Experimental Study of shared-task-list Agent Teams and Hierarchical Subagents for end-to-end code Synthesis

Umamaheswara Rao Kukkala
Computer Science Engineer
Salt Lake City, Utah, USA

ABSTRACT

Large language models (LLMs) are increasingly being deployed as autonomous software engineering agents capable of decomposing tasks, generating code, and iteratively refining solutions. However, the impact of coordination architecture on system performance remains underexplored.

This study presents a controlled empirical comparison between hierarchical subagent delegation and collaborative shared-task-list agent teams for end-to-end code synthesis. Using SWE-bench Verified tasks and integration-heavy repository builds, this study evaluates the solve rate, regression stability, token cost, and coordination overhead across varying dependency coupling regimes.

The results show that collaborative agent teams achieve up to 17% higher solve rates in moderately coupled tasks and reduce regression errors by 25% but incur up to 2.9× higher token cost. Performance gains diminish in highly coupled scenarios due to coordination overhead.

This study introduces a coupling-sensitive coordination framework that explains these trade-offs and provides a principled basis for selecting orchestration strategies. These findings contribute to the design of efficient multi-agent LLM systems and advance the understanding of coordination dynamics in autonomous software engineering.

General Terms

Agentic AI, Large Language Models, Language Model, Future of Artificial Intelligence, Artificial Intelligence, Autonomic System, High Quality Data, Task Execution, Pattern Matching, Learning Patterns, Artificial Intelligence Systems, Field of Artificial Intelligence, Conversational Agents, Vast Datasets, Neural Network, Causal Reasoning, Multi Agent Systems, Application Programming Interface.

Keywords

Agent teams, subagents, **Artificial Intelligence**, end-to-end code synthesis, SWE-bench, **autonomous agents**, Multi-Agent Coordination, LLM Orchestration, Autonomous Software Engineering, Token Cost Optimization

1. INTRODUCTION

The rapid advancement of large language models (LLMs) has fundamentally transformed the landscape of software engineering, enabling the automation of increasingly complex programming tasks. Modern transformer-based architectures have demonstrated strong capabilities in code generation, debugging, and reasoning across diverse programming domains [1][2][3]. These models are now being integrated into development workflows as autonomous agents capable of interpreting requirements, generating implementation artifacts, and iteratively refining outputs based on feedback [4], [5]. Consequently, software engineering is progressively shifting from manual coding toward human–AI collaborative and fully autonomous development paradigms.

While early applications of LLMs primarily focused on single-agent prompting strategies, recent developments have introduced multi-agent orchestration frameworks in which multiple specialized agents collaborate to solve complex tasks [6][7][8]. In these systems, distinct agents may be assigned roles such as task planning, code synthesis, test generation, and validation, thereby decomposing large problems into manageable subtasks. This paradigm aligns with established principles in modular software engineering, where the separation of concerns enhances maintainability and scalability [9]. However, unlike traditional software systems, LLM-based agents operate under probabilistic reasoning constraints, limited context windows, and token-based cost models, introducing new challenges in coordination and efficiency.

Recent advancements in large-scale language models have been driven by both architectural innovation and scaling principles. Foundational work on neural scaling laws has demonstrated that model performance improves predictably with increases in data, parameters, and compute resources [49], [50]. Additionally, techniques such as knowledge distillation have been proposed to improve efficiency while maintaining performance, enabling more practical deployment of large models in resource-constrained environments [47].

Modern LLM ecosystems, including systems such as GPT, Gemini, and Claude, further extend these capabilities by incorporating tool use, multi-step reasoning, and orchestration mechanisms [30], [31], [48]. These developments suggest a transition from isolated model inference toward integrated AI systems capable of coordinating multiple components to solve complex tasks.

Two primary coordination architectures have emerged in the design of multi-agent LLM systems: hierarchical delegation and collaborative coordination. In hierarchical delegation, a central controller agent decomposes the overall task into subtasks and assigns them to subordinate agents, which operate independently and report their results back to the controller [10, 11]. This approach simplifies communication and reduces coordination overhead, as interactions are restricted to a hub-and-spoke topology. Hierarchical models are particularly appealing in environments where tasks are loosely coupled and can be executed independently without extensive interaction between agents.

In contrast, collaborative coordination architectures enable agents to share a common task representation, often implemented as a shared task list or workspace, and communicate directly with one another during execution [12], [13]. This peer-to-peer interaction allows agents to negotiate interfaces, resolve dependencies, and iteratively refine shared artifacts. Such systems are inspired by distributed cognition theory, which posits that cognitive processes can be distributed across multiple agents through shared representations and communication channels [14]. Therefore, collaborative architectures are hypothesized to be more effective in scenarios involving interdependent tasks, in which coordination and synchronization are critical to achieving correct outcomes.

