CS 639: Foundation Models
Agents II

Fred Sala

University of Wisconsin-Madison

April 23, 2026

Announcements

- **Homework 5**: out
- Project---ongoing!
- **Next class (4/28): will happen offline; I will release lecture recordings**
  - We'll have extra OH next week!
- Class outline

Thursday April 23	Agents II
Tuesday April 28	Applications
Thursday April 30	Future Areas

End of Class

Outline

- **Review of LLM-Powered Agents**
  - Motivation, goals, differences vs classical agents / standard LMs, overall architectures and key components
- **Agentic Evaluation**
  - What to measure, offline vs online, metrics for trajectories, outcomes, tool use, memories
- **Agent Design: Case Studies**
  - Two examples: deep research and slop-aware coding. Step-by-step design and results

Outline

- **Review of LLM-Powered Agents**
  - Motivation, goals, differences vs classical agents / standard LMs, overall architectures and key components
- **Agentic Evaluation**
  - What to measure, offline vs online, metrics for trajectories, outcomes, tool use, memories
- **Agent Design: Case Studies**
  - Two examples: deep research and slop-aware coding. Step-by-step design and results

Review: What's an Agent?

Let's **differentiate** from a standard LLM

- These usually answer a prompt (single response)
- Agents act, often taking many steps, to accomplish a **goal**
  - Interact with tools, environments, feedback loops
- But, still **built on top of LLMs**
- What do we use them for?
  - As we saw in benchmarks, coding, computer use, research, science, etc.

More Concrete Definition

An LLM agent is a system that uses a FM to **reason**, **plan,** and **act** in an environment

- Often with tools and memory.
- Note: the agent includes the **harness/scaffold**, not just the FM or LLM that powers it

Perception
reads state / context

Planning
chooses next subgoal

Action
calls tools or takes steps

Memory
stores what matters

Adaptation
updates after feedback

Single-Agent vs. **Multi-Agent**

We can create multiple agents

- These get to interact with each other to perform goal
- Note that we can simulate this with a single agent too!
- Key difference: multiple agents permits using **different** harnesses and models
  - This specialization can be more efficient: use cheaper models for easier tasks, etc.
  - But…more complexity!

Single Agent **Patterns**

Variations based on where planning, execution, and verification are placed. Some examples:

- **ReAct**
  - Interleave reasoning and execution
- **Plan/execute**
  - Planning phase followed by execution
- **Reflection**
  - Generate, critique, revise
- **Verifier-based**

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

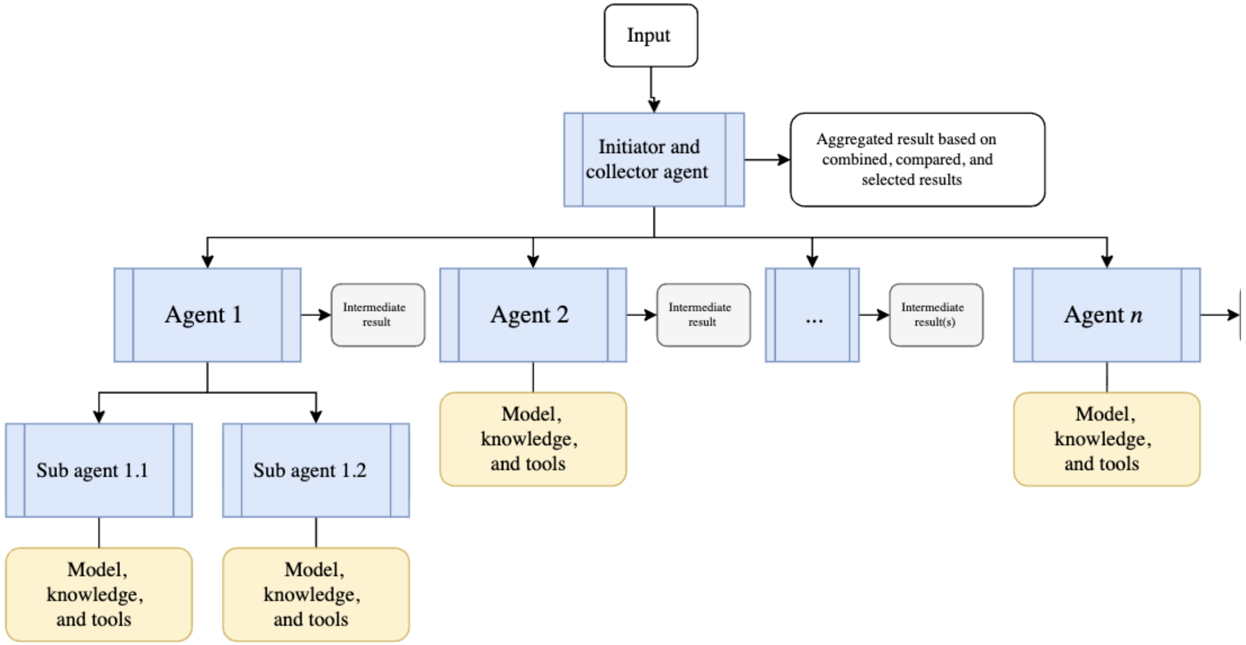
ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

Multi-Agent Patterns

Much more flexible

- Large design space!
- Most common approach: central **orchestrator** splits up work and delegates to **specialists**

