Reliability in AI (Agentic) Systems

Ion Stoica

databricks Anyscale Arena OPAQUE

What is reliability?

"*The ability of a **system** or component to function under stated conditions for a specified period of time.*" NIST

Properties of a reliable systems

- Accuracy & correctness
- Consistency & predictability
- Robustness
- Safety

Reliability, critical to AI success

Pro

'The most advanced organizations aren't failing less; they're seeing failures sooner': Many firms are already having to roll back AI customer service tools

News By Craig Hale published May 14, 2026

Customer service AI is more about governance, report says

Stanford, CA 94305, USA

LangChain Products Learn Docs Company Pricing Try LangSmith Get a demo

State of Agent Engineering

We surveyed over 1,300 professionals — from engineers and product managers to business leaders and executives — to uncover the state of AI agents. Let's dive into the data and break down how AI agents are being used (or not) today.

VB

VB Event

AI agents are entering their rebuild era as enterprises confront the reliability problem

VB Staff

8:00 am, PT, May 29, 2026

Our own findings

Only agents in production

- 86 deployments
- 20 case studies
- 5 domains

Measuring Agents in Production

Melissa Z. Pan [1 *] **Negar Arabzadeh** [1 *] **Riccardo Cogo** [2] **Yuxuan Zhu** [3] **Alexander Xiong** [1] **Lakshya A Agrawal** [1] **Huanzhi Mao** [1] **Emma Shen** [1] **Sid Pallerla** [1] **Liana Patel** [4] **Shu Liu** [1] **Tianneng Shi** [1] **Xiaoyuan Liu** [1] **Jared Quincy Davis** [4] **Emmanuele Lacavalla** [2] **Alessandro Basile** [2] **Shuyi Yang** [2] **Paul Castro** [5] **Daniel Kang** [3] **Koushik Sen** [1] **Dawn Song** [1] **Joseph E. Gonzalez** [1] **Ion Stoica** [1] **Matei Zaharia** [1 *] **Marquita Ellis** [5 *]
[1]UC Berkeley [2] Intesa Sanpaolo [3] UIUC [4] Stanford University [5] IBM Research

Abstract

LLM-based agents already operate in production across many industries, yet we lack an understanding of what technical methods make deployments successful. We present the first systematic study of **M**easuring **A**gents in **P**roduction, MAP, using first-hand data from agent developers. We conducted 20 case studies via in-depth interviews and surveyed 86 deployed systems practitioners across 26 domains. We investigate why organizations build agents, how they build them, how they evaluate them, and their top development challenges. Our study finds that production agents are built using simple, controllable approaches: 68% execute at most 10 steps before human intervention, 70% rely on prompting off-the-shelf models instead of weight tuning, and 74% depend primarily on human evaluation. Reliability (consistent correct behavior over time) remains the top development challenge, which practitioners currently address through systems-level design. MAP documents the current state of production agents, providing the research community with visibility into deployment realities and underexplored research avenues.

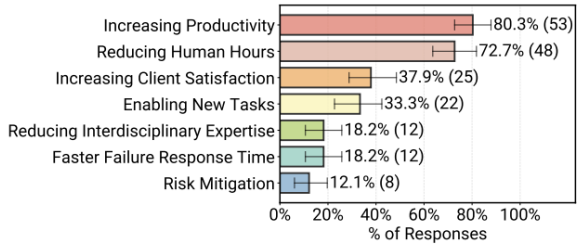


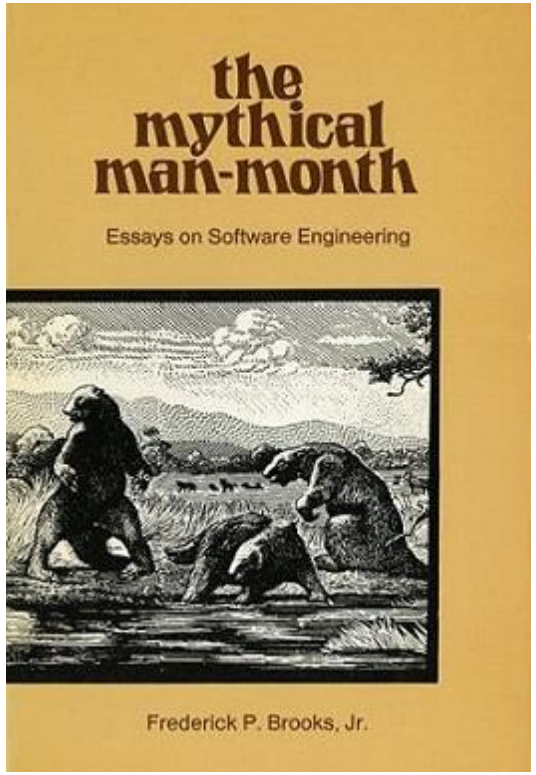
Figure 1. Reasons practitioners build and deploy AI agents (N=66). The question is multi-select, so proportions do not sum to 1. Error bars indicate 95th percentile intervals estimated from 1,000 bootstrap samples with replacement.

have gained substantial research interest, demonstrating potential in areas such as drug discovery and scientific discovery (Huang et al., 2025; Novikov et al., 2025; Lu et al., 2024). Industry is also now deploying agents in domains central to society: finance, healthcare, and education (IACPM, McKinsey, 2025; UHS, Inc., 2025; Linsenmayer, 2025).

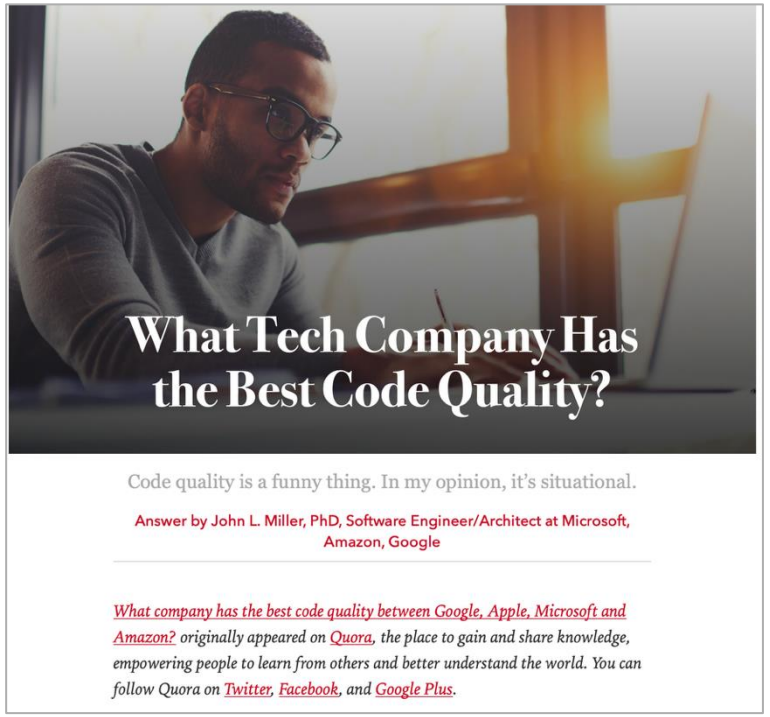
Despite widespread excitement about agents, studies show agent deployments often fail or underdeliver (Xue et al., 2025; Reuters Staff, 2025; Shome et al., 2026). The stark contrast between the potential of agents and their failures raises the fundamental question of what enables successful

Similar to software systems

Productizing takes ~10-50x more than prototyping



Prototype → production: 9x



https://apple.news/Ax8SMUhGqTGeblSjKsNrp1Q

Prototype → production: > 50x

Main goal of productization: make system reliable!

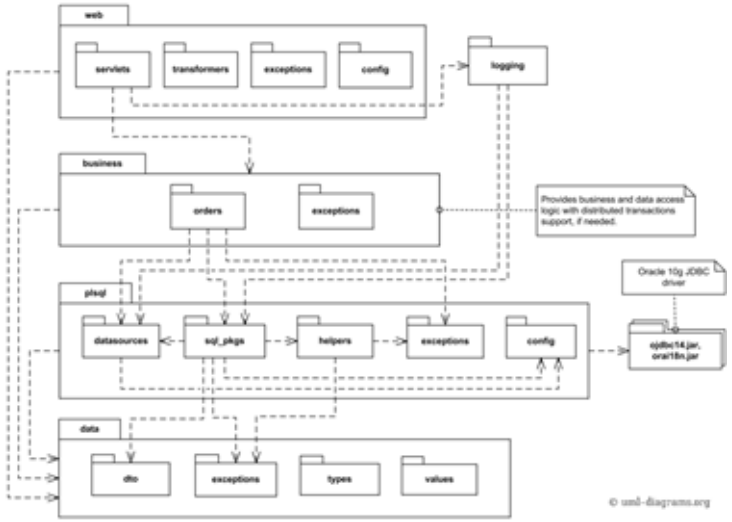
What does it take to productize a feature?

