# 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]

*"Happy families are all alike; each unhappy family is unhappy in its own way."* (Tolstoy [1])
*"Successful systems all work alike; each failing system has its own problems."* (Berkeley'25)

## 1 Introduction

Recently, Large Language Model (LLM) based agentic systems have gained significant attention in the AI community [2–4]. Building on this characteristic, multi-agent systems are increasingly explored in various domains, such as software engineering, drug discoveries, scientific simulations, and general-purpose agents [5–11]. In this study, we define an LLM-based **agent** as an artificial entity with prompt specifications (initial state), conversation trace (state), and ability to interact with the environments such as tool usage (action). A **multi-agent system** (**MAS**) is then defined as a collection of agents designed to interact through orchestration, enabling collective intelligence. MAS are structured to coordinate efforts, enabling task decomposition, performance parallelization, context isolation, specialized model ensembling, and diverse reasoning discussions [12–17].

Despite the increasing adoption of MAS, their performance gains often remain minimal compared to single-agent frameworks [18] or simple baselines like best-of-N sampling [19]. Our empirical

[1] `https://github.com/multi-agent-systems-failure-taxonomy/MAST`
[2] `https://huggingface.co/datasets/mcemri/MAST-Data`

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

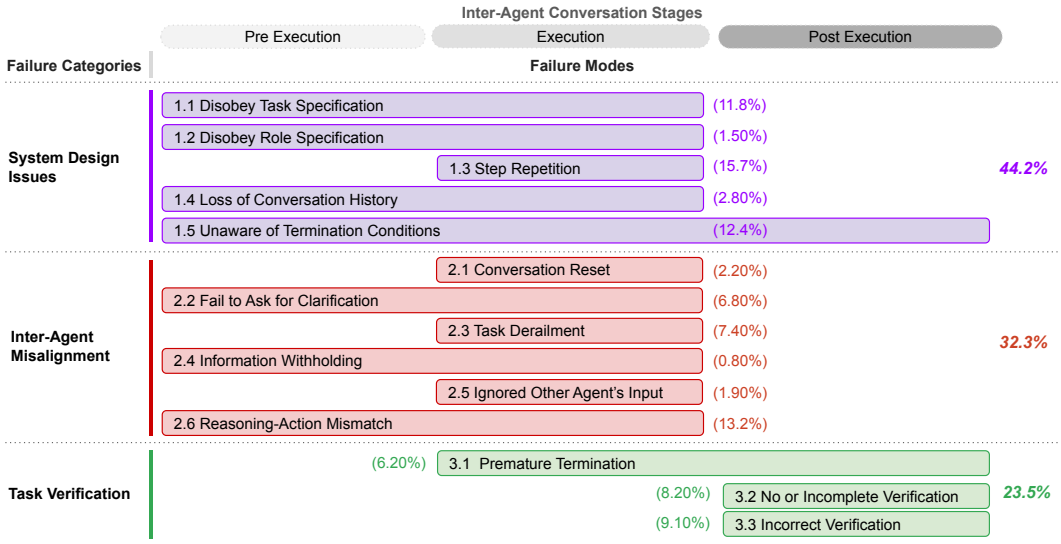


Figure 1: MAST: A **Taxonomy of MAS Failure Modes**. The inter-agent conversation stages indicate when a failure typically occurs within the end-to-end MAS execution pipeline. A failure mode spanning multiple stages signifies that the underlying issue can manifest or have implications across these different phases of operation. The percentages shown represent the prevalence of each failure mode and category as observed in our analysis of 1642 MAS execution traces. Detailed definitions for each failure mode and illustrative examples are available in Appendix A.

analysis reveals 41% to 86.7% failure rate on 7 state-of-the-art (SOTA) open-source MAS detailed in Figure 5 (Appendix B). Furthermore, there is no clear consensus on how to build robust and reliable MAS. This motivates the fundamental question we address: *Why do MAS fail?*

To address this question and systematically understand MAS failures, we introduce MAST-Data, a comprehensive, high-quality collection of 1642 annotated execution traces. We define failures as instances where the MAS does not achieve its intended task objectives. As shown in Table 1, we collect traces from 7 popular MAS frameworks run with two main model families (GPT-4 series and Claude series), covering tasks such as coding, math problem-solving, and general agent functionalities. Alongside MAST-Data, we also release MAST-Data-human, a smaller dataset featuring 21 traces annotated by three human experts each during our inter-annotator agreement studies. We create MAST-Data to outline failure dynamics in MAS and to guide the development of better future systems.

Systematically annotating failures in diverse MAS for a large-scale dataset like MAST-Data presents challenges unique to MAS: the difficulty in verifying ground truth for root cause detection and the absence of standardized failure definitions. To mitigate these challenges in creating MAST-Data, we first develop the **M**ulti-**A**gent **S**ystem Failure **T**axonomy (MAST), illustrated in Figure 1. We build MAST using Grounded Theory [20] from a close analysis of over 150 MAS execution traces (each averaging over 15,000 lines of text). This analysis spans a subset of five open-source MAS frameworks and involves six expert human annotators. To ensure generalizable definitions in MAST for labeling MAST-Data, three annotators independently and iteratively labeled a total of 15 traces until achieving high inter-annotator agreement ($\kappa = 0.88$). This comprehensive analysis results in 14 distinct failure modes, clustered into 3 categories. While MAST serves as a foundational first step towards unifying the understanding of MAS failures, we do not claim it covers every potential failure pattern. To enable scalable annotation for the full MAST-Data, we then develop an LLM-as-a-judge pipeline (which we term the LLM annotator) [21] using OpenAI's o1 model. We calibrate the LLM annotator to achieve high agreement with human expert annotations ($\kappa = 0.77$), and additionally validated applicability on two additional unseen MAS and benchmarks ($\kappa = 0.79$).