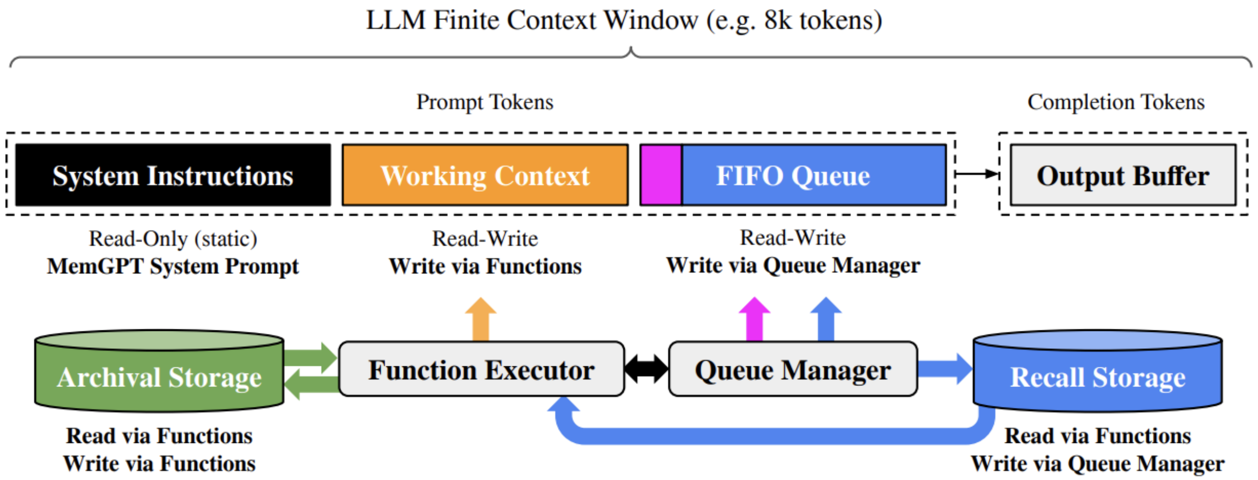


Microsoft

Components: Memories

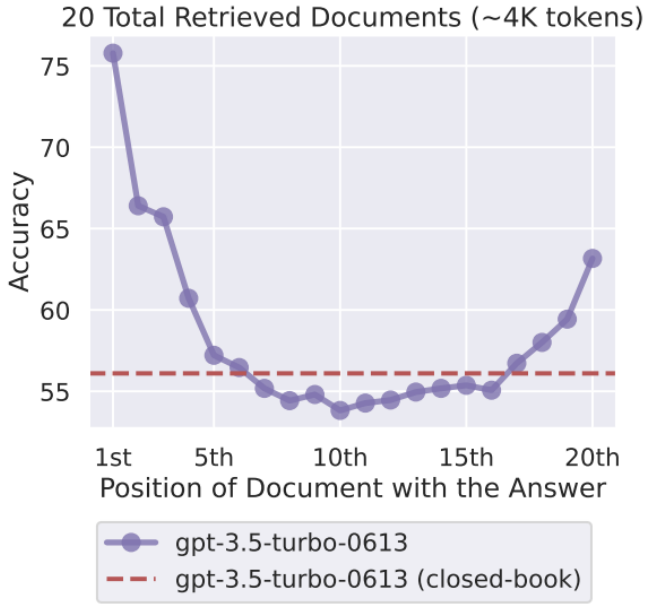
Interactive agents require the use of **memory**

- I.e., we need to store state/other useful information
- Simplest approach: **use LLM context window** as the memory
  - Increasingly tractable with very long-context windows
- Can also create memory systems/agents
  - E.g., MemGPT (Packer et al '24)
- Or buffers, RAG, etc.



Interlude: Long Context Windows

- Simplest approach: **use LLM context window** as the memory
  - Increasingly tractable with very long-context windows
- Note that the advertised long context may not be realistic!
- Liu et al '23 ("Lost in the Middle: How Language Models Use Long Contexts")
- Needle-in a haystack benchmarks
  - Numerous variants



Components: Reasoning and Planning

Classic AI problem (decompose task, plan)

- Addressable via reasoning capabilities of LLMs,
  - Including older methods: chain-of-thought, tree-of-thoughts, self-reflection
  - Example: **ReAct** (Reason + Act) (Yao et al '23)

(1) Hotspot QA

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple … originally designed to control the Front Row media center program …

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (software) ', …]

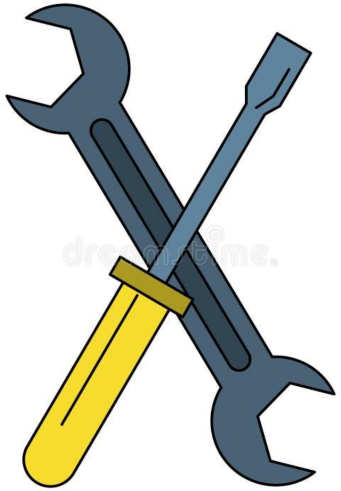
Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys]

Components: Tool Use

Standard component in agentic settings

- Where do we get tools?
  - APIs, web search, etc
  - Or, in some cases use of physical tools
- Most agent frameworks permit the use of web/code-based tools
- Major challenge: tool usage is **noisy**



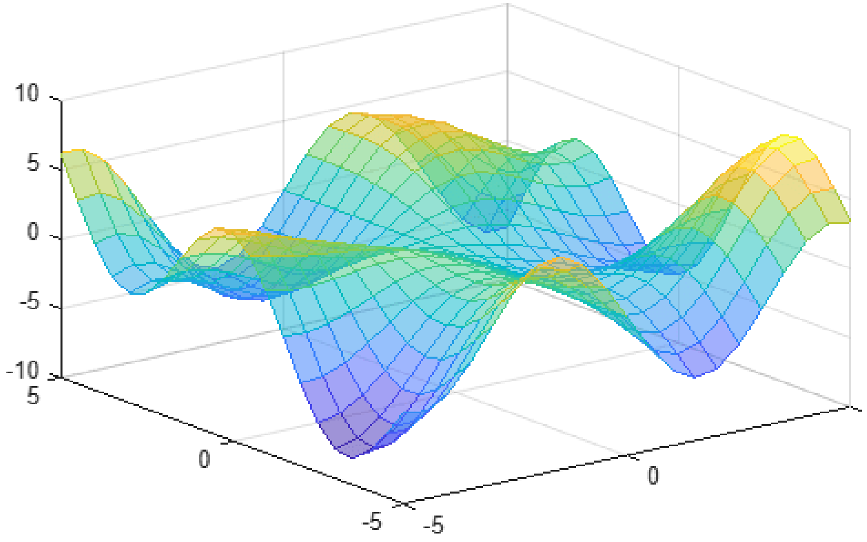
Outline

- **Review of LLM-Powered Agents**
  - Motivation, goals, differences vs classical agents / standard LMs, overall architectures and key components
- **Agentic Evaluation**
  - What to measure, offline vs online, metrics for trajectories, outcomes, tool use, memories
- **Agent Design: Case Studies**
  - Two examples: deep research and slop-aware coding. Step-by-step design and results

Agentic Evaluation: **What** to Measure?

As we've seen, richer than for standard LLMs. Key question: **what should we measure**? Some options:

- Task success rate: did it complete the goal?
- Tool-call accuracy: were calls valid and useful?
- Step efficiency: how much work did it waste?
- Cost and latency: can it run in practice?
- Safety: did it stay within permissions?



Agentic Evaluation: **What** to Measure?

Agents generate trajectories, not just **final answer:**

- Performance depends on tools, state, and stopping criteria
- Small intermediate mistakes → compound into large failures
- Evaluation must look at **process** and **outcome** (and **cost**)
- Lots of numbers to report!