- Testing & Debugging
- Compliance/governance
- Deployment & Observability
- …

Testing & debugging a software system

A software system consists of
many components

- E.g., functions, objects, actors, tasks, …



Each component has a clear specification, interface

- E.g., define expected output given input

Testing: check each component satisfies its spec

Testing & **debugging** a software system

Debugging:

- Identify component and code violating its spec
- Fix problem by changing faulty code
- Repeat process until all tests pass

Debugging an AI agent much harder

Agent: a system in which an LLM acts as a central reasoning engine to accomplish a task by

- repeatedly interacting with environment…
- using external tools…
- until stopping condition is true
  (e.g., task done, error, budget exceeded)

Debugging an AI agent much harder

LLMs

- Rarely have clear and complete specification → hard to know an error even occurs
- Black box → equivalent to debugging a program by only seeing its output

Agent Loop → Multi-Agent Systems

Multi-Agent Systems (MAS)

Many agents coordinating to accomplish a task

Context isolation
Each sub-agent has its own window; it only sees what it needs.

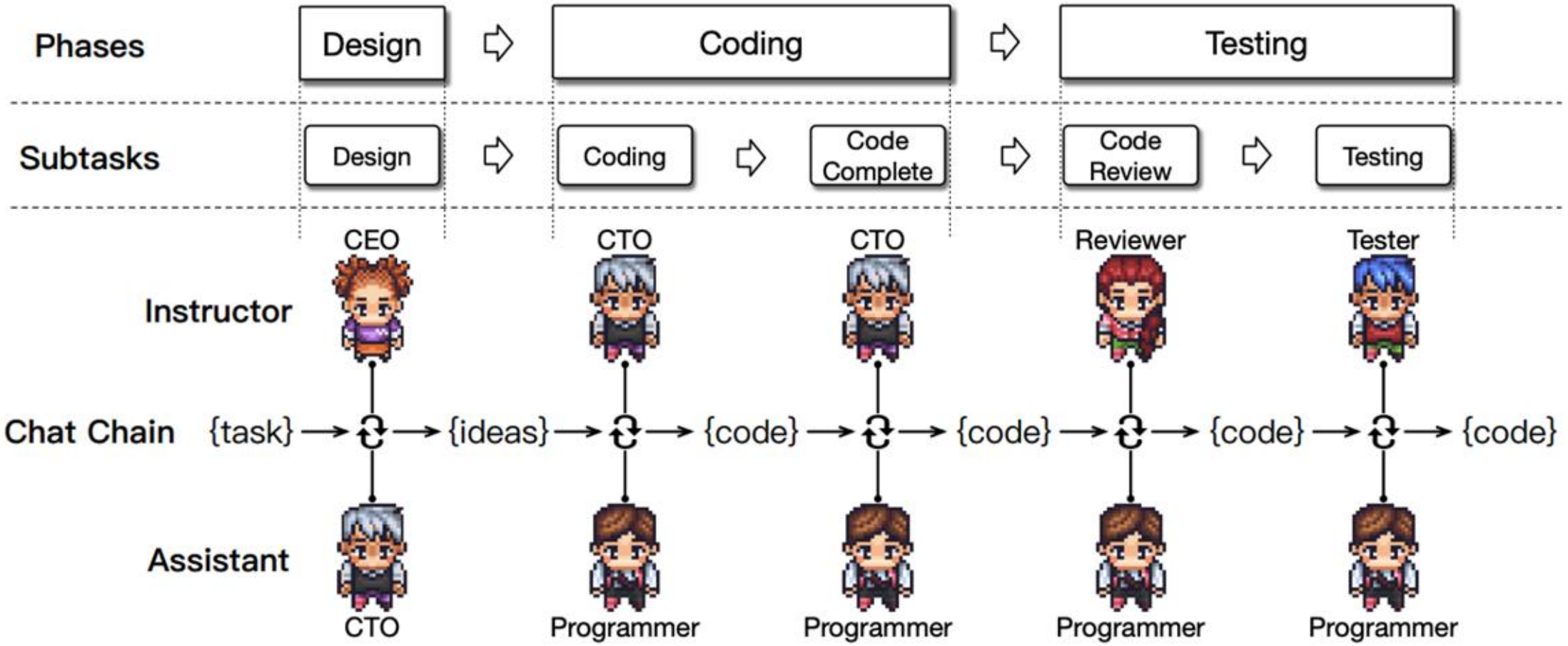
Specialization
Each sub-agent can have a narrow role, a tailored prompt, a curated tool set, and even a different model..

...

Parallelism
Sub-agents run independent subtasks in parallel to reduce running time.

Modularity
Each agent is a component you can build, test, swap, and reuse, so easier to reason about than one monolithic agent.

Example: ChatDev



Debugging MAS

First step: identify errors!

Two projects:

- MAST
- ATLAS

MAST

(**M**ulti **A**gent **S**ystem failure **T**axonomy)

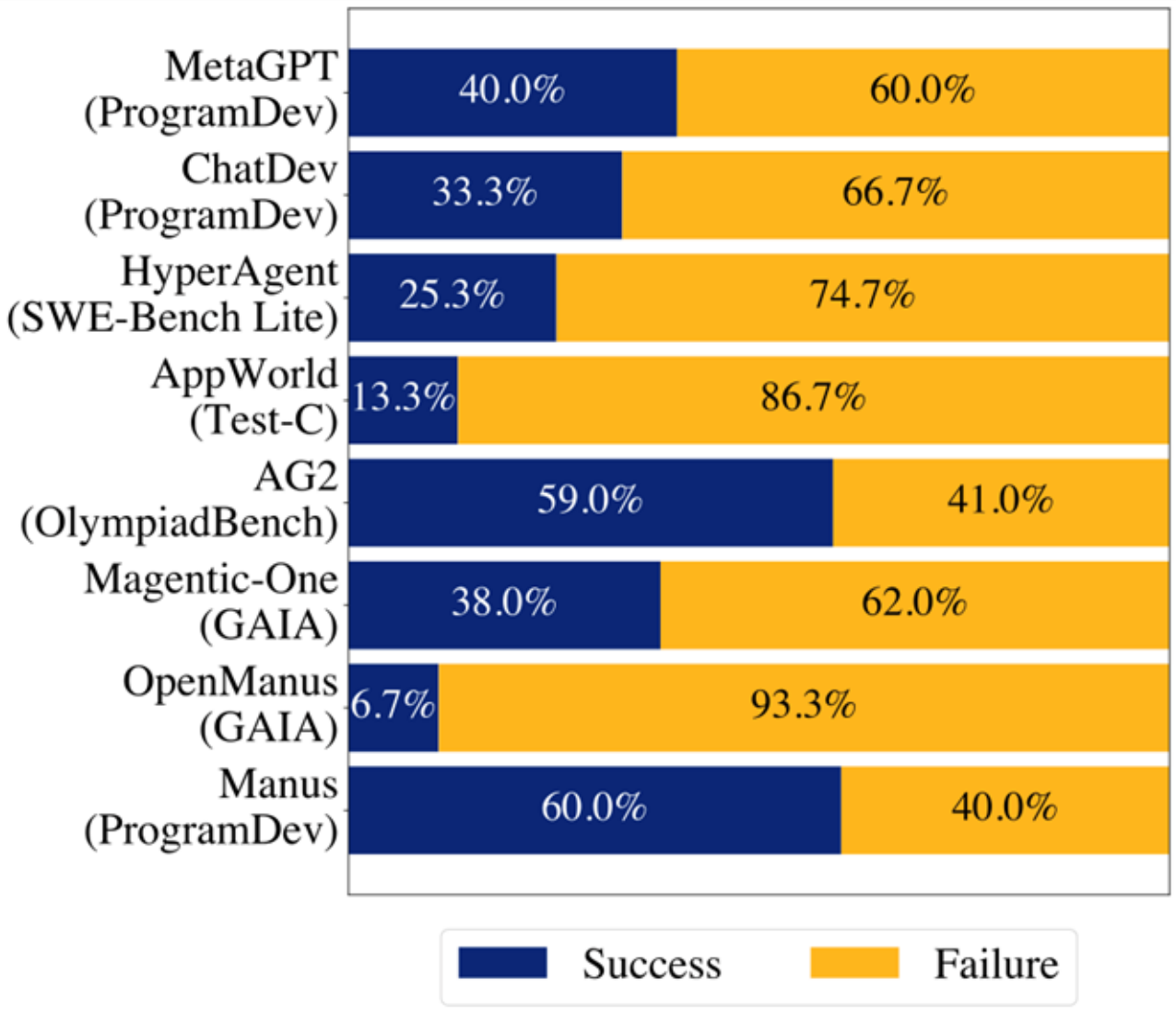
MAS fail >50% on hard benchmarks

Why Do Multi-Agent LLM Systems Fail?