To demonstrate MAST's practical usage, our case studies (Appendix H) highlight its role in guiding MAS development, and in Section C we describe how to use the MAST easily as a python library using `pip install agentdash`. For example, the CPO agent in ChatDev can exhibit 'Failure Mode 1.2 - Disobey Role Specification' by terminating conversation without the CEO agent's consensus. We

demonstrate that a straightforward system workflow adjustment ensuring the CEO had the final say contributed to a +9.4% increase in overall task success rate. While such MAST-guided interventions demonstrate improvements, achieving robust MAS reliability often requires more than isolated fixes, pointing towards the need for more complex solutions and fundamental MAS redesigns.

Table 1: MAST-Data configuration details. HE: Human Evaluated (Task completions rates are checked by humans), HA: Human Annotated (Failure modes are annotated by humans), LA: LLM Annotated (Failure modes are annotated by LLM-as-a-Judge).

| MAS | Benchmark | LLM | Annotation | Trace # |
|---|---|---|---|---|
| ChatDev | ProgramDev | GPT-4o | HE, HA, LA | 30 |
| MetaGPT | ProgramDev | GPT-4o | HE, HA, LA | 30 |
| HyperAgent | SWE-Bench Lite | Claude-3.7-Sonnet | HE, HA, LA | 30 |
| AppWorld | Test-C | GPT-4o | HE, HA, LA | 30 |
| AG2 (MathChat) | GSM-Plus | GPT-4 | HE, HA, LA | 30 |
| Magentic-One | GAIA | GPT-4o | HE, HA, LA | 30 |
| OpenManus | ProgramDev | GPT-4o | HE, HA, LA | 30 |
| ChatDev | ProgramDev-v2 | GPT-4o | LA | 100 |
| MetaGPT | ProgramDev-v2 | GPT-4o | LA | 100 |
| MetaGPT | ProgramDev-v2 | Claude-3.7-Sonnet | LA | 100 |
| ChatDev | ProgramDev-v2 | Qwen2.5-Coder-32B-Instruct | LA | 100 |
| MetaGPT | ProgramDev-v2 | Qwen2.5-Coder-32B-Instruct | LA | 100 |
| ChatDev | ProgramDev-v2 | CodeLlama-7b-Instruct-hf | LA | 100 |
| MetaGPT | ProgramDev-v2 | CodeLlama-7b-Instruct-hf | LA | 100 |
| AG2 (MathChat) | OlympiadBench | GPT-4o | HE, LA | 206 |
| AG2 (MathChat) | GSMPlus | Claude-3.7-Sonnet | HE, LA | 193 |
| AG2 (MathChat) | MMLU | GPT-4o-mini | HE, LA | 168 |
| Magentic-One | GAIA | GPT-4o | HE, LA | 165 |

These findings suggest MAST reflects fundamental design challenges inherent in current MAS, not just artifacts of specific MAS implementation. By systematically defining failures, MAST serves as a framework to guide failure diagnosis and opens concrete research problems for the community. We have released our traces and annotations and open-sourced the LLM annotator pipeline to foster research in the design of more robust and reliable MAS.

The contributions of this paper are as follows:

- We introduce and **open-source** MAST-Data, the first large-scale MAS failure dataset with consistent annotations from 7 MAS and four model families. And MAST-Data-human, a detailed inter-annotator study results with human labels. Together serve to facilitate research into MAS failures.
- We introduce MAST, the first empirically grounded **taxonomy of MAS failures**, providing a structured framework for defining, understanding and annotating failures.
- We develop a scalable LLM-as-a-judge **annotation pipeline** integrated with MAST for efficiently annotating MAST-Data and enabling analysis of MAS performance, diagnosis of failure modes, and understanding of failure breakdowns.
- We demonstrate through **case studies** that failures identified by MAST often stem from system design issues, not just LLM limitations or simple prompt following, and require more than superficial fixes, thereby highlighting the need for structural MAS redesigns.

## 2 Related Work

### 2.1 Challenges in Agentic Systems

The promising capabilities of agentic systems have inspired research into solving specific challenges. For instance, Agent Workflow Memory [22] addresses long-horizon web navigation by introducing workflow memory. DSPy [23] tackles issues in programming agentic flows, while StateFlow [24] focuses on state control within agentic workflows to improve task-solving capabilities. Several surveys also highlight challenges and potential risks specifically within MAS [25, 26]. While these works meaningfully contribute towards understanding specific issues or providing high-level overviews, they do not offer a fine-grained, empirically grounded taxonomy of *why* MAS fail across diverse systems and tasks. Numerous benchmarks also exist to evaluate agentic systems [27–32]. These evaluations are crucial but primarily facilitate a top-down perspective, focusing on aggregate performance or high-level objectives like trustworthiness and security [33, 34]. Our work complements these efforts by providing a bottom-up analysis focused on identifying specific failure modes in MAS.