Offline vs **Online** Evaluation

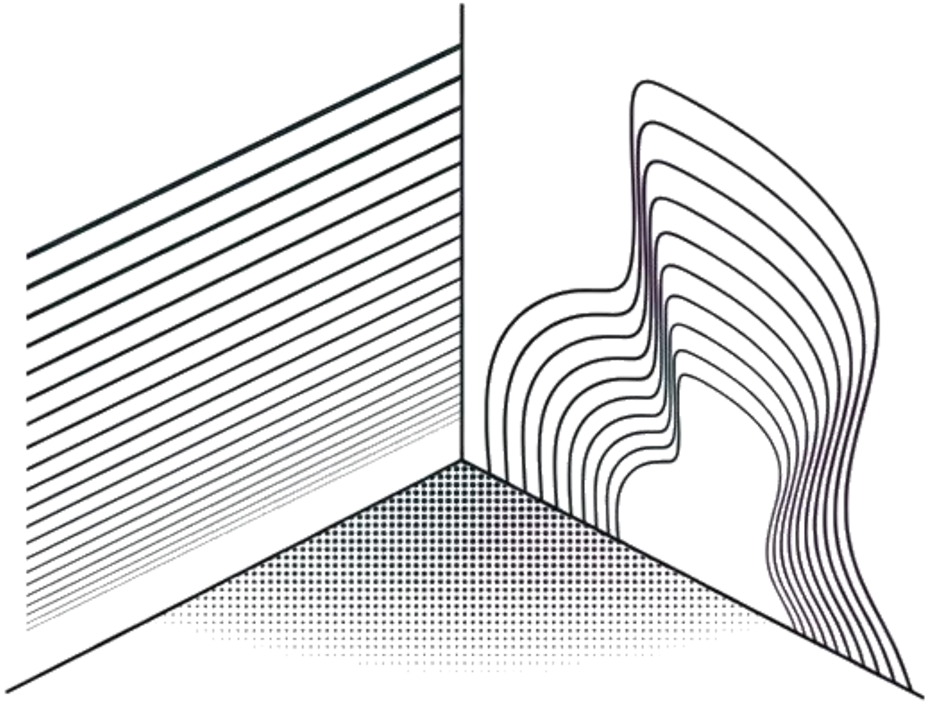
Another choice for evaluation.

- Offline: replay tasks or environments with fixed ground truth
  - Offline evaluation is cheaper and more reproducible
  - Almost always what we choose
- Online: obtain measurements in live deployments
  - Captures real user behavior and changing environments
  - More meaningful, but often too expensive

Building Evaluation Data

A checklist:

- Include easy, medium, and hard tasks (diverse **difficulty**)
- Include tasks with branching, recovery, and long horizon
- Include **tool-heavy**, **memory-heavy** cases
- Keep a held-out set so that you do not overfit on the benchmark



Metrics: **Trajectory**-Level

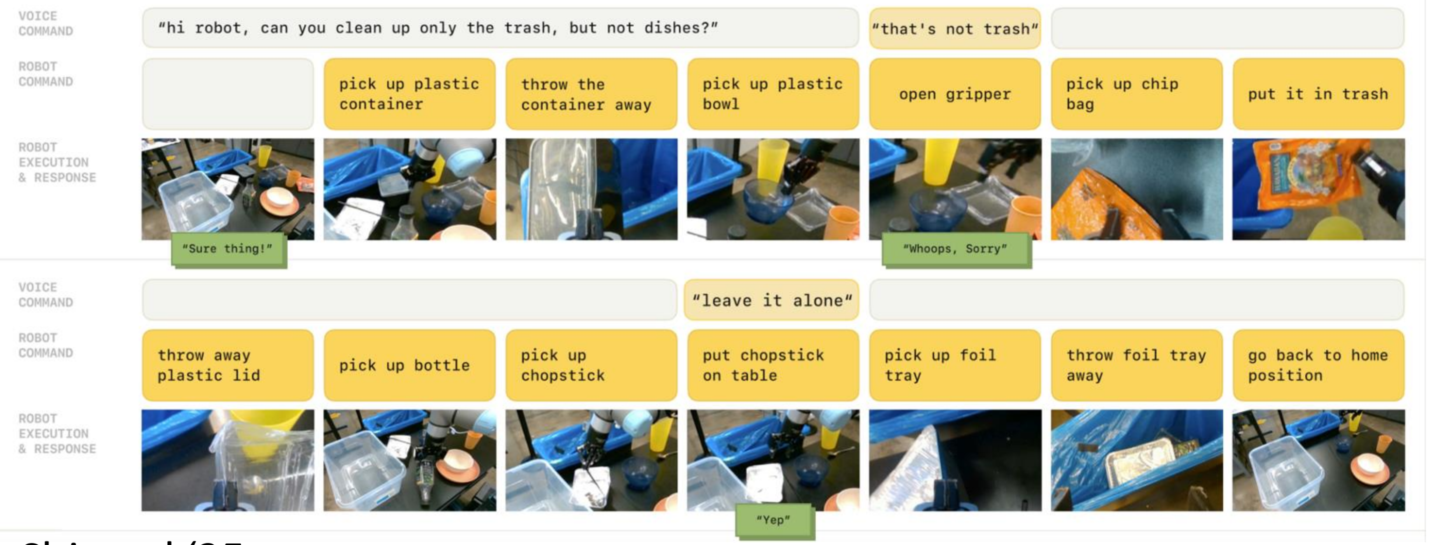
As we've mentioned, metrics are usually very rich

- Split between **outcome**-level and **trajector**y-level
  - Plus certain specific metrics for particular components.
- **Trajectory** level:
  - **Tool-call validity**: for ex., names and arguments correct?
  - **Step efficiency**: how many actions were unnecessary?
  - **Recovery**: can the agent fix intermediate failures? At what rate?
  - **Stop quality**: did it stop too early or too late

Metrics: Outcome-Level

Typically very multi-faceted.

- Outcome-level:
  - **Task completion**: did the system achieve the user's goal?
  - **Answer quality**: correctness? Rubric criteria met?
  - **Side effects**: did agent change anything it should not have?
  - **Human/user burden**: how much correction or follow-up was required?
    - Ex: robot must get advice constantly

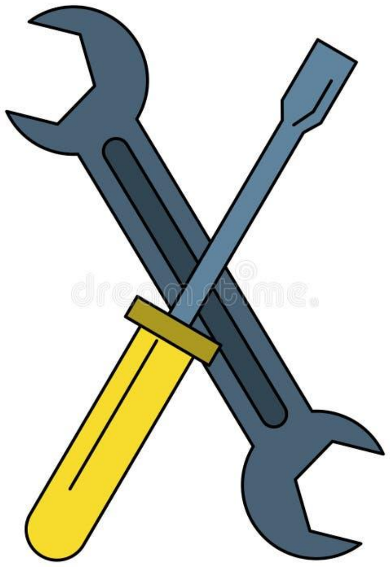


Shi et al '25

Evaluating Tool Usage

Warning: not necessarily reflected in outcome!

- Often have to check **tool selection**, **arguments**, and **output interpretation** separately
  - Why? A correct final answer can still hide broken tool usage
- **Abstention** matters: sometimes best action is not to use tool
- Useful tool measurements often report the first bad tool step



Evaluating **Memory** Components

Likewise for memories:

- Can the agent **recall important facts** later in the task?
- Can it update beliefs after **new evidence** is found?
- Can it resist **stale summaries** and context contradictions?
- For multi-session scenarios, do we obtain consistency across time and sessions?