Mert Cemri[1*] **Melissa Z. Pan**[1*] **Shuyi Yang**[2*] **Lakshya A Agrawal**[1] **Bhavya Chopra**[1]
Rishabh Tiwari[1] **Kurt Keutzer**[1] **Aditya Parameswaran**[1] **Dan Klein**[1]
Kannan Ramchandran[1] **Matei Zaharia**[1] **Joseph E. Gonzalez**[1] **Ion Stoica**[1]
[1]UC Berkeley [2]Intesa Sanpaolo *Equal Contribution

Abstract

Despite enthusiasm for Multi-Agent LLM Systems (MAS), their performance gains on popular benchmarks are often minimal. This gap highlights a critical need for a principled understanding of why MAS fail. Addressing this question requires systematic identification and analysis of failure patterns. We introduce `MAST-Data`, a comprehensive dataset of 1600+ annotated traces collected across 7 popular MAS frameworks. `MAST-Data` is the first multi-agent system dataset to outline the failure dynamics in MAS for guiding the development of better future systems. To enable systematic classification of failures for `MAST-Data`, we build the first **M**ulti-**A**gent **S**ystem Failure **T**axonomy (`MAST`). We develop `MAST` through rigorous analysis of 150 traces, guided closely by expert human annotators and validated by high inter-annotator agreement ($\kappa = 0.88$). This process identifies 14 unique modes, clustered into 3 categories: (i) system design issues, (ii) inter-agent misalignment, and (iii) task verification. To enable scalable annotation, we develop an LLM-as-a-Judge pipeline with high agreement with human annotations. We leverage `MAST` and `MAST-Data` to analyze failure patterns across models (GPT4, Claude 3, Qwen2.5, CodeLlama) and tasks (coding, math, general agent), demonstrating opportunities for improvement through better MAS design. Our analysis provides insights revealing that identified failures require more sophisticated solutions, highlighting a clear roadmap for future research. We publicly release our comprehensive dataset (`MAST-Data`), the `MAST`, and our LLM annotator to facilitate widespread research and development in MAS.[1 2]



Why do Multi-Agent Systems Fail?

Goal

Systematically **identify**, **classify**, and **analyze** the reasons why MAS fail

How do we do it?

Manually inspected **150+** traces to build a failure taxonomy

LLM judge pipeline to evaluate **1000+** more traces (MAD)

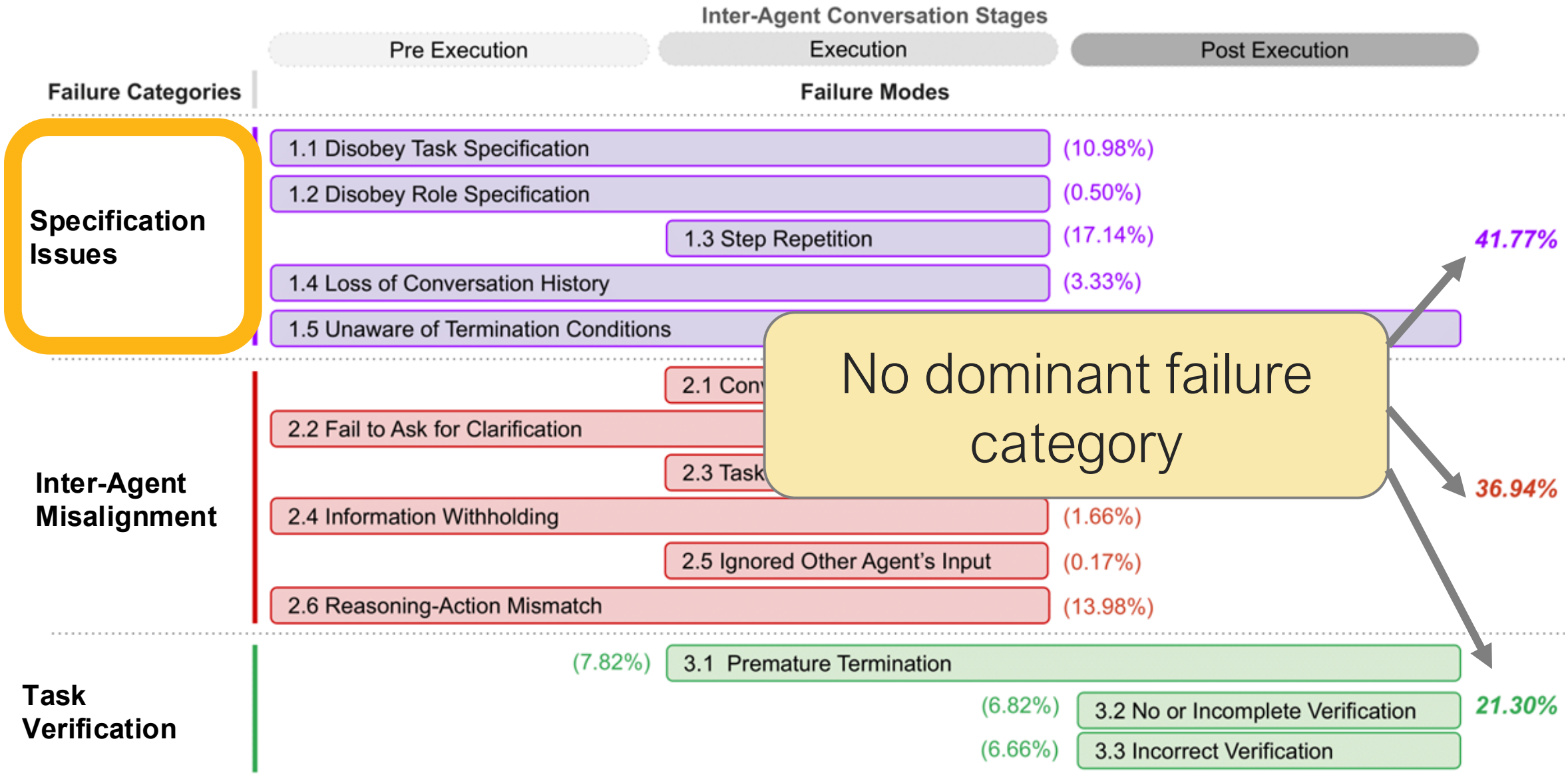
- 95% agreement with humans

Open-sourced taxonomy (**MAST**) and dataset (**MAD**)

Agentic frameworks:

- AppWorld
- HyperAgent
- AG2
- ChatDev
- MetaGPT
- MagenticOne
- OpenManus

MAST: Multi-Agent Systems failure Taxonomy



FC1: Specification issues

Failures originate from system design decisions, and poor or ambiguous prompt specifications.

1. Disobey task specifications
2. Disobey role specifications
3. Step repetition
4. Conversation loss
5. Agents unaware of termination conditions

CEO

... **Once we all** have expressed our opinion(s) and **agree** with the results of the discussion …

Ok, we will do…
<INFO> Website <\INFO>
…
<end of phase>

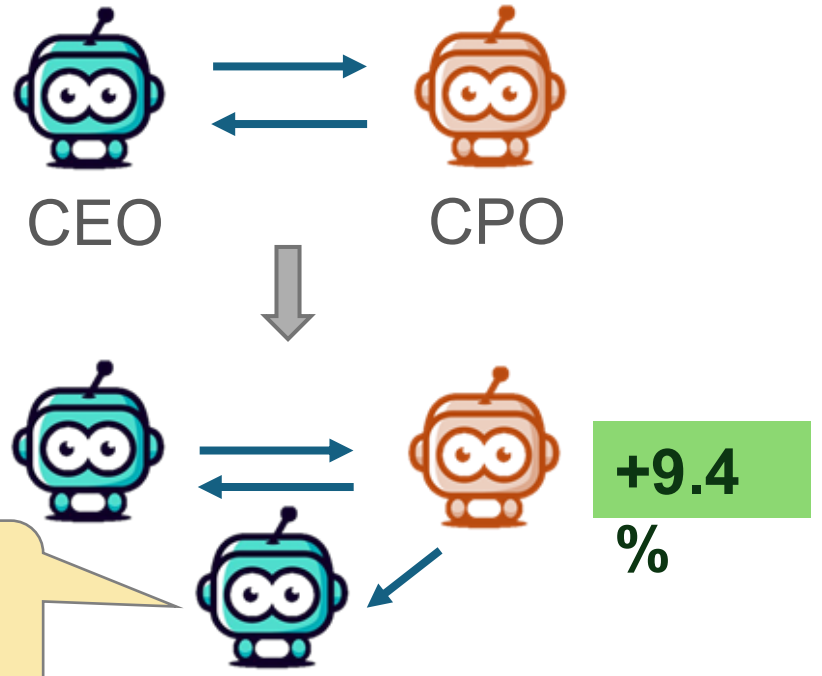
CPO

CPO doesn't wait for CEO to make decision!