### 2.2 Design Principles for Agentic Systems

Several works highlight challenges in building robust agentic systems and suggest design principles, often focused on single-agent settings. For instance, Anthropic's blog post emphasizes modular components and avoiding overly complex frameworks [35]. Similarly, Kapoor et al. [19] demonstrates how complexity can hinder practical adoption. Our work extends these insights to the multi-agent context. By systematically collecting and analyzing a large corpus of MAS failure instances within `MAST-Data`, and by developing `MAST`, we provide not only a structured understanding of *why* MAS fail but also empirical data from `MAST-Data` to support the development and validation of more robust design principles for MAS. This aligns with the call for clearer specifications and design principles [36].

### 2.3 Related Datasets and Taxonomy

Despite the growing interest in LLM agents, dedicated research systematically characterizing their failure modes remains limited, particularly for MAS. While Bansal et al. [37] catalogs challenges in human-agent interaction, our contribution focuses specifically on failures within autonomous MAS execution. Other related work includes taxonomies for evaluating multi-turn LLM conversations [38] or specific capabilities like code generation [39]. These differ significantly from our goal of developing a generalizable failure taxonomy for multi-agent interactions and coordination.

Further related efforts aim to improve MAS through different approaches. AgentEval [40] proposes a framework using LLM agents to define and quantify multi-dimensional evaluation criteria reflecting task utility for end-users. AGDebugger [41] introduces an interactive tool enabling developers to debug and steer agent teams by inspecting and editing message histories. And currentwork by Zhang et al. [42] present the Who&When dataset and MAS debugger, which focuses on summarizing failures for specific task items by attributing them to particular agents and error steps.

Thus, `MAST-Data` and `MAST` represent, to our knowledge, the first empirically derived, comprehensive dataset and taxonomy focused specifically on MAS failures focus on failure patterns. Identifying these patterns highlights the need for continued research into robust evaluation metrics and mitigation strategies tailored for the unique challenges of MAS.

## 3 The Multi-Agent Systems Dataset

To facilitate a principled understanding of why MAS fail and to guide the development of more reliable future systems, we introduce `MAST-Data`, the Multi-Agent System Failure Dataset. `MAST-Data` is a comprehensive, empirically grounded dataset comprising 1642 annotated execution traces collected from 7 popular MAS frameworks, covering domains of coding, math, and generic tasks.

Constructing such a dataset, however, presents distinct challenges. First, unlike in traditional software where failures often have clearly identifiable root causes, failures in MAS are frequently complex. They involve convoluted agent interactions and the compounding effects of individual model behaviors and overall system design. Therefore, pinpointing the precise nature and origin of a failure in MAS

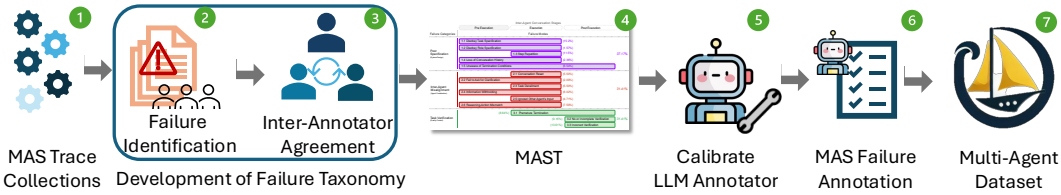


Figure 2: Methodological workflow for constructing the MAST-Data dataset, involving the empirical identification of failure modes, the development of MAST, iterative refinement through inter-annotator agreement studies ($\kappa = 0.88$), and the creation of a scalable LLM annotation pipeline. This figure highlights our systematic approach to creating a comprehensive dataset for studying MAS failures.

requires more than simple error detection; it necessitates understanding the system's dynamics. Second, the lack of a standardized failure framework with clear definitions makes identifying and classifying MAS failures across different systems inconsistent, which complicates annotation and cross-system analysis.

To address these challenges, we develop a rigorous, principled methodology to construct MAST-Data. In this section, we detail our approach, which centers on building the first empirical MAS failure taxonomy, MAST, and a scalable annotation pipeline for systematic and comprehensive data collection. Figure 2 summarizes our methodological workflow.

### 3.1 Data Collection with Grounded Theory Analysis

To uncover a comprehensive set of failure patterns that are both diverse and generalizable to standardize failure labels which we detail further in Section 3.2, we first collect 150 traces from five MAS frameworks, which are closely examined by six human experts. Our goal at this stage is to identify as many distinct failure modes as possible, ensuring these observed patterns are not merely artifacts of a single system but can likely apply more broadly. To achieve this without predefined hypotheses, we adopt the **Grounded Theory** (GT) approach [20]. This qualitative research method allows failure modes to emerge organically from empirical data.

For this initial data collection, we use *theoretical sampling* [43] to ensure robust coverage across different system objectives and interaction patterns. This method guides our selection of the five MAS frameworks (HyperAgent, AppWorld, AG2, ChatDev, and MetaGPT) and two task categories (programming and math problem-solving). We then iteratively analyze these traces using core GT techniques: *open coding* [44] to label trace data with observed failure behaviors; *constant comparative analysis* to refine our understanding of these failure behaviors and their recurrence across systems; *memoing* to document insights; and *theorizing* to structure these findings into an initial set of failure modes with their definitions. This iterative analysis continues until we reach *theoretical saturation*, where further data analysis does not yield new failure mode insights. This initial process requires significant human effort, over 20 hours of annotation per expert for these 150 traces.

### 3.2 Standardizing Failure Labels via Inter-Annotator Agreement

To make the failure observations from our GT analysis useful for creating consistent labels in MAST-Data, we recognize the critical need for standardized definitions that apply uniformly across different MAS. To address this, we develop the failure taxonomy - MAST. MAST serves as a foundational first step towards a common understanding of MAS failures by providing clear, empirically grounded failure observation labels. We provide a detailed description and analysis of MAST in Section 4.