Break & Questions

Outline

- **Review of LLM-Powered Agents**
  - Motivation, goals, differences vs classical agents / standard LMs, overall architectures and key components
- **Agentic Evaluation**
  - What to measure, offline vs online, metrics for trajectories, outcomes, tool use, memories
- **Agent Design: Case Studies**
  - Two examples: deep research and slop-aware coding. Step-by-step design and results

Case studies

Let's see two examples of the agentic benchmark construction process, from the inside (**disclaimer: from my students** :))

- (1) **LiveResearchBench**: evaluates deep research agents
- (2) **SlopCodeBench**: evaluates coding agents iteratively

Published as a conference paper at ICLR 2026

LIVERESEARCHBENCH: A LIVE BENCHMARK FOR USER-CENTRIC DEEP RESEARCH IN THE WILD

Jiayu Wang[1*], Yifei Ming[3*], Riya Dulepet[2], Qinglin Chen[3], Austin Xu[3], Zixuan Ke[3], Frederic Sala[1], Aws Albarghouthi[1], Caiming Xiong[3], Shafiq Joty[3]
[1]University of Wisconsin-Madison [2]Stanford University [3]Salesforce AI Research

A

Deep research—producing comprehe
and synthesizing information from
important frontier for agentic sys
four principles are essential: tas
realistic information needs, (2) dyna
parametric knowledge, (3) unamb
across users, and (4) multi-faceted
numerous web sources and in-dept
of these principles, often focusing
questions that hinder fair compariso
LiveResearchBench, a benchmark o
enterprise, and academia, each requi
and synthesis. Built with over 1,50
provides a rigorous basis for systema
long-form reports, we introduce **De**
content- and report-level quality, inclu
and association, consistency and d
complementary evaluation protocol
and high agreement with human j
DeepEval, we conduct a comprehe
systems, including single-agent web
agent systems. Our analysis reveals c
key system components needed to ac
code is available at: `https://gi`
`LiveResearchBench.`

SlopCodeBench: Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks

Gabriel Orlanski[1*] Devjeet Roy[2] Alexander Yun[1] Changho Shin[1] Alex Gu[3]
Albert Ge[1] Dyah Adila[1] Frederic Sala[1] Aws Albarghouthi[1]
[1]University of Wisconsin–Madison [2]Washington State University [3]MIT

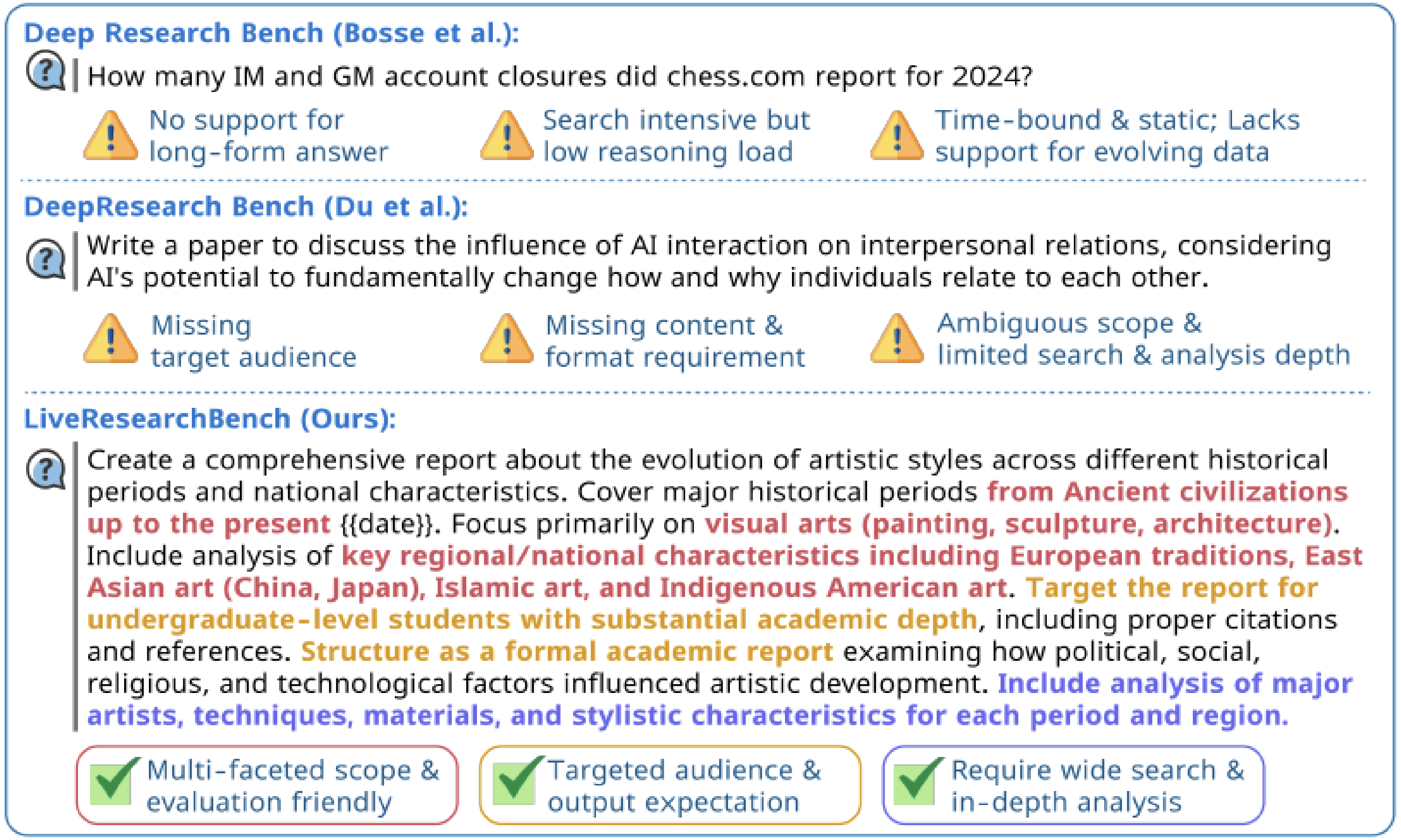
Abstract

Software development is iterative, yet agentic coding benchmarks overwhelmingly evaluate single-shot solutions against complete specifications. Code can pass the test suite but become progressively harder to extend. Recent iterative benchmarks attempt to close this gap, but constrain the agent's design decisions too tightly to faithfully measure how code quality shapes future extensions. We introduce SlopCodeBench, a language-agnostic benchmark comprising 20 problems and 93 checkpoints, in which agents repeatedly extend their own prior solutions under evolving specifications that force architectural decisions without prescribing internal structure. We track two trajectory-level quality signals: verbosity, the fraction of redundant or duplicated code, and structural erosion, the share of complexity mass concentrated in high-complexity functions. No agent solves any problem end-to-end across 11 models; the highest checkpoint solve rate is 17.2%. Quality degrades steadily: erosion rises in 80% of trajectories and verbosity in 89.8%. Against 48 open-source Python repositories, agent code is 2.2x more verbose and markedly more eroded. Tracking 20 of those repositories over time shows that human code stays flat, while agent code deteriorates with each iteration. A prompt-intervention study shows that initial quality can be improved, but it does not halt degradation. These results demonstrate that pass-rate benchmarks systematically undermeasure extension robustness, and that current agents lack the design discipline iterative software development demands.