FC1: Specification issues

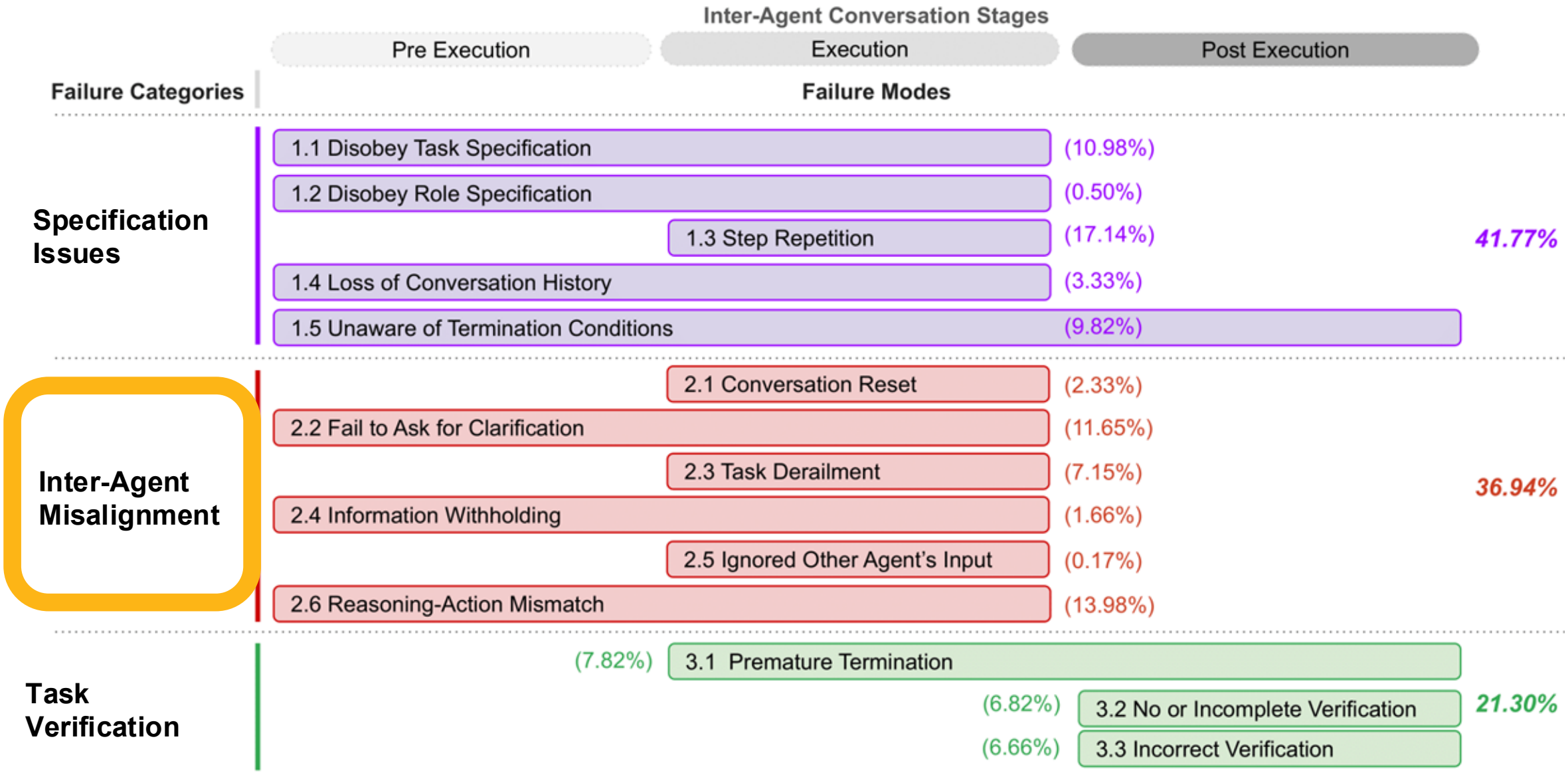
Underlying LLMs not the only limitation!

- Prompt specification
- System design (specification)



Improving MAS design → improve perf with same model

MAST: Multi-Agent Systems Failure Taxonomy



FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch

FC2: Inter-agent misalignment

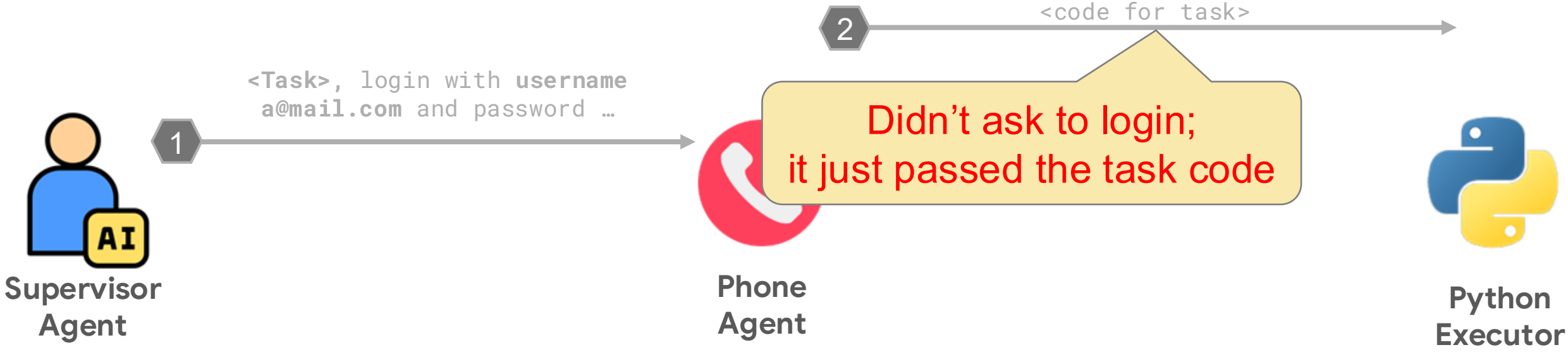
Failures in agent coordination to achieve a common goal.

- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch

FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

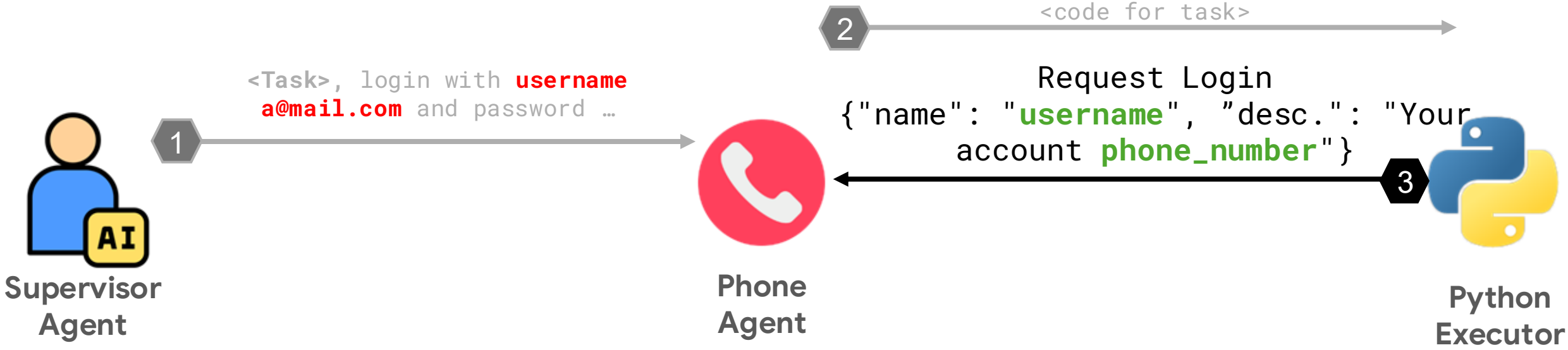
- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch



FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

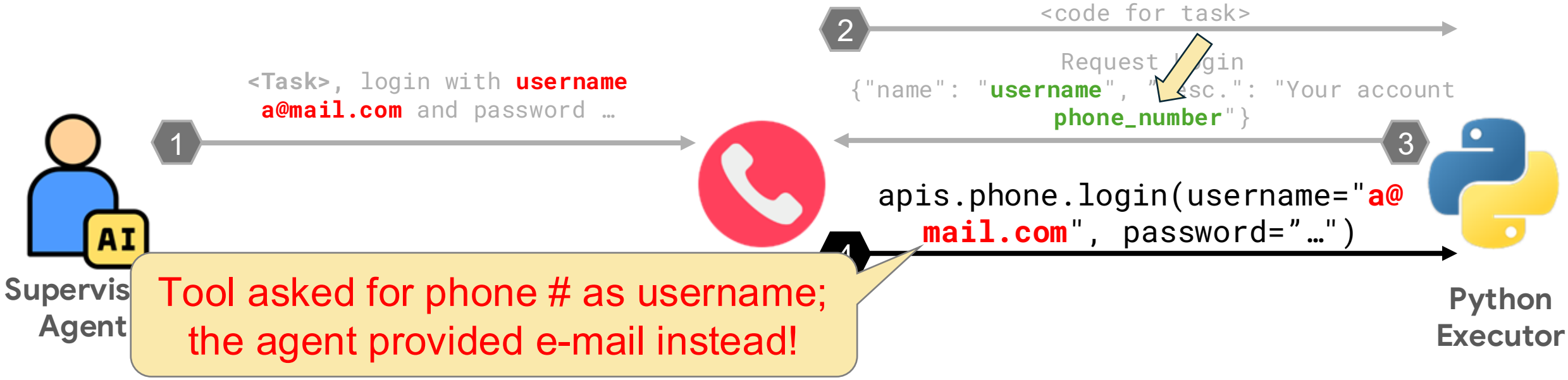
- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch



FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

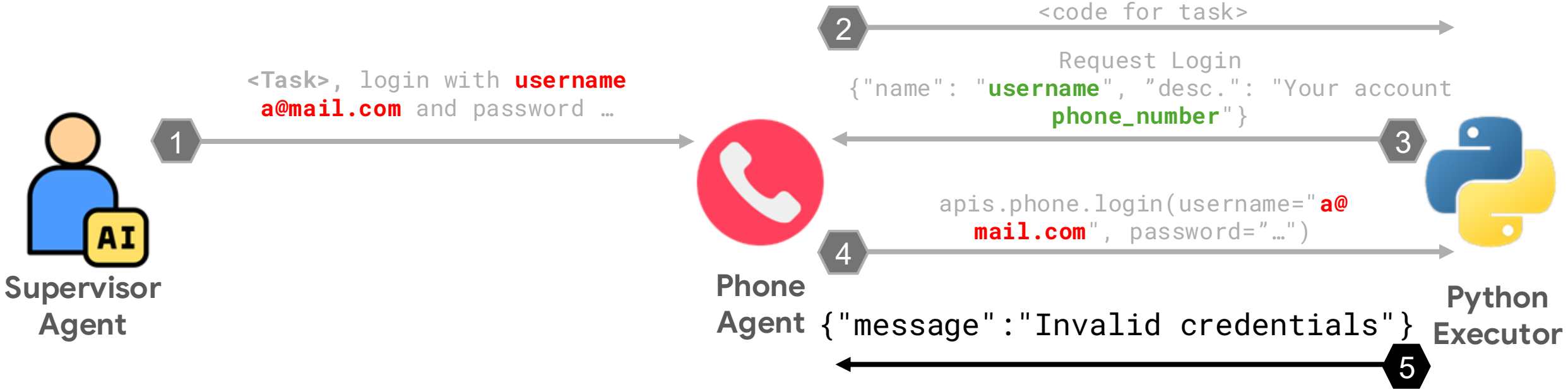
- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch



FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

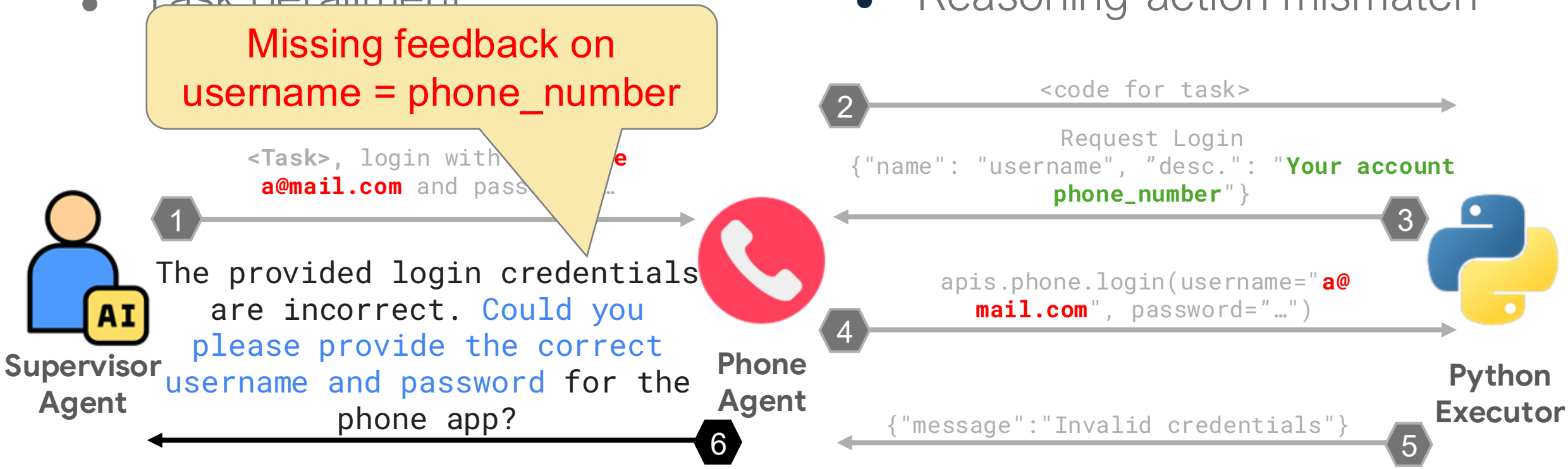
- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch



FC2: Inter-agent misalignment

Failures in agent coordination to achieve a common goal.

- Conversation reset
- Failure to ask for clarification
- Task derailment
- Information withholding
- Ignore input
- Reasoning-action mismatch

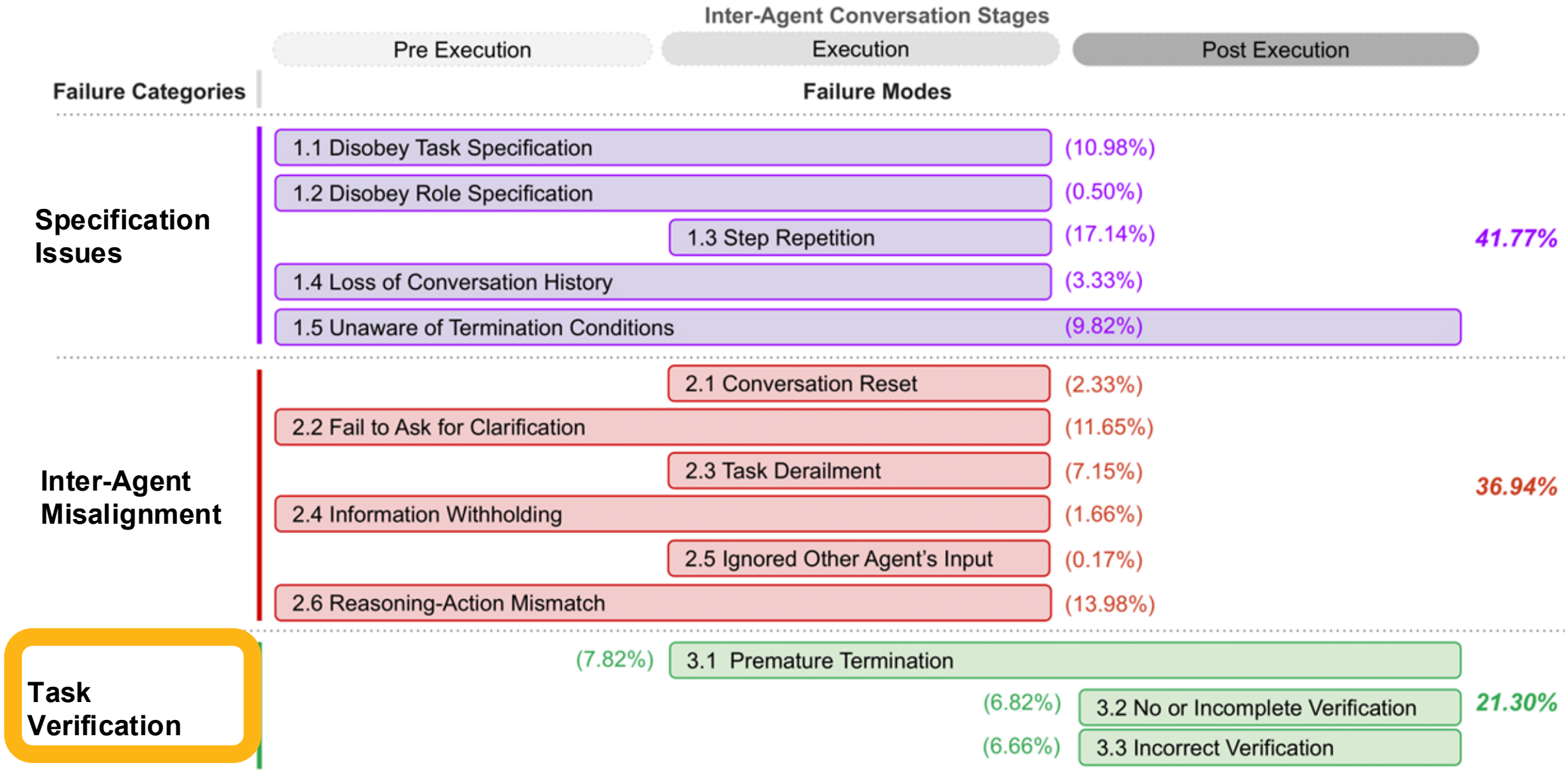


FC2: Inter-agent misalignment

Solutions focused on protocols only are often insufficient to prevent FC2 failures!

MAS demands deeper “**social reasoning**” abilities from agents

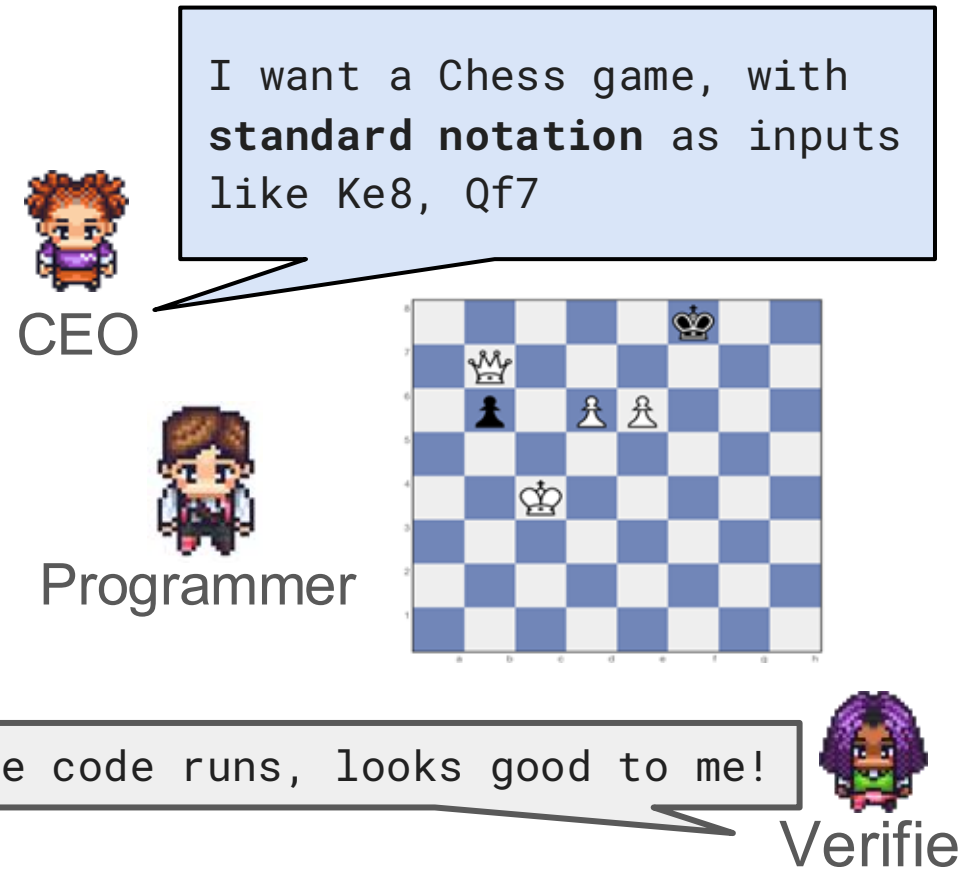
MAST: Multi-Agent Systems Failure Taxonomy