To develop a taxonomy that is unambiguous and consistently applicable by different annotators, we rigorously validate and refine MAST definitions through Inter-Annotator Agreement (IAA) studies. This iterative process begins with a preliminary version of MAST derived from our GT findings. In each round of IAA, three expert annotators independently label a subset of five randomly selected traces from our initial 150+ trace collection using MAST. We then facilitate discussions to collectively resolve any disagreements. Based on these discussions, we iteratively refine MAST by adjusting failure mode definitions, adding new modes, or removing and merging existing ones until we achieve

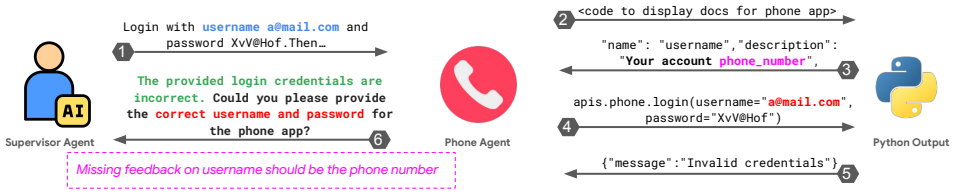


Figure 3: Visualization of a trace segment in MAST-Data. This illustrates an agent-to-agent conversation exhibiting Failure Mode 2.4: Information Withholding. The Phone Agent fails to communicate API requirements (username format) to the Supervisor Agent, who also fails to seek clarification, leading to repeated failed logins and task failure.

high consensus. We conduct three such rounds of IAA, requiring about 10 hours in total solely for resolving disagreements, not including the annotation time itself. We measure agreement using Cohen's Kappa score, achieving a strong average of $\kappa = 0.88$ in the final rounds. This high IAA score signifies that MAST provides a clear and shared understanding of failure modes, crucial for the consistent annotation of MAST-Data. Figure 3 illustrates an example of a trace snippet with a MAST label.

### 3.3 Enabling Scalable Annotation: The LLM-as-a-Judge Pipeline

Manually annotating over 1600 MAS traces with fine-grained failure modes is time-consuming and costly. To enable scalable and automated failure annotation for MAST-Data, we develop an LLM-as-a-Judge pipeline (LLM annotator), building upon our validated MAST. This pipeline prompts an LLM (OpenAI's o1 model) with an execution trace, the MAST definitions, and few-shot examples from our human-annotated data (details in Appendix N) to classify observed failure modes. We validate the LLM annotator's reliability against expert human annotations on a held-out set from our IAA studies. The LLM annotator achieves high agreement with human experts (accuracy 94%, Cohen's Kappa of 0.77; Table 2), confirming its suitability for scaling the annotation process while adhering to MAST definitions.

Table 2: Performance of LLM-as-a-judge pipeline

| Model | Accuracy | Recall | Precision | F1 | Cohen's $\kappa$ |
|---|---|---|---|---|---|
| o1 | 0.89 | 0.62 | 0.68 | 0.64 | 0.58 |
| o1 (few shot) | 0.94 | 0.77 | 0.833 | 0.80 | 0.77 |

### 3.4 Constructing the Multi-Agent Dataset

Before large-scale data collection for MAST-Data, we confirm the generalizability of our finalized MAST and the LLM annotator. We evaluate their performance on two new MAS (OpenManus and Magentic-One) with two new benchmarks (MMLU and GAIA, the latter representing a new general-agent task domain for validation) not part of the initial MAST development. An additional human IAA round on these out-of-domain traces using the finalized MAST yields a strong Cohen's Kappa score of 0.79. This demonstrates MAST's effectiveness in capturing failures in diverse systems and tasks without further modification, supporting the robustness of our annotation approach for broader application. We further detail the uniqueness of MAST failure modes via a correlation study in Appendix E.

Leveraging our validated MAST and LLM annotator, we expand data collection to construct MAST-Data, comprising 1642 annotated traces from seven popular MAS frameworks (Table 1). These frameworks include the five from our initial studies, the two from the generalization validation, and Manus [45] as detailed in Appendix B. These traces cover diverse tasks like coding, math problem-solving, and general agent functionalities. For MAST-Data, our LLM annotator identifies MAST failure modes in each trace and provides a corresponding reason. We also release MAST-Data-human, consisting of all traces annotated by human experts during our IAA studies, where each annotation specifies MAST failure modes with textual justifications. We open-source

MAST-Data and MAST-Data-human as resources to analyze MAS failure dynamics and guide robust system design.

## 4 The Multi-Agent System Failure Taxonomy

This section details MAST, a key result of our study and a critical component that guides the creation and analysis of MAST-Data. MAST provides the first empirically grounded, structured framework for defining, understanding, and annotating common failures in MAS. Here, we present its structure, the failure categories it defines, and key insights derived from its development and application.

MAST, illustrated in Figure 1, identifies 14 fine-grained failure modes, which we map to MAS execution stages (Pre-Execution, Execution, and Post-Execution) where their root causes commonly emerge. These modes are organized into 3 overarching categories reflecting the fundamental nature of the observed failures. While we recognize that prior works have noted some individual failure types and we do not claim MAST is exhaustive, it offers precise definitions for a structured approach to understanding why MAS fail. Detailed definitions for each failure mode (FM) are available in Appendix A, with specific examples in Appendix N.