1. LiveResearchBench: **Motivation**

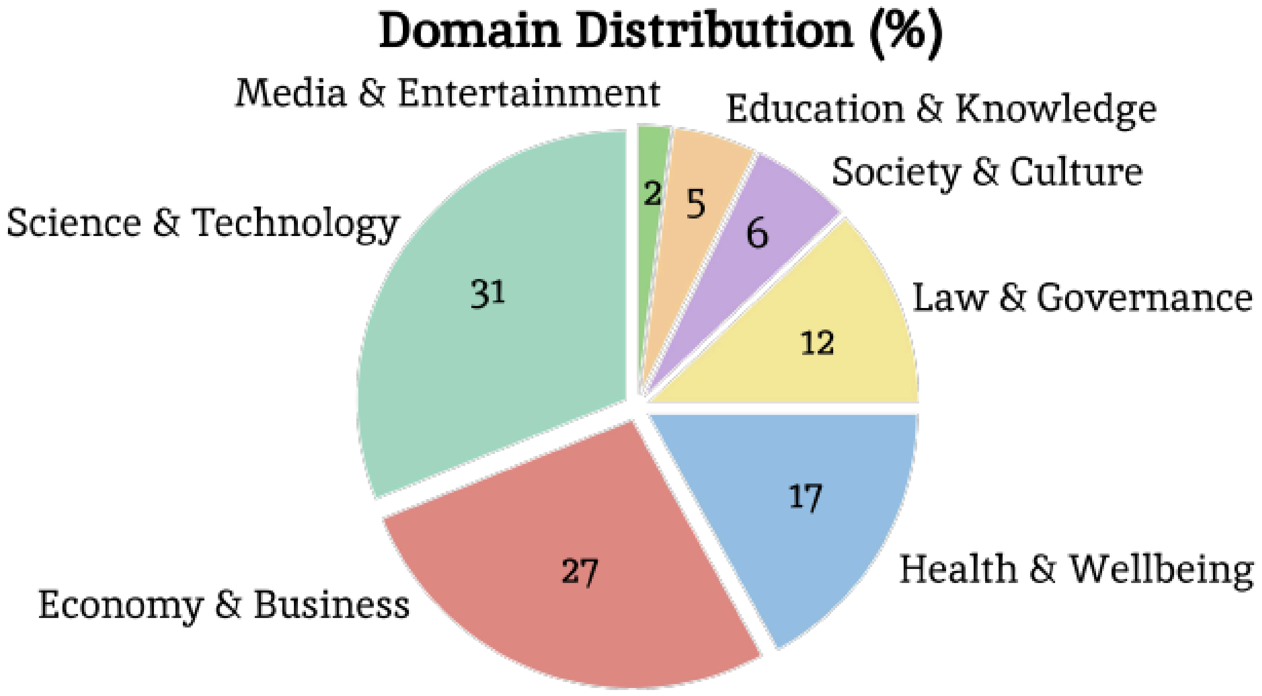
Goal: evaluate how good **deep research** agents are

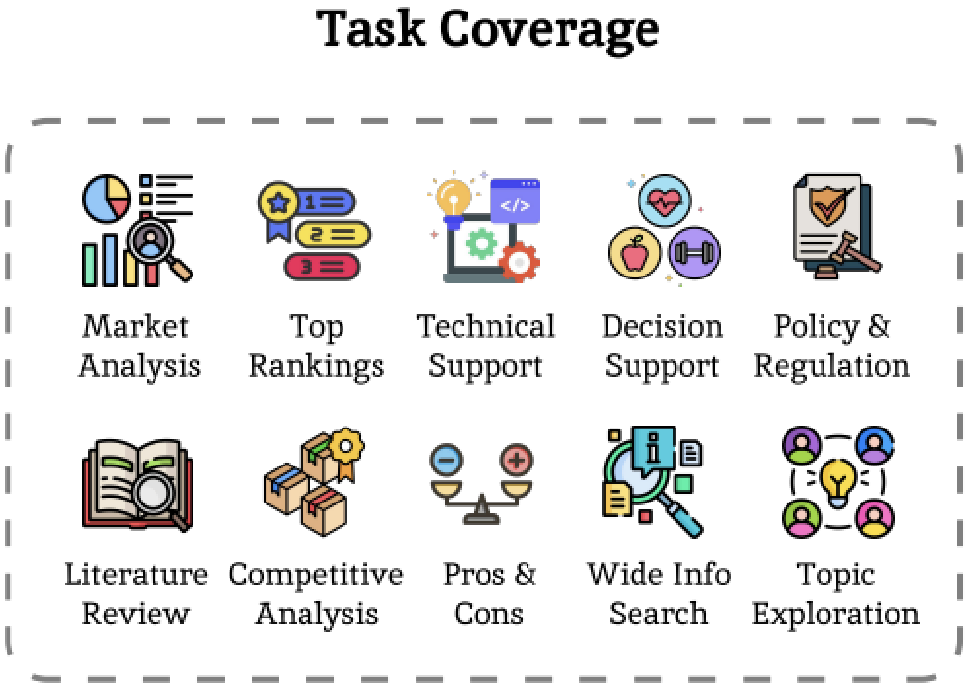
- First, study existing approaches & key aspects of use



Deep research **domains & tasks**

Next goal: what tasks and domains do users perform deep research in? Poll summary:

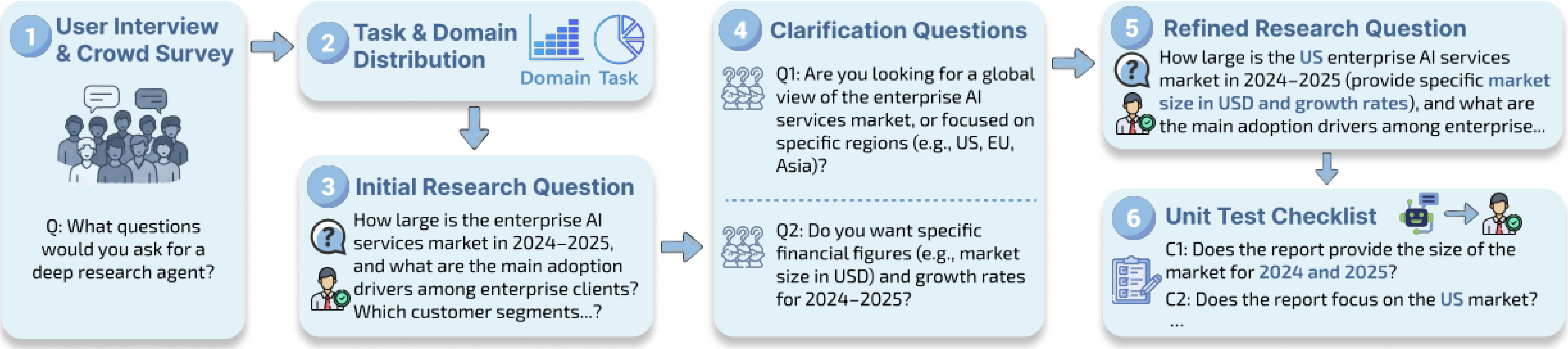




Building the deep research **queries**

Process involving polling users, crafting queries, and refining them

- Somewhat time- and resource-intensive ☹



Developing the **eval mechanism**

Now that we have our queries, how will we measure the outputs?