FC3: Task verification

Failures related to **quality control** of the output

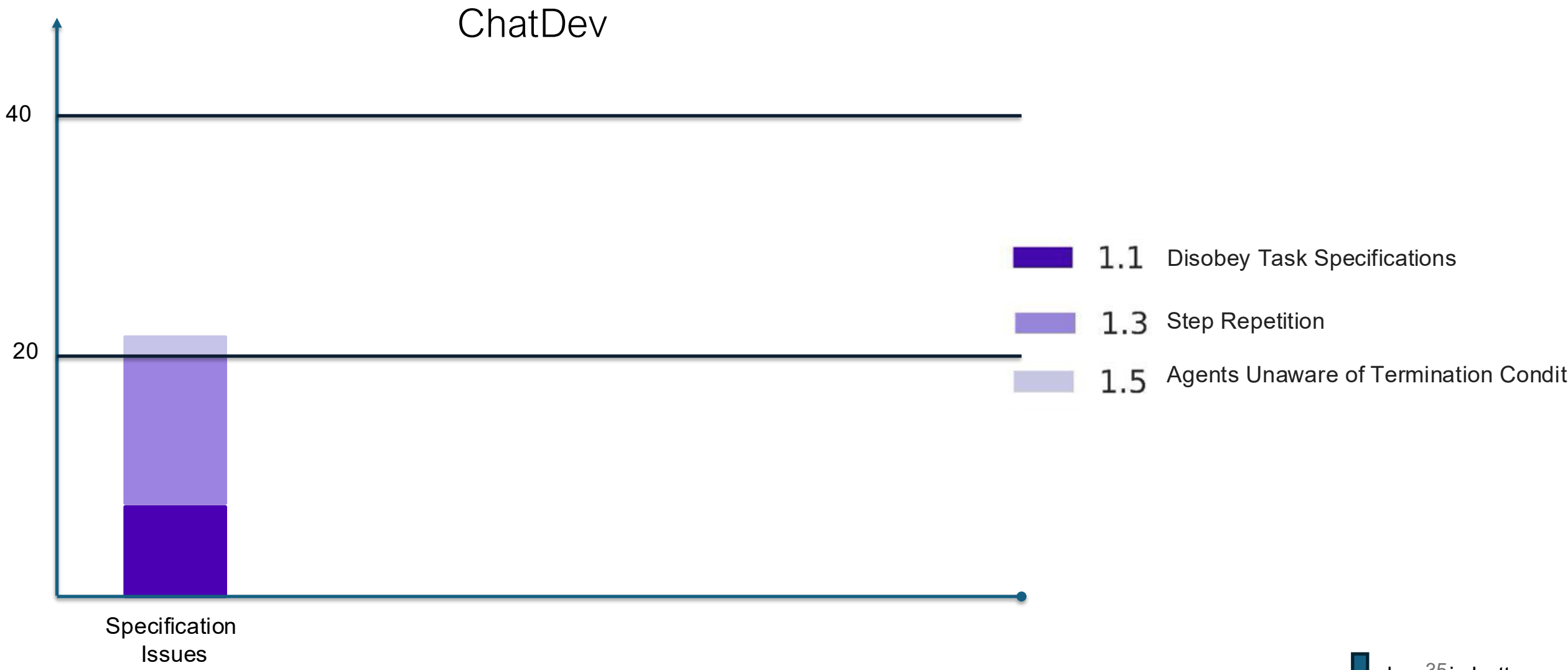
1. Premature termination
2. No or incomplete verification
3. Incorrect verification



FC3: Task verification

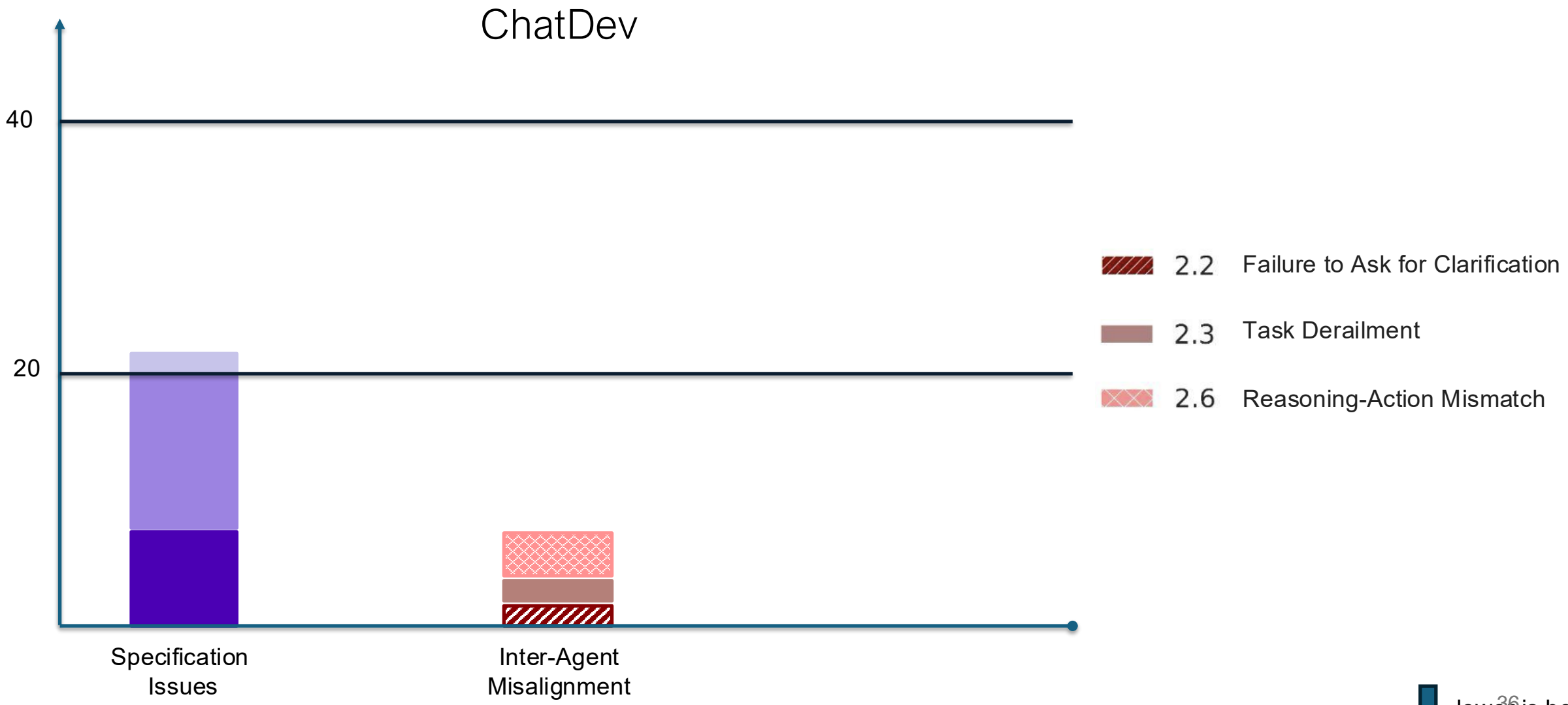
Unit testing & multi-level verification is needed!