We acknowledge that some MAS failures can stem from fundamental limitations of current LLMs, such as hallucination or instruction following. However, in developing MAST, we focus on identifying failure patterns where improvements in system design, agent coordination, and verification can offer room to improve the reliability of MAS, often independently of or complementary to advancements in the base models themselves. We now discuss each failure category (FC) in MAST and its implications.

**FC1. System Design Issues.** Failures originate from system design decisions, and poor or ambiguous prompt specifications.
**Insight 1.** MAS failure is not merely a function of challenges in the underlying model; a well-designed MAS can result in performance gain when using the same underlying model.

Failures in FC1 occur during execution but often reflect flaws in pre-execution design choices regarding system architecture, prompt instructions, or state management. These include failing to follow task requirements (FM-1.1, 11.8%) or agent roles (FM-1.2, 1.5%), step repetitions (FM-1.3, 15.7%), context loss (FM-1.4, 2.80%), or not recognizing task completion (FM-1.5, 12.4%). While FM-1.1 and FM-1.2, disobey specifications, may seem like general instruction-following limitation, we identify deeper causes: (1) flaws in MAS design regarding agent roles and workflow, (2) poor user prompt specifications, or (3) limitations of the underlying LLM. We posit that a well-designed MAS should interpret high-level objectives with minimal but clear user input to mitigate the impact of points (2) and (3).

For instance, when ChatDev is tasked to create a Wordle game with the prompt *a standard wordle game by providing a daily 5-letter...*, the generated program uses a fixed word dictionary. Even with a more explicit prompt like *...without having a fixed word bank, and randomly select a new 5-letter word each day*, ChatDev still produces code with a fixed list and new errors. This suggests failures stem from the MAS's design for interpreting specifications. Our intervention studies (Appendix H) show that improving agent role specifications alone yields a +9.4% success rate increase for ChatDev with the same user prompt and LLM (GPT-4o).

**FC2. Inter-Agent Misalignment.** Failures arise from a breakdown in critical information flow from inter-agent interaction and coordination during execution.
**Insight 2.** Solutions focused on context or communication protocols are often insufficient for FC2 failures, which demand deeper 'social reasoning' abilities from agents.

FC2 covers failures in agent coordination. These include unexpected conversation resets (FM-2.1, 2.20%), proceeding with wrong assumptions instead of seeking clarification (FM-2.2, 6.80%), task derailment (FM-2.3, 7.40%), withholding crucial information (FM-2.4, 0.85%), ignoring other agents' input (FM-2.5, 1.90%), or mismatches between reasoning and action (FM-2.6, 13.2%). Figure 3 illustrates information withholding (FM-2.4). Diagnosing FC2 failures can be complex, as similar surface behaviors (e.g., missing information) can stem from different root causes like withholding (FM-2.4), ignoring input (FM-2.5), or context mismanagement (FM-1.4), underscoring the need for MAST's fine-grained modes.

Recent system innovations, such as Model Context Protocol [46] and Agent to Agent [47], improve agent communication by standardizing message formats from different tool or agent providers. However, the errors we observe in FC2 occur even when agents within the same framework communicate using natural language. This signals a deeper agent interaction dynamic challenge: the collapse of 'theory of mind' [48], where agents fail to accurately model other agents' informational needs. Addressing this likely requires structural improvements to the content of agent messages or enhancing models' contextual reasoning and their capacity to infer other agents' informational needs, such as through targeted training, as base LLMs are generally not pre-trained for such nuanced inter-agent dynamics. Thus, robust solutions will likely involve a combination of improved MAS architecture and model-level advancements in communicative intelligence.

**FC3. Task Verification.** Failures involve inadequate verification processes that fail to detect or correct errors, or premature termination of tasks.
**Insight 3.** Multi-Level Verification is Needed. Current verifier implementations are often insufficient; sole reliance on final-stage, low-level checks is inadequate.

FC3 failures are related to the quality control of the final output, including premature termination (FM-3.1, 6.20%), no or incomplete verification (FM-3.2, 8.20%), or incorrect verification (FM-3.3, 9.10%). These highlight challenges in ensuring output correctness and reliability. Systems with explicit verifiers like MetaGPT and ChatDev generally show fewer total failures (Figure 4), indicating explicit checks help. However, the presence of a verifier is not a silver bullet, as overall MAS success rates can still be low. For example (FM-3.2), a ChatDev-generated chess program passes superficial checks (e.g., code compilation) but contains runtime bugs because it fails to validate against actual game rules, rendering the output unusable despite review phases.

During our GT analysis of MAS traces, we find that many existing verifiers perform only superficial checks, despite being prompted to perform thorough verification, such as checking if the code compiles or if there are leftover `TODO` comments. We posit that MAS development should take lessons from traditional software development where programmers test their code before committing. More rigorous verification is needed, such as using external knowledge, collecting testing output throughout generation, and multi-level checks for both low-level correctness and high-level objectives. We demonstrate this in an intervention study where adding a high-level task objective verification step to ChatDev yields a +15.6% improvement in task success on ProgramDev (details in Appendix H).