- We use the principles we discussed. Craft outcome and trajectory-level metrics and build rubrics from them

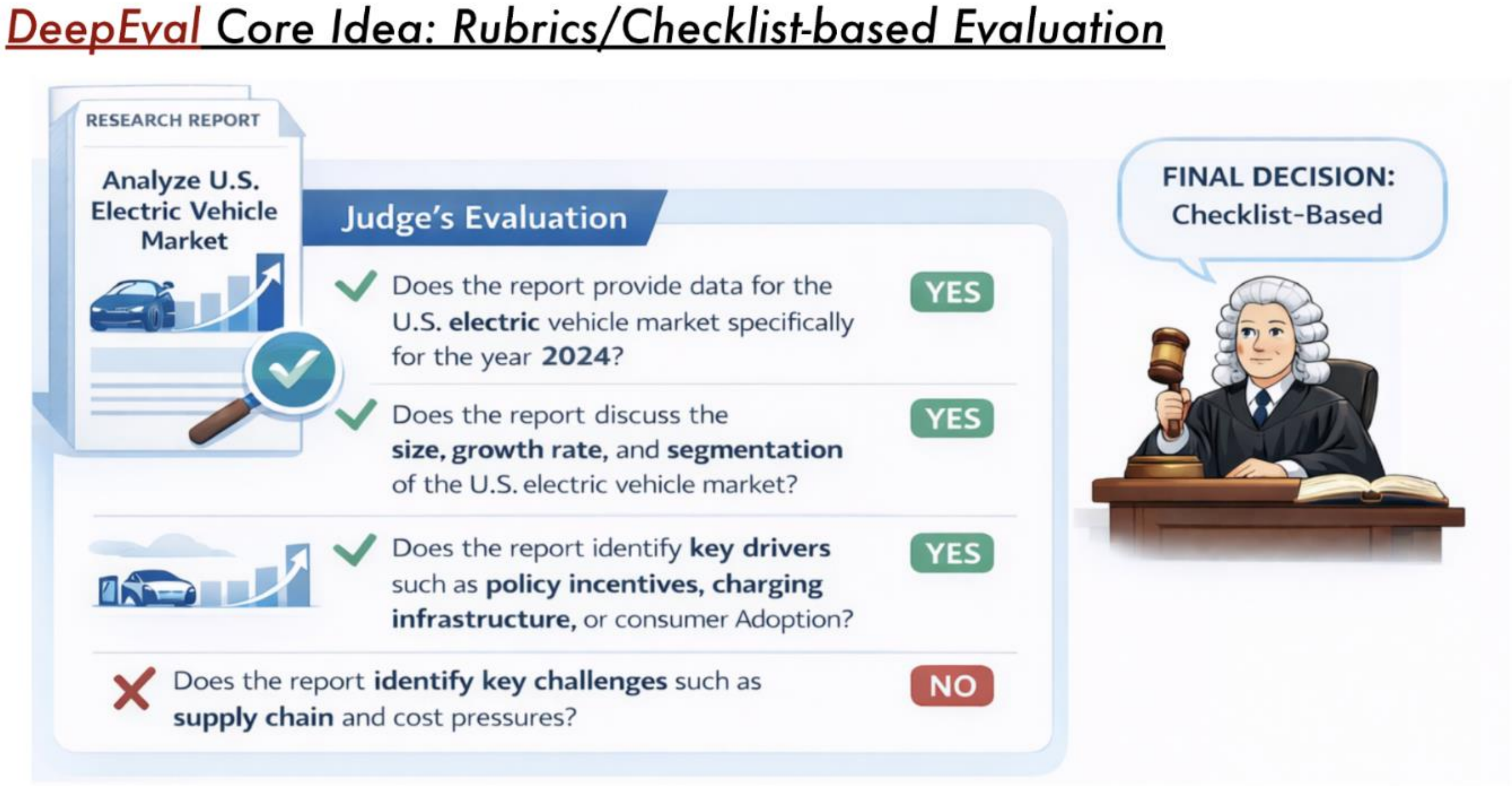
DeepEval: A comprehensive evaluation suite for deep research

- **Report-level metrics:**
  - **Presentation & organization**
  - **Factual & logical consistency**
- **Content-level metrics:**
  - **Coverage & comprehensiveness**: *are all aspects of multi-faceted query addressed?*
  - **Analysis depth**: *do report provide substantive insights beyond simple info gathering?*
  - **Citation association**: *are factual claims/numbers cited?*
  - **Citation accuracy**: *do cited sources genuinely support their claims?*

Developing the **eval mechanism**

How do we measure outputs against these metrics?

- As before, LLM-as-a-judge. Correlate with human expert raters to calibrate.



Choosing **agents** to benchmark & **results**

Three classes of deep research agents, **four** broad categories of scores to look at + averages.

Main Findings

Agent Name	Presentation & Organization	Fact & Logic Cons	Coverage &	Citation	Avg
Single-Agent with Web Search					
GPT-5	71.6	6			
GPT-4.1	66.0	6			
GPT-5-mini	61.4	6			
Gemini 2.5 Pro	51.9	7			
Claude 4 Sonnet	81.9	6			
Claude 4.1 Opus	81.6	6			
Perplexity Sonar Reasoning	82.1	7			
Perplexity Sonar Reasoning Pro	79.6	71.9	4		
Single-Agent Deep Research					
OpenAI o3 Deep Research	71.3	64.2	8		
OpenAI o4-mini Deep Research	74.3	62.3	7		
Perplexity Sonar Deep Research	**83.5**	67.4	6		
Grok-4 Deep Research	69.1	57.4	8		
Gemini Deep Research	62.1	63.0	[illegible]	[illegible]	[illegible]
Multi-Agent Deep Research					
Manus	75.0	63.1	73.3		
Grok-4 Heavy Deep Research	75.9	59.4	**89.3**		
Deerflow+ (w. GPT-5)	78.8	69.9	61.6		
Open Deep Research (w. GPT-5)	81.0	71.3	65.3		

Observations

- Multi-agent systems lead on average across all dimensions
- Single-agent with web search excel in consistency
- MAS lead in citation association & presentation

Observations

- Models produce fluent reports
- But they struggle with citation correctness and formatting

Observations

- While many agents are good at information finding
- But they struggle with deep analysis

Case Study 2: **SlopCodeBench**

Motivation: code slop is taking over the world… how do we measure it?

tldraw
@tldraw

This week we're going to begin automatically closing pull requests from external contributors. I hate this, sorry.

steveruizok opened 1 minute ago — Member

Hey all, update on the tldraw policy with regard to contributions.