Understanding failures profile

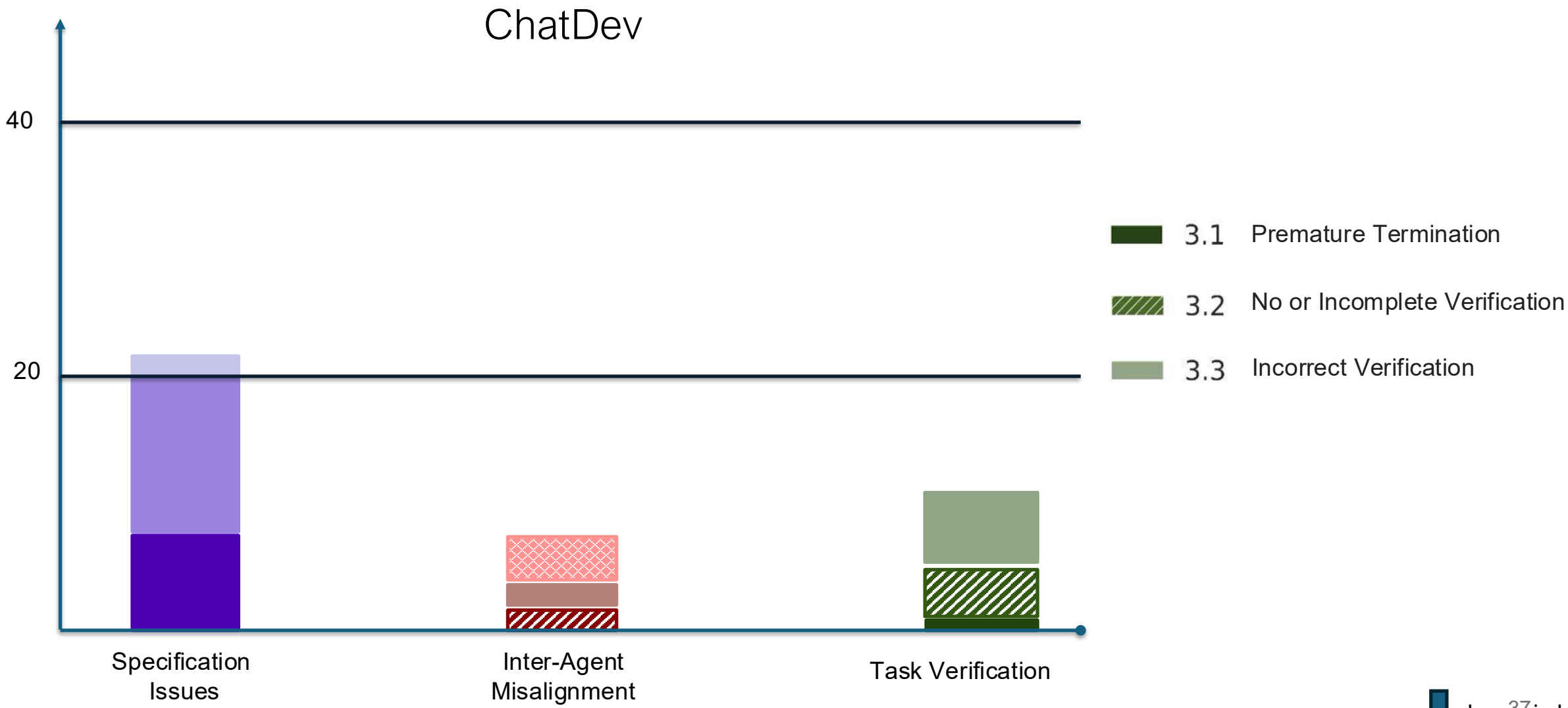


lower is better

Understanding failures profile

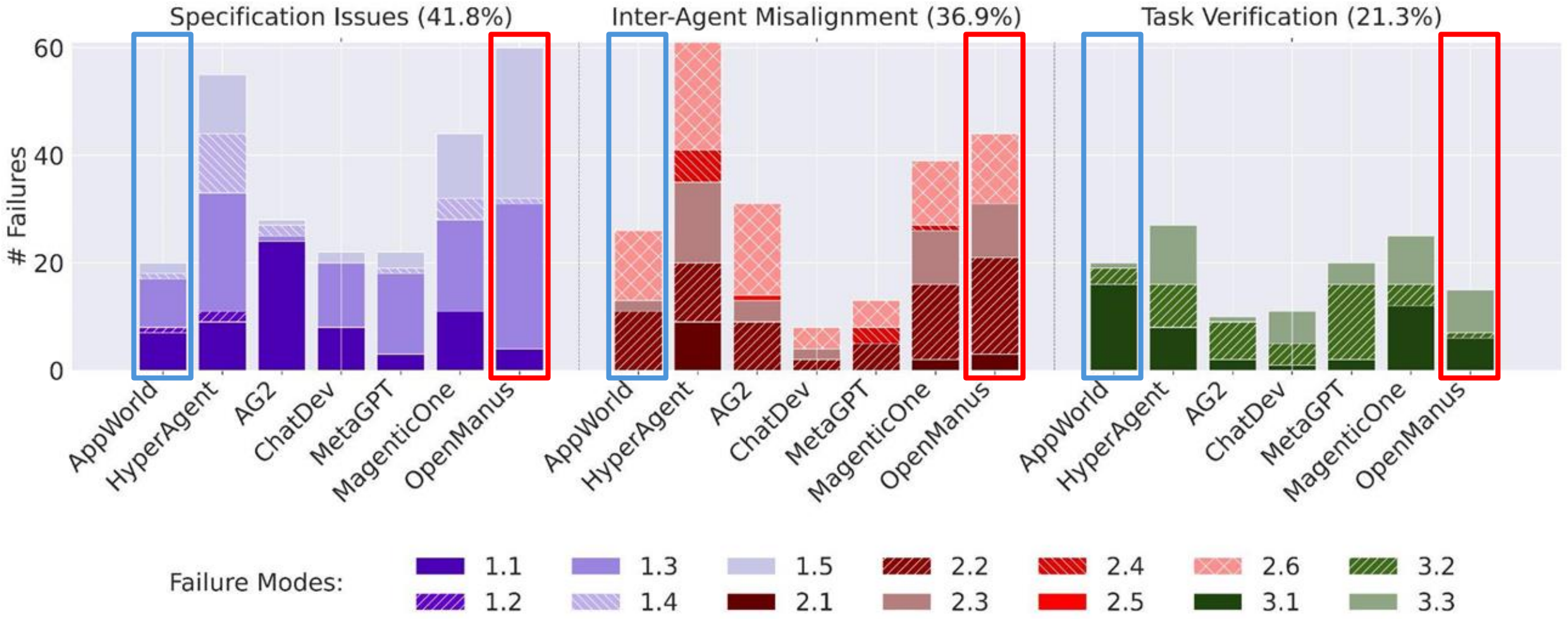


Understanding failures profile



lower is better

Different failure profiles



Going beyond diagnosis with MAST

How do we build custom taxonomies that are special for:

1. Each agentic architecture
2. Each domain (coding, finance, math, writing)

How do we build better agentic systems using these taxonomies

ATLAS

(**A**daptive **T**axonomy **L**earning for **A**gentic **S**ystems)

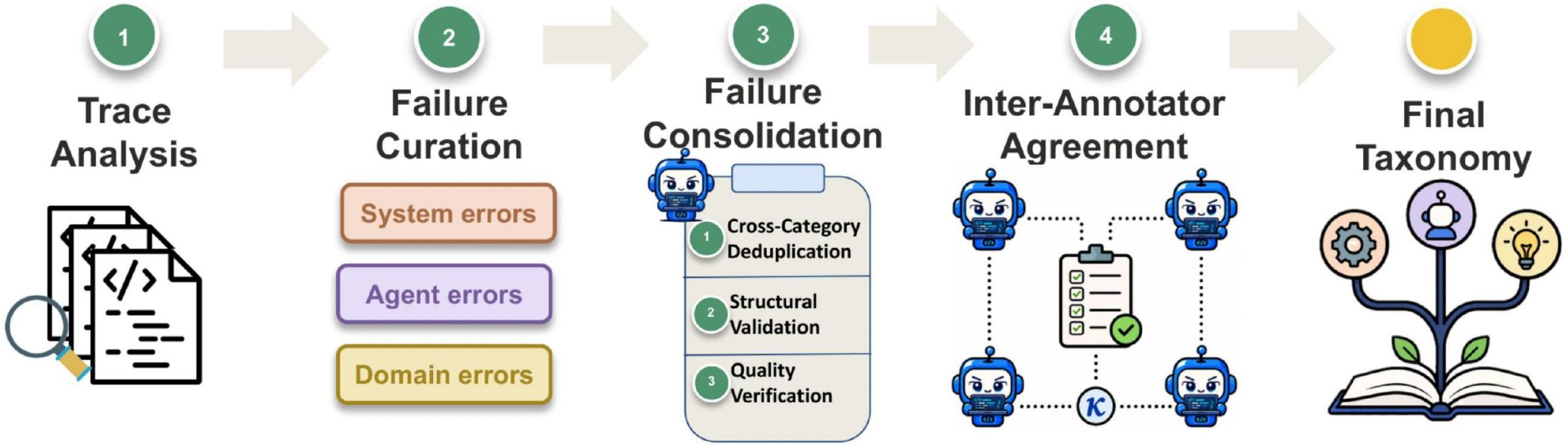
Need for domain-specific taxonomies

Discovered failure modes barely overlap across different domains

Each number represents “failure overlap” across different benchmarks (Jaccard similarity)

ATLAS: Adaptive Taxonomy Learning for Agentic Systems

Fully automate the way we build taxonomies given an agentic system and rollouts



Can you leverage MAST / ATLAS to improve MAS reliability?

How does a MAS trace look like?