## 5 Towards better Multi-Agent LLM Systems

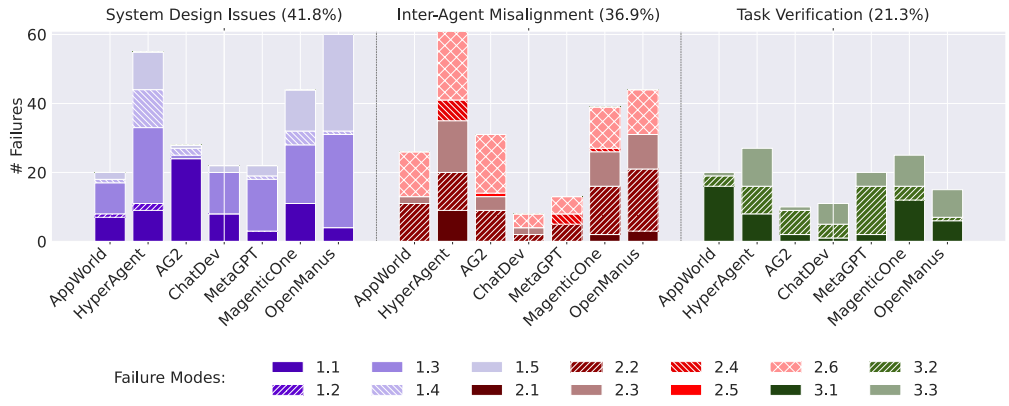


Figure 4: Distribution of failure in `MAST-Data` with `MAST` labels on total 210 traces. This plot visualizes the failure distributions of the first 30 traces for each system. As the specific tasks and benchmarks may differ across the MAS configurations shown, these results are intended to illustrate system-specific failure profiles rather than to serve as a performance comparison across MAS.

We now discuss the broader implications and usage of `MAST-Data` and `MAST`. `MAST-Data`, with its annotations grounded in `MAST`, provides crucial empirical evidence, while `MAST` offers a foundational framework and practical tool for understanding, debugging, and ultimately improving MAS. By

concretely defining failure modes and providing a large-scale dataset of their occurrences, our work outlines the challenges in building reliable MAS and opens targeted research problems for the community.

### 5.1 Failure Breakdown in MAST-Data

Our analysis of MAST-Data reveals that failure distributions differ markedly across various MAS, often reflecting their unique architectural characteristics and design philosophies. For example, as illustrated in Figure 4, we observe specific patterns: AppWorld frequently suffers from premature terminations (FM-3.1), potentially due to its star topology and lack of a predefined workflow making termination conditions less obvious; OpenManus exhibits a tendency towards step repetition (FM-1.3); and HyperAgent could benefit from addressing its dominant failure modes of step repetition (FM-1.3) and incorrect verification (FM-3.3). These system-specific profiles underscore that there is no one-size-fits-all solution to MAS failures.

We also use MAST-Data to study the impact of different underlying language models and MAS designs on failure patterns. For instance, when comparing GPT-4o and Claude 3.7 Sonnet within the MetaGPT framework on programming tasks, we find that while GPT-4o generally performs better than Claude, it shows significantly fewer FC1 (System Design Issues) failures by 39%. We also examine the impact of different MAS designs on the same benchmark, such as comparing MetaGPT and ChatDev on ProgramDev. Here, while MetaGPT generally outperforms ChatDev by having 60-68% less failure in FC1 and FC2, it has 1.56x more FC3 failure than ChatDev. These comparative analyses, detailed further in Appendix F, provide insights into how model choice and architectural patterns influence system performance and distribution of failures.

### 5.2 MAST as a Practical Development Tool

Developing robust MAS is challenging: aggregate success rates can obscure the specific impacts of optimizations. MAST addresses this by providing a structured vocabulary for systematic failure breakdown. Using our LLM annotator with MAST, developers can obtain quantitative analyses of failure profiles for specific systems. We demonstrate MAST's practical usage in guiding MAS improvement in our case studies (Appendix H). The Failure Mode breakdown analysis (Appendix H.3) shows which failure modes were mitigated and reveals any resulting trade-offs. This granular view, moving beyond aggregate metrics, is crucial for understanding *why* an intervention works and for iterating effectively towards more robust systems.

### 5.3 Beyond Model Capabilities: The Primacy of System Design

While one could simply attribute failures in MAST-Data to limitations of present-day LLM (e.g., hallucinations, misalignment), we conjecture that improvements in the base model capabilities will be insufficient to address the full MAST. Instead, we argue that good MAS design requires organizational understanding – even organizations of sophisticated individuals can fail catastrophically [49] if the organization structure is flawed. Previous research in high-reliability organizations has shown that well-defined design principles can prevent such failures [50, 51].

Consistent with organization theories, our findings indicate that many MAS failures arise from the challenges in organizational design and agent coordination rather than the limitations of individual agents. In our intervention case studies (Appendix H), we apply MAS system workflow and prompt changes respectively (results in Table 5). With the same underlying model, we achieve max improvements of 15.6%. This highlights that MAS failures can be address with better system designs.

Although first step interventions lead to performance gains, not all failure modes are resolved, and task completion rates still remain low, indicating that more substantial improvements are needed. Achieving high reliability may requires combinatorial changes ranging from agent system organization to model level improvements (see Table 4). MAST, by providing a clear framework of failure points identified from MAST-Data, helps identify where these structural weaknesses lie and can guide the design and evaluation of more sophisticated MAS architectures.