For the good of the project, we're going to begin **automatically closing pull requests from external contributors**. We will of course continue to welcome issues, bug reports, and discussions. This is a temporary policy until GitHub provides better tools for managing contributions.

Getting ahead of it

Like many other open-source projects on GitHub, we've recently seen a significant increase in contributions generated entirely by AI tools. While some of these pull requests are formally correct, most suffer from incomplete or misleading context, misunderstanding of the codebase, and little to no follow-up engagement from their authors.

An open pull request represents a commitment from maintainers: that the contribution will be reviewed carefully and considered seriously for inclusion. For that commitment to remain meaningful, we need to be more selective. For now, that means closing first and selectively re-opening pull requests that are actually under consideration, with the expectation that most unsolicited submissions will not be reviewed.

Weird year coming

I first made the tldraw repository public in 2021, and I've been proud to invite public contribution to its code and have been happy to have many improving pull requests over the years. I'm sorry to shut that down, however I sincerely believe this decision is in the best interests of the project, our code, and our community. With luck, GitHub will soon roll out management features that let us open things back up.

This is going to be a weird year for programmers and open source especially. For now, whether you've contributed before, are interested in contributing in the future, or just are a friend of the project: thank you and please hang on while we all figure this stuff out.

Miles Brundage
@Miles_Brundage

Spinning up your agents to write a zillion lines of code overnight = getting a good night's slop

Lucas Beyer (bl16)
@giffmana
Follow

"live by the slop, die by the slop"

I really like this expression, I'mma steal it.

> **vik** @vikhyatk · Dec 5, 2025
> Replying to @vikhyatk
> didn't realize this was happening because the LLM put a try/catch to fall back to eager decoding.
>
> live by the slop, die by the slop

SlopCodeBench Motivation & Principles

How do we evaluate? Single snapshot is not sufficient… Slop is often a property of **iteration**

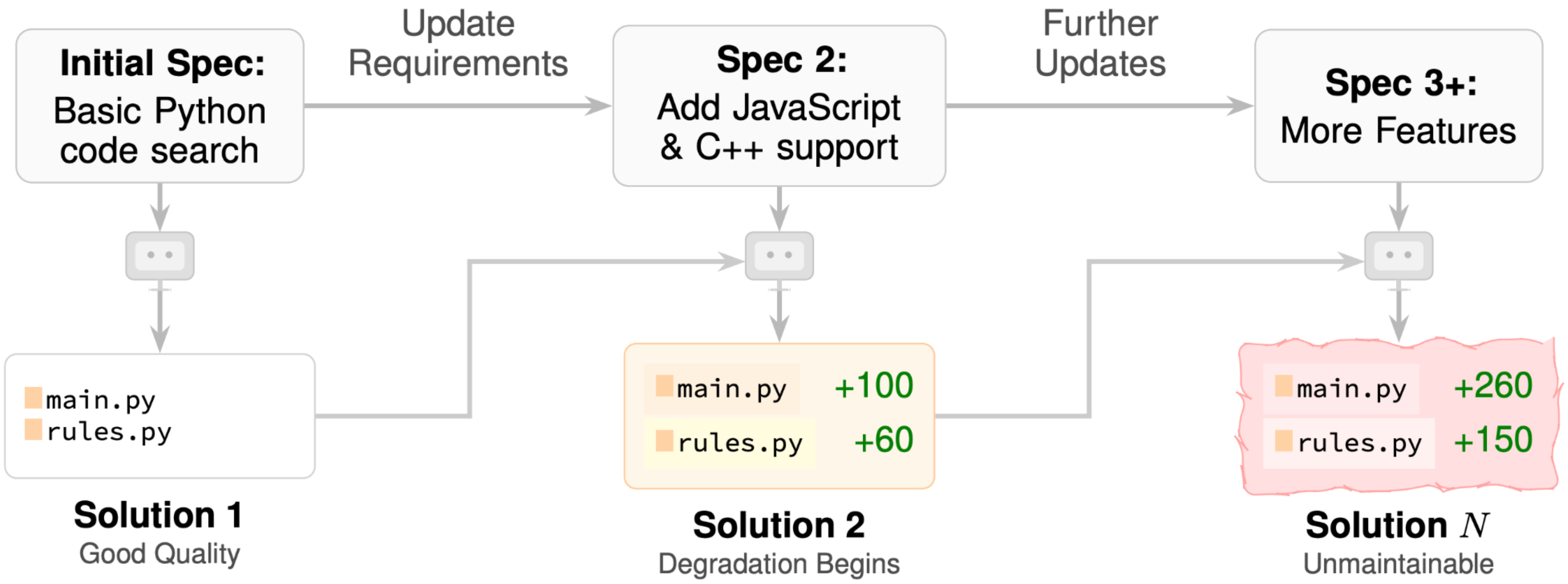
```python
def find_matches(contents, rule):
    if rule['kind'] == "exact":
        for match in contents.index(rule['pat
            print(...)
    elif rule['kind'] == "regex":
        for match in re.findall(rule['patterr
            print(...)

def main(root, rules):
    for f in root.rglob("*.py"):
        for rule in rules:
            find_matches(f.read_text(), rule)
```

```diff
 def find_matches(contents, rule):
     if rule['kind'] == "exact":
         for match in contents.index(rule['pattern']):
             print(...)
     elif rule['kind'] == "regex":
         for match in re.findall(rule['pattern'], contents):
             print(...)
+    elif rule['kind'] == "pattern":
+        tree = parse_ast(contents)
+        for node in walk(tree):
+            # ...duplicate validation, output formatting
+            print(...)
```

SlopCodeBench **Design**

SlopCodeBench instead uses iterative evaluation.



SlopCodeBench **Design**

SlopCodeBench instead uses **iterative evaluation**.

- Problems are ordered checkpoints
- Each checkpoint has a NL specification and a set of tests. Agents are given just the natural language and the existing workspace at each checkpoint.

Build a command-line code searcher for Python codebases. It takes a directory of source files and a set of **rules**, then prints **one JSON object per match** (JSON Lines) to **STDOUT**.

Supported language (by file extension only): **Python** (`.py`). Supported rule types (exactly these two): **exact match** and **vanilla regex**.

Deliverable

Write an executable named `code_search.py` that accepts:

```
python code_search.py <root_dir> --rules <rules_file> [--encoding <name>]
```

- `<root_dir>`: path to the codebase to scan.
- `--rules <rules_file>`: path to a JSON **array** of rules (see schema).
- `--encoding <name>`: optional; default `utf-8`. Files that fail to decode are **skipped**.

Output: JSON Lines to **STDOUT**, one object per match (schema below). **On success (even if zero matches):** exit code **0**.

Inputs

Extend your code searcher to support **JavaScript** and **C++** source files.