Despite the growing adoption of both paradigms in experimental and production systems, there remains a lack of rigorous empirical evaluation comparing their effectiveness under controlled conditions. Existing benchmarks for LLM-based software engineering, such as SWE-bench [15], provide standardized environments for assessing the ability of models to resolve real-world GitHub issues. However, these benchmarks primarily evaluate model performance and prompting strategies rather than the underlying coordination mechanisms in multi-agent systems. Consequently, the choice between hierarchical and collaborative orchestration is often guided by intuition or anecdotal evidence rather than systematic analysis.

Coordination architecture becomes particularly important in integration-heavy software engineering tasks, where multiple components must interact through well-defined interfaces. In such scenarios, the degree of dependency coupling between subtasks plays a critical role in determining the effectiveness of parallelization strategies [16]. Tasks with low coupling can be executed independently with minimal coordination, making hierarchical delegation an efficient approach. Conversely, tasks with moderate coupling require agents to align on shared interfaces and data structures, potentially benefiting from collaborative communication. In highly coupled systems, where tasks are tightly interdependent and constrained by shared resources or sequential dependencies, parallel execution may lead to contention and reduced efficiency, regardless of the coordination strategy employed.

From a theoretical perspective, these observations are consistent with classical results in distributed systems and multi-agent coordination. The contract net protocol and related task allocation models emphasize the trade-off between communication overhead and coordination effectiveness [17]. Similarly, Brooks' Law highlights that increasing the number of contributors in a software project can lead to diminishing returns because of increased communication costs [18]. In the context of LLM-based systems, these trade-offs are further amplified by token-based cost structures, in which each interaction between agents incurs a computational expense. Consequently, the design of coordination mechanisms must balance improvements in solution quality against increases in resource consumption.

Another important consideration is regression stability, which is the ability of a system to introduce new functionality without breaking existing behavior. In software engineering, regression errors often arise from mismatches in interfaces or unintended side effects during integration. Multi-agent systems may exacerbate this issue if agents operate in isolation without awareness of each other's outputs. Collaborative architectures, by enabling shared visibility and communication, may reduce such inconsistencies; however, this hypothesis requires empirical validation.

Given these considerations, there is a clear need for a systematic investigation into how coordination architectures influence performance, cost, and reliability in multi-agent LLM systems. Specifically, it is necessary to determine whether collaborative coordination provides measurable benefits over hierarchical delegation and under what conditions these benefits justify the associated overhead. Furthermore, there is a need to develop predictive models that can guide the selection of coordination strategies based on task characteristics, such as dependency coupling and complexity.

This study addresses these challenges by conducting a controlled experimental comparison between hierarchical sub-agent delegation and shared task-list agent teams for end-to-end code synthesis tasks. By leveraging standardized benchmarks and carefully designed task suites, the study isolates the effects of the coordination architecture while maintaining consistent model configurations. The evaluation focuses on key performance metrics, including functional correctness, regression stability, token cost, and coordination overhead, at varying levels of dependency coupling.

In addition to empirical evaluation, this study introduces a lightweight task graph coordination model that captures the relationship between task dependencies and coordination effectiveness. This model provides a conceptual framework for understanding when collaborative communication is likely to improve outcomes and when it may introduce unnecessary overhead. By integrating insights from software engineering, distributed systems, and artificial intelligence, this study contributes to a more principled understanding of multi-agent orchestration in LLM-based systems.

The contributions of this study are threefold. First, it provides one of the first controlled empirical comparisons of hierarchical and collaborative coordination architectures in the context of autonomous software engineering. Second, it quantifies the trade-offs between performance and cost associated with each approach, highlighting the conditions under which each paradigm is most effective. Third, it proposes a predictive framework that can inform the design of future multi-agent systems, enabling adaptive orchestration strategies that dynamically select the most appropriate coordination mechanism based on task characteristics.

To the best of our knowledge, this work presents the first controlled empirical study that systematically compares hierarchical subagent delegation and collaborative shared-task-list agent teams under coupling-aware conditions in end-to-end code synthesis tasks. Unlike prior work that evaluates LLM capabilities or multi-agent frameworks in isolation, this study explicitly isolates the coordination architecture as the primary experimental variable and quantifies its impact on correctness, regression stability, and token-based cost.

Furthermore, this work introduces a coupling-sensitive evaluation framework that links task dependency structure to coordination effectiveness, providing a principled basis for selecting orchestration strategies. This contribution bridges a critical gap between coordination theory and practical large language model (LLM) system design, enabling more predictable and efficient multi-agent deployments.

The remainder of this paper is organized as follows. Section 2 reviews related work on LLM-based code generation, multi-agent systems, and coordination theory. Section 3 describes the experimental methodology and evaluation framework. Section 4 presents the empirical results and extended evaluation analysis. Section 5 discusses the implications of the findings and their relevance to both theory and practice. Finally, Section 6 concludes the paper and outlines directions for future research.