## 6 Conclusion

In this study, we conduct the first systematic investigation into why MAS fail. This investigation results in the MAST-Data: a comprehensive public resource of over 1600 annotated execution traces from 7 popular MAS frameworks, which we create to outline MAS failure dynamics and guide future system development. To enable MAST-Data's systematic annotation and analysis, we first develop the **M**ulti-**A**gent **S**ystem Failure **T**axonomy (MAST). We build MAST through a rigorous Grounded Theory-based analysis of an initial 150 traces, validating its definitions with strong inter-annotator agreement and identifying 14 distinct failure modes across 3 categories. For scalable annotation of MAST-Data using MAST, we then develop an LLM annotator, confirming its high agreement with human experts. Together, MAST-Data and MAST provide a foundational framework and empirical grounding for future MAS research.

We are excited about the potential of MAS, but their widespread adoption hinges on achieving greater reliability. Our work, through the public release of MAST-Data, MAST, and the LLM annotator, contributes towards this goal. MAST-Data offers a rich empirical basis for understanding current failure dynamics, while MAST provides a standardized language and framework to diagnose and mitigate these failures. By systematically identifying and categorizing challenges, we aim to open up concrete research directions and equip the community to develop more robust and effective multi-agent systems.

## References


[1] Leo Tolstoy. *Anna Karenina*. The Russian Messenger, 1878.

[2] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL https://arxiv.org/abs/2305.15334.

[3] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

[4] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

[5] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023. URL https://arxiv.org/abs/2307.07924.

[6] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2024. URL https://arxiv.org/abs/2407.16741.

[7] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, Yossi Matias, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yunhan Xu, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Towards an ai co-scientist, 2025. URL https://arxiv.org/abs/2502.18864.

[8] Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, 2024. doi: 10.1101/2024.11.11.623004. URL https://www.biorxiv.org/content/early/2024/11/12/2024.11.11.623004.

[9] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

[10] Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, and Sirui Hong. Openmanus: An open-source framework for building general ai agents. https://github.com/mannaandpoem/OpenManus, 2025.

[11] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

[12] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Vision and the road ahead, 2024. URL https://arxiv.org/abs/2404.04834.

[13] Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models, 2023. URL https://arxiv.org/abs/2307.04738.

[14] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models, 2024. URL https://arxiv.org/abs/2307.02485.

[15] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

[16] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[17] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

[18] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024. URL https://arxiv.org/abs/2407.01489.

[19] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter, 2024. URL https://arxiv.org/abs/2407.01502.

[20] Barney G. Glaser and Anselm L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, 1967.

[21] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

[22] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

[23] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023. URL https://arxiv.org/abs/2310.03714.

[24] Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. Stateflow: Enhancing llm task-solving through state-driven workflows, 2024. URL https://arxiv.org/abs/2403.11322.

[25] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. Llm multi-agent systems: Challenges and open problems, 2024. URL https://arxiv.org/abs/2402.03578.

[26] Lewis Hammond, Alan Chan, Jesse Clifton, Jason Hoelscher-Obermaier, Akbir Khan, Euan McLean, Chandler Smith, Wolfram Barfuss, Jakob Foerster, Tomáš Gavenčiak, The Anh Han, Edward Hughes, Vojtěch Kovařík, Jan Kulveit, Joel Z. Leibo, Caspar Oesterheld, Christian Schroeder de Witt, Nisarg Shah, Michael Wellman, Paolo Bova, Theodor Cimpeanu, Carson Ezell, Quentin Feuillade-Montixi, Matija Franklin, Esben Kran, Igor Krawczuk, Max Lamparth, Niklas Lauffer, Alexander Meinke, Sumeet Motwani, Anka Reuel, Vincent Conitzer, Michael Dennis, Iason Gabriel, Adam Gleave, Gillian Hadfield, Nika Haghtalab, Atoosa Kasirzadeh, Sébastien Krier, Kate Larson, Joel Lehman, David C. Parkes, Georgios Piliouras, and Iyad Rahwan. Multi-agent risks from advanced ai, 2025. URL https://arxiv.org/abs/2502.14143.

[27] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[28] Ji-Lun Peng, Sijia Cheng, Egil Diau, Yung-Yu Shih, Po-Heng Chen, Yen-Ting Lin, and Yun-Nung Chen. A survey of useful llm evaluation. *arXiv preprint arXiv:2406.00936*, 2024.

[29] Wei Wang, Dan Zhang, Tao Feng, Boyan Wang, and Jie Tang. Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems. *arXiv preprint arXiv:2408.15971*, 2024.

[30] Timothée Anne, Noah Syrkis, Meriem Elhosni, Florian Turati, Franck Legendre, Alain Jaquier, and Sebastian Risi. Harnessing language for coordination: A framework and benchmark for llm-driven multi-agent control. *arXiv preprint arXiv:2412.11761*, 2024.

[31] Matteo Bettini, Amanda Prorok, and Vincent Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024.

[32] Qian Long, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. Teamcraft: A benchmark for multi-modal multi-agent systems in minecraft. *arXiv preprint arXiv:2412.05255*, 2024.

[33] Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. Trustworthy llms: A survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374*, 2023.

[34] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[35] Anthropic, Dec 2024. URL https://www.anthropic.com/research/building-effective-agents.

[36] Ion Stoica, Matei Zaharia, Joseph Gonzalez, Ken Goldberg, Hao Zhang, Anastasios Angelopoulos, Shishir G Patil, Lingjiao Chen, Wei-Lin Chiang, and Jared Q Davis. Specifications: The missing link to making the development of llm systems an engineering discipline. *arXiv preprint arXiv:2412.05299*, 2024.

