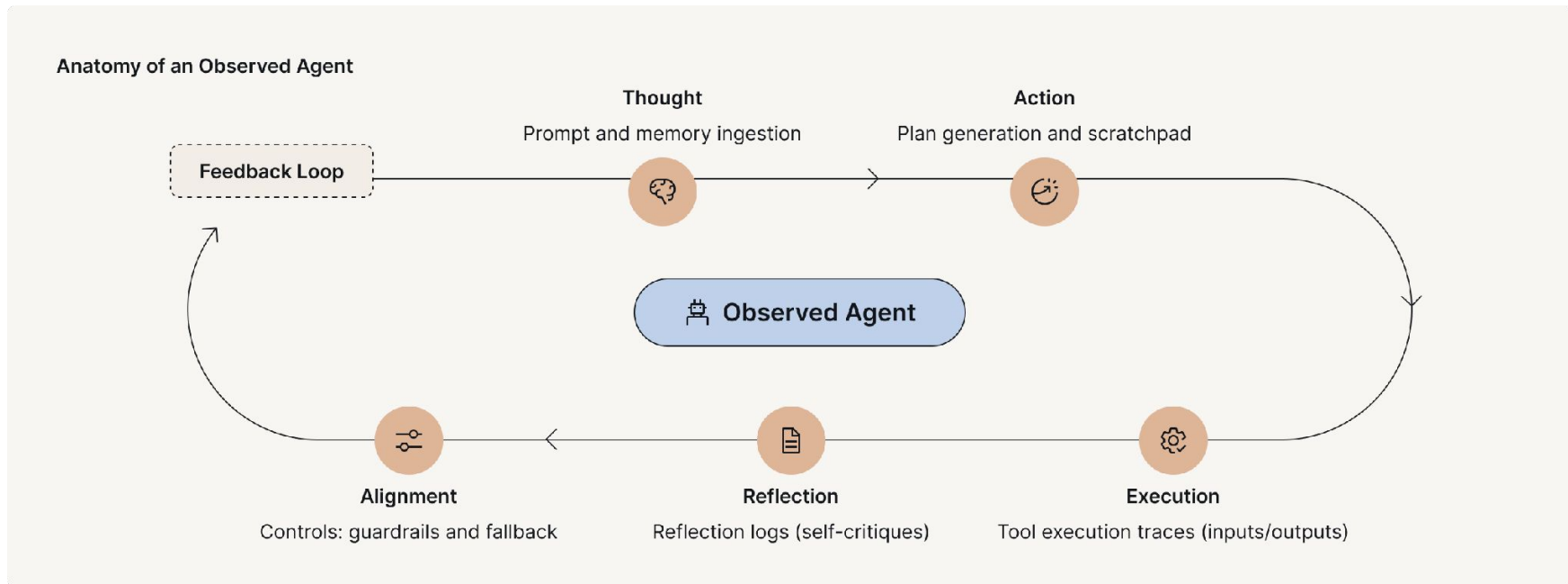
Errors

0 (Overall error rate: 0.0%)



Where the agentic stack is being built.

Reflection & Feedback Loops



WTF Is skill.md?



Agentic AI Foundation

AGNTCon + MCPCon
China

01

Portable Agent Capabilities

Portable packages of instructions, APIs, and scripts that give agents specialized domain expertise.

02

Prevent Context Bloat

Loads only the specific knowledge needed for a task, optimizing efficiency and context usage.

03

Customizable Configurations

Includes custom architectural choices, testing style guides, API rules, and more.

Visual Specification Structure

What Skills look like

A Skill directory:

```
anthropic_brand/  
├ SKILL.md  
├ docs.md  
├ slide-decks.md  
└ apply_template.py
```

anthropic/brand_style/SKILL.md

YAML Frontmatter

```
---  
name: Anthropic Brand Style Guidelines  
description: Applies Anthropic's official brand colors and  
typography to PowerPoint presentations  
---
```

Metadata

Overview

Markdown

This skill provides Anthropic's official brand identity resources for PowerPoint presentations. It includes a pre-branded template and tools to apply Anthropic styling to existing presentations.

Freeform
Content

Colors

Main Colors:

- Dark: '#141413' - Primary text and dark backgrounds
- Light: '#faf9f5' - Light backgrounds and text on dark
- Light Gray: '#e8e6dc' - Subtle backgrounds

Typography

- *Headings*: Poppins (with Arial fallback)
- *Body Text*: Lora (with Georgia fallback)

Core Resources

[DSPy Official Docs](#) 

[GEPA Paper](#)  [GEPA Blog](#) 

[Standalone GEPA Library](#) 

[Agent Skills](#) 

[Evals](#) 

[Observability](#) 

Access Slides

Scan the QR code to view or download



Takeaways

01 Program, don't prompt

02 Let optimizers reflect (GEPA)

03 Evals = the product

04 Observe everything

05 Skills > sprawling prompts

06 Domain knowledge still wins

Thank You! 非常感谢!

Got questions? Ask away!



Agentic AI Foundation

AGNTCon + MCPCon
China



👋 **Suvrakamal Das, or Subhro**

😊 ML Engineer @Mattoboard

📄 Website - <https://subhrokomol.in/>

🐙 Open Source: @JaynouOliver

Where the **agentic stack** is being built.