New Requirements

File type → language

Scan these extensions:

Language	Extensions
Python	`.py`
JavaScript	`.js`, `.mjs`, `.cjs`
C++	`.cc`, `.cpp`, `.cxx`, `.hh`, `.hpp`, `.hxx`

Rule language filtering

Rules may specify `"languages"` from this set: `["python", "javascript", "cpp"]`. If omitted, the rule applies to **all three**.

SlopCodeBench **Data**

40 problems and growing: mostly CLI

- Largely built manually and verified

Problem	CPs	Description	Designs Tested	Iterative Progression
circuit_eval	8	Parses .circ files, evaluates logic gates, validates circuit graphs	Graph validation, topological sort, optimization pas	Adds cycle detection, then multi-pass
code_search	5	Rule-based code searcher matching exact strings and regexes in source files	Regex engine, file tra tern matching	
dag_execution	3	Executes task pipelines from a spec file with dependency ordering	DSL parsing, DAG expression evaluation	
database_migration	5	Applies JSON-defined schema migrations to SQLite	DDL generation, sch tion, transaction rollb	
dynamic_buffer	4	Infers column transforms from I/O samples and generates a preprocessing module	Code generation, buffers, inference	
dyn._config_svc.	4	REST API storing versioned JSON configs with deep-merge semantics	Immutable versionin erarchies, cycle detec	
etl_pipeline	5	Parses and normalizes ETL step definitions from JSON	Pipeline DSL, expres tion, schema checkin	
eve_industry	6	Generates manufacturing recipes from EVE Online's static data export	Compressed data par chical domain data	
eve_jump_planner	3	Plans jump freighter routes with fatigue and isotope calculations	Physics simulation, m optimization	
eve_market_tools	4	REST API building price books from gzipped market-order CSVs	Data ingestion, in-me ing, outlier filtering	
eve_route_planner	3	Plans ship routes with warp physics and gate travel	Warp simulation, gra region-lock rules	

				and ship selection
execution_server	6	HTTP API spawning processes, capturing output, tracking run statistics	Process management, timeout/signal handling	Adds encoding preservation, then persistent reusable environments with concurrency modes
file_backup	4	Simulates backup scheduling from YAML with timezone-aware cron rules	Cron scheduling, glob exclusion, event streaming	Evolves to incremental backups with change detection and efficient archive rewriting
file_merger	4	Merges multiple CSVs by aligning schemas and sorting on composite keys	Schema union, type coercion, null handling	Adds type coercion strategies, then nested types (struct, array, map) and JSON fields
file_query_tool	5	SQL engine querying a folder of CSVs with SELECT, JOIN, WHERE, GROUP BY	SQL parsing, join algorithms, aggregation	Extends with HAVING and LIMIT/OFFSET, then CTEs, recursive CTEs, and correlated subqueries
layered_config	4	Merges layered YAML/JSON inputs into a canonical experiment spec	Multi-layer merge, anchor resolution, conflict detection	Adds schema validation and hardware profiles, then per-run variant synthesis
log_query	5	SQL-like query language for filtering and projecting over NDJSON logs	Grammar parsing, boolean expressions, null propagation	Extends with nested field access, then nested subqueries and correlated lookups
metric_xform_lang	5	DSL for metric transformations with filtering, grouping, and windowing	DSL interpreter, window functions, aggregation	Adds time-based windows and statistics, then versioned DSL with resumption state
migrate_configs	5	Applies transformation rules to config files across JSON, YAML, TOML, INI	Multi-format parsing, dot-path navigation	Adds glob-based file scoping, then validation rules with early-exit semantics
trajectory_api	5	REST API storing and querying agent experiment trajectories	Request validation, indexing, full-text search	Adds time-range filtering, then sandboxed tool execution and file-level checkpointing

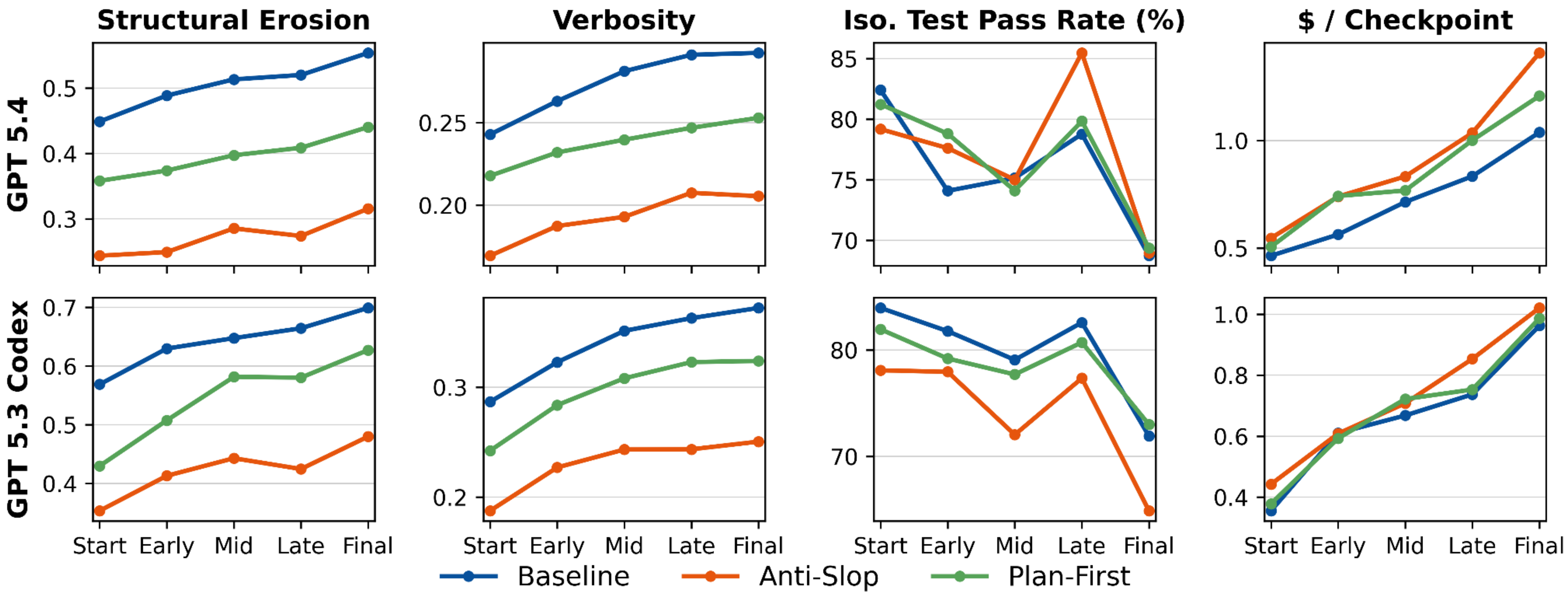
SlopCodeBench **Metrics**

Two **core metrics**:

- **Structural Erosion**
  - Deeper architectural problems indicative of haphazardly solving the problem to achieve a local optima
- **Verbosity**
  - Surface-level erosion through redundant or unnecessary code.

SlopCodeBench **Findings**

Can obtain lots of interesting insights!



https://www.scbench.ai/

Thank You!