[37] Gagan Bansal, Jennifer Wortman Vaughan, Saleema Amershi, Eric Horvitz, Adam Fourney, Hussein Mozannar, Victor Dibia, and Daniel S. Weld. Challenges in human-agent communication. Technical Report MSR-TR-2024-53, Microsoft, December 2024. URL https://www.microsoft.com/en-us/research/publication/human-agent-interaction-challenges/.

[38] Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 7421–7454. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.401. URL http://dx.doi.org/10.18653/v1/2024.acl-long.401.

[39] Song Da, Zijie Zhou, Zhijie Wang, Yuheng Huang, Shengmai Chen, Bonan Kou, Lei Ma, and Tianyi Zhang. An empirical study of code generation errors made by large language models. In *In 7th Annual Symposium on Machine Programming*, 2023.

[40] Negar Arabzadeh, Siqing Huo, Nikhil Mehta, Qinqyun Wu, Chi Wang, Ahmed Awadallah, Charles L. A. Clarke, and Julia Kiseleva. Assessing and verifying task utility in llm-powered applications, 2024. URL https://arxiv.org/abs/2405.02178.

[41] Will Epperson, Gagan Bansal, Victor Dibia, Adam Fourney, Jack Gerrits, Erkang (Eric) Zhu, and Saleema Amershi. Interactive debugging and steering of multi-agent ai systems. In *CHI 2025*, April 2025. URL https://arxiv.org/abs/2503.02068.

[42] Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems, 2025. URL https://arxiv.org/abs/2505.00212.

[43] Claire B Draucker, Donna S Martsolf, Ratchneewan Ross, and Thomas B Rusk. Theoretical sampling and category development in grounded theory. *Qualitative health research*, 17(8): 1137–1148, 2007.

[44] Shahedul Huq Khandkar. Open coding. *University of Calgary*, 23(2009):2009, 2009.

[45] Manus AI. Manus. https://manus.im/, 2025.

[46] Anthropic. Model context protocol: Introduction. https://modelcontextprotocol.io/introduction, dec 2024.

[47] Rao Surapaneni, Miku Jha, Michael Vakoc, and Todd Segal. A2a: A new era of agent interoperability, April 2025. URL https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/. Google Developers Blog.

[48] Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. Llm-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models, 2025. URL https://arxiv.org/abs/2310.03903.

[49] Charles Perrow. *Normal Accidents: Living with High-Risk Technologies*. Princeton University Press, Princeton, NJ, 1984. ISBN 978-0691004129.

[50] Karlene H. Roberts. New challenges in organizational research: High reliability organizations. *Organization & Environment*, 3(2):111–125, 1989. doi: 10.1177/108602668900300202.

[51] Gene I Rochlin. Reliable organizations: Present research and future directions. *Journal of contingencies and crisis management.*, 4(2), 1996. ISSN 0966-0879.

[52] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[53] Huy Nhat Phan, Tien N Nguyen, Phong X Nguyen, and Nghi DQ Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.

[54] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

[55] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[56] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, 2024.

[57] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[58] Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. Does prompt formatting have any impact on llm performance? *arXiv preprint arXiv:2411.10541*, 2024.

[59] Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*, 2023.

[60] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

[61] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[62] LangChain. Langgraph, 2024. URL https://www.langchain.com/langgraph.

[63] Anthropic. Building effective agents, 2024. URL https://www.anthropic.com/research/building-effective-agents.

[64] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[65] Fatemeh Haji, Mazal Bethany, Maryam Tabar, Jason Chiang, Anthony Rios, and Peyman Najafirad. Improving llm reasoning with multi-agent tree-of-thought validator agent. *arXiv preprint arXiv:2409.11527*, 2024.

[66] Zhenran Xu, Senbao Shi, Baotian Hu, Jindi Yu, Dongfang Li, Min Zhang, and Yuxiang Wu. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv:2311.08152*, 2023.

[67] Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. Inference scaling f laws: The limits of llm resampling with imperfect verifiers. *arXiv preprint arXiv:2411.17501*, 2024.

[68] Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more llm calls all you need? towards scaling laws of compound inference systems. *arXiv preprint arXiv:2403.02419*, 2024.

[69] Kush Jain, Gabriel Synnaeve, and Baptiste Rozière. Testgeneval: A real world unit test generation and test completion benchmark. *arXiv preprint arXiv:2410.00752*, 2024.

[70] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*, 2023.

[71] Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. Question answering over knowledge bases by leveraging semantic parsing and neuro-symbolic reasoning. *arXiv preprint arXiv:2012.01707*, 2020.

[72] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

[73] Yaru Niu, Rohan R Paleja, and Matthew C Gombolay. Multi-agent graph-attention communication and teaming. In *AAMAS*, volume 21, page 20th, 2021.

[74] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*, 31, 2018.

[75] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*, 2018.

[76] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

[77] Xudong Guo, Daming Shi, Junjie Yu, and Wenhui Fan. Heterogeneous multi-agent reinforcement learning for zero-shot scalable collaboration. *arXiv preprint arXiv:2404.03869*, 2024.

[78] Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. *arXiv preprint arXiv:2410.08115*, 2024.

[79] Eric Horvitz. Uncertainty, action, and interaction: In pursuit of mixed-initiative computing. *IEEE Intelligent Systems*, 14(5):17–20, 1999.

[80] Barnali Chakraborty and Debapratim Purkayastha. Servicenow: From startup to world's most innovative company. *IUP Journal of Entrepreneurship Development*, 20(1), 2023.

[81] Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *arXiv preprint arXiv:2402.19255*, 2024.

[82] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[83] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[84] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.