System Runtime Log:

- User & System Prompts
- Agent Chat Histories
- Logs from Code Executions

Lots of unstructured messy data

```
| **chat_turn_limit** | 10 |
| **placeholders** | {} |
| **memory** | No existed memory |

[2025-29-03 23:32:42 INFO] Chief Technology Officer: **Chief Technology Officer<->Chief E

[ChatDev is a software company powered by multiple intelligent agents, such as chief exec                    hief
product officer, chief technology officer, etc, with a multi-agent organizational structu                    ld
through programming'.
You are Chief Executive Officer. Now, we are both working at ChatDev and we share a commo
complete a task assigned by a new customer.
Your main responsibilities include being an active decision-maker on users' demands and other key policy issues, leader, manager, and
executor. Your decision-making role involves high-level decisions about policy and strategy; and your communicator role can involve speaking
to the organization's management and employees.
Here is a new customer's task: Design a chess game, allowing two players to take turns and determining the winner. It should be playable
from Linux Terminal, and does not require me to access a dedicated UX or GUI platform, i.e. print the results on terminal at each stage and
let me play it there by entering inputs (formal chess notation such as Ke8). Include standard chess features like castling, en passant, and
pawn promotion. Enforce check and checkmate rules..
To complete the task, I will give you one or more instructions, and you must help me to write a specific solution that appropriately solves
the requested instruction based on your expertise and my needs.]

Python

[2025-29-03 23:32:42 INFO] flask app.py did not start for online log
[2025-29-03 23:32:42 INFO] Chief Executive Officer: **Chief Technology Officer<->Chief Executive Officer on : LanguageChoose, turn 0**

[ChatDev is a software company powered by multiple intelligent agents, such as chief executive officer, chief human resources officer, chief
product officer, chief technology officer, etc, with a multi-agent organizational structure and the mission of 'changing the digital world
through programming'.
You are Chief Technology Officer. we are both working at ChatDev. We share a common interest in collaborating to successfully complete a
task assigned by a new customer.
You are very familiar to information technology. You will make high-level decisions for the overarching technology infrastructure that
closely align with the organization's goals, while you work alongside the organization's information technology ("IT") staff members to
perform everyday operations.
```

Key idea

Instead of using raw logs to debug, use failure modes produced by taxonomy to structure feedback

Leveraging taxonomies to improve MAS

A: As a tool

The taxonomy is in the prompt. The agent self-diagnoses while it runs

B: As a skill

A separate agent grades the trace with the taxonomy and shares reflection

	CLAUDE CODE, HAIKU 4.5	SWE-AGENT, GPT-5
Self reflection	**64.0%**	**60%**
MAST in-prompt	**67.3%**	**68%**
ATLAS Pattern A	**70.7%**	**70%**
ATLAS Pattern B	**69.3%**	**78%**

SWE-bench Verified Mini, 50 instances. Best of three seeds.

Leveraging taxonomies to improve LLM-as-a-Judge

Terminal bench 2.0 with different agent harnesses

	TERMINUS-2	CLAUDE-CODE	FORGECODE
Pass@1	61.8%	57.5%	81.8%
LLM-as-a-Verifier	61.8%	61.2%	86.4%
MAST-Judge	68.5%	69.0%	88.8%
ATLAS-Judge	**73.0%**	**72.4%**	**89.9%**
Best-of-5 oracle	77.5%	80.5%	89.9%

Summary

Reliability, critical to AI success

MAST and ATLAS: Understand why MAS fail

Can use failure categories to improve reliability

Just starting: many challenges ahead!

Parting thoughts

Reliability, important in every system we relay on

Reliability, a stapple of the engineering disciplines

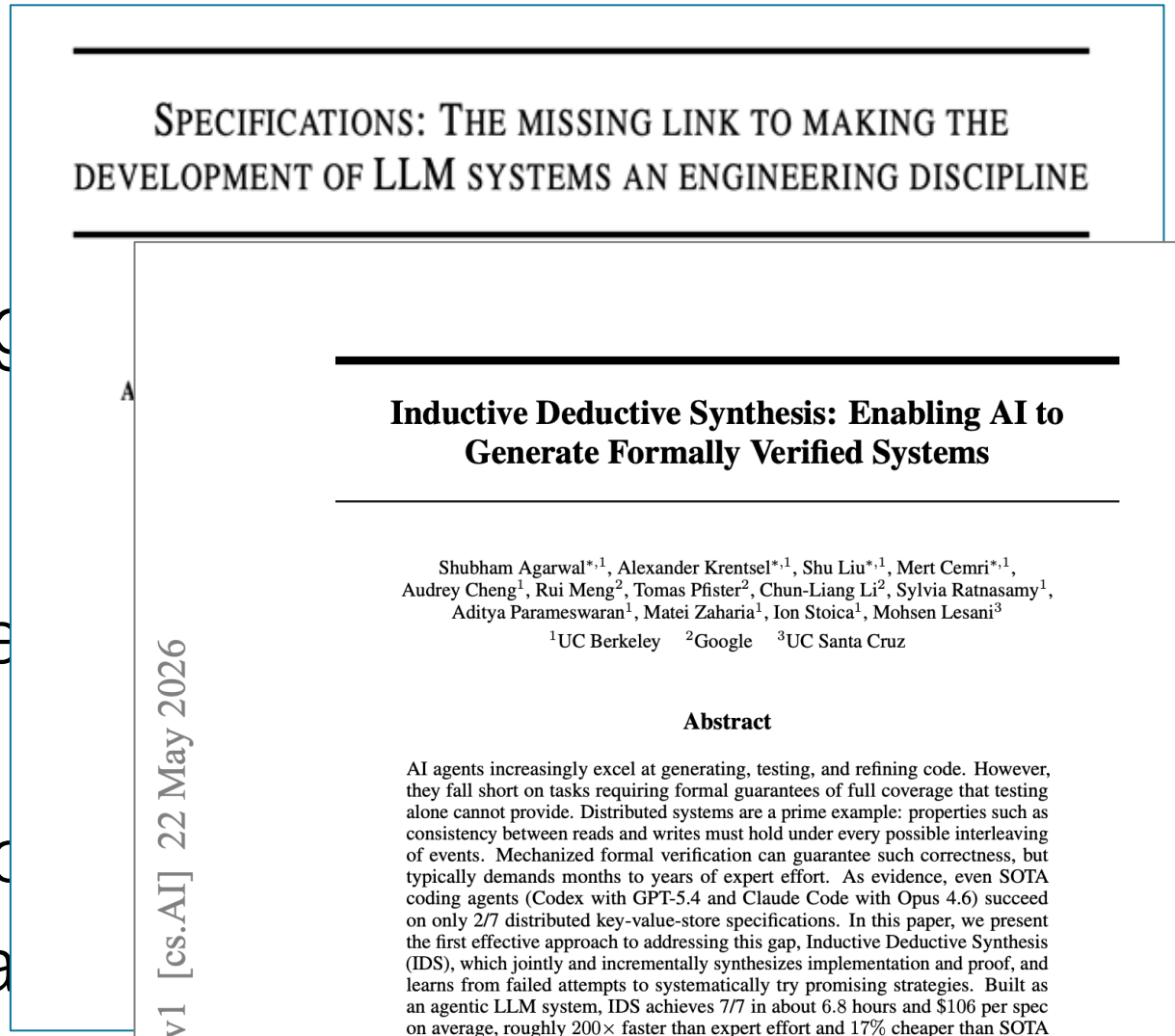
Reliability in AI, arguably more difficult than other systems

Parting thoughts

Specifications, key to buildin

Specifications enable us to:

- **Verify** the system works as inte
- **Debug** the system: find and fix
- **Decompose** system: faster, mo
- **Reuse** existing components: fa
- **Automated decision making**: no hu



SPECIFICATIONS: THE MISSING LINK TO MAKING THE DEVELOPMENT OF LLM SYSTEMS AN ENGINEERING DISCIPLINE

Inductive Deductive Synthesis: Enabling AI to Generate Formally Verified Systems

Shubham Agarwal*,[1], Alexander Krentsel*,[1], Shu Liu*,[1], Mert Cemri*,[1], Audrey Cheng[1], Rui Meng[2], Tomas Pfister[2], Chun-Liang Li[2], Sylvia Ratnasamy[1], Aditya Parameswaran[1], Matei Zaharia[1], Ion Stoica[1], Mohsen Lesani[3]
[1]UC Berkeley [2]Google [3]UC Santa Cruz

Abstract

AI agents increasingly excel at generating, testing, and refining code. However, they fall short on tasks requiring formal guarantees of full coverage that testing alone cannot provide. Distributed systems are a prime example: properties such as consistency between reads and writes must hold under every possible interleaving of events. Mechanized formal verification can guarantee such correctness, but typically demands months to years of expert effort. As evidence, even SOTA coding agents (Codex with GPT-5.4 and Claude Code with Opus 4.6) succeed on only 2/7 distributed key-value-store specifications. In this paper, we present the first effective approach to addressing this gap, Inductive Deductive Synthesis (IDS), which jointly and incrementally synthesizes implementation and proof, and learns from failed attempts to systematically try promising strategies. Built as an agentic LLM system, IDS achieves 7/7 in about 6.8 hours and $106 per spec on average, roughly 200× faster than expert effort and 17% cheaper than SOTA agents. IDS further incorporates performance feedback into the same loop, yielding implementations up to 3× faster than published verified systems.[1]

Learn from other disciplines on how to build reliable AI systems

Parting thoughts

Specifications help make agents verifiable

… but alone don’t ensure correctness

Hardware can enforce correctness (e.g., policy enforcement, data access) and provide provable evidence

Thank you!

The Path Forward

Specify agent behavior

Verify the agent obeys specification, or/and

Enforce specification during agent execution

Two (complementary) approaches:

- Software: tests, formal methods
- Hardware: provable evidence, security enforcement