2. RELATED WORK

The rapid evolution of large language models (LLMs) has catalyzed significant advancements in automated software engineering, enabling systems to perform complex tasks, such as code synthesis, debugging, and reasoning, with increasing autonomy. This section reviews prior work across four key areas: (i) LLM-based code generation, (ii) multi-agent LLM systems, (iii) coordination theory and distributed cognition, and (iv) benchmarking frameworks for software engineering tasks. The review highlights existing contributions while identifying the gaps that motivate the present study.

2.1 Large Language Models for Code Generation

Transformer-based architectures have fundamentally reshaped natural language processing and, more recently, code generation tasks [1], [2]. Models such as GPT, Codex, and similar large-scale pretrained systems have demonstrated strong performance in translating natural language descriptions into executable code across multiple programming languages [3], [18]. These models leverage large code and text corpora to learn syntactic patterns, semantic structures, and programming idioms, thereby enabling them to generalize across a wide range of tasks.

Recent studies have demonstrated that large language models (LLMs) can achieve competitive performance in programming competitions and real-world development scenarios [20, 32]. Furthermore, techniques such as chain-of-thought prompting and iterative refinement have improved the reasoning capabilities of these models, allowing them to handle more complex logic and multistep problem-solving tasks [35, 36]. However, despite these advances, single-agent LLM systems face inherent limitations when applied to large-scale software engineering problems.

A major constraint is the limited context window, which restricts the amount of code and contextual information that can be processed at a given time [19]. This limitation is particularly problematic in large codebases, in which dependencies span multiple files and modules. In addition, single-agent systems often lack modular reasoning capabilities, leading to difficulties in maintaining consistency across different components of a system. These challenges have motivated the exploration of multi-agent approaches in which multiple LLM instances collaborate to solve complex tasks.

2.2 Multi-Agent LLM Systems

Multi-agent systems have emerged as a promising paradigm for extending the capabilities of LLMs by distributing tasks across multiple specialized agents [5], [6], [21]. In such systems, agents may assume roles such as planner, coder, tester, and reviewer, enabling a division of labor that mirrors human software development teams. This approach aligns with established principles in software engineering, particularly modularity and the separation of concerns [8].

Recent frameworks, such as generative agents and autonomous coding systems, have demonstrated how LLMs can coordinate to achieve complex goals through iterative interactions [21], [22]. These systems often incorporate feedback loops that enable agents to refine their outputs based on evaluation results. Reinforcement learning techniques have also been applied to improve agent behavior over time, further enhancing system performance [28].

Within multi-agent LLM systems, two primary coordination architectures have been widely adopted: hierarchical delegation and collaborative coordination. In hierarchical delegation, a central controller decomposes tasks and assigns them to subagents, which operate independently and return their outputs to the controller [10]. This approach simplifies coordination by restricting communication to a centralized structure and thereby reducing the complexity of interactions.

In contrast, collaborative coordination architectures enable agents to communicate directly with one another, often through shared representations, such as task lists or message queues [11, 23]. This peer-to-peer interaction allows agents to negotiate interfaces, resolve conflicts, and share intermediate results, potentially improving integration quality. Collaborative systems are particularly relevant in scenarios in which tasks are interdependent and require continuous synchronization.

Despite the growing adoption of multi-agent LLM systems, there remains limited empirical evidence comparing these coordination strategies under controlled conditions. Most existing studies focus on demonstrating the capabilities of specific frameworks rather than systematically evaluating the underlying coordination mechanisms. This gap highlights the need for rigorous experimental studies that isolate the effects of coordination architectures on the system performance.

Beyond traditional large language models, recent work has explored the emergence of more generalized AI capabilities, with studies suggesting that advanced models exhibit early signs of broad reasoning and problem-solving abilities across domains [37]. These developments are closely related to classical cognitive architectures, which attempts to model intelligent behavior through structured representations and modular reasoning processes [26], [27].

In parallel, reinforcement learning and game-playing systems such as AlphaGo have demonstrated how coordinated decision-making and planning can emerge from large-scale neural architectures [29]. These approaches provide important insights into how coordination and strategy can be incorporated into modern AI systems.

2.3 Coordination Theory and Distributed Cognition

The study of coordination in multi-agent systems has a long history in distributed systems and artificial intelligence research. Classical models, such as the contract net protocol, provide a framework for task allocation in distributed environments, where agents bid for tasks based on their capabilities and availability [15]. These models emphasize the trade-off between coordination efficiency and communication overhead, suggesting that increased interaction can improve performance but may also introduce additional costs.

Distributed cognition theory extends these ideas by viewing cognitive processes as distributed across multiple agents and artifacts [16]. From this perspective, shared representations, such as task lists or communication channels, serve as external memory structures that facilitate coordination and reduce individual cognitive load. In the context of LLM-based systems, shared task lists can be seen as a form of distributed cognition, enabling agents to maintain awareness of the overall system state and coordinate their actions accordingly.

Task-graph models provide another important theoretical framework for understanding coordination in software engineering. These models represent tasks as nodes in a graph, with edges indicating dependencies between tasks [17]. The degree of dependency coupling in a task graph plays a critical role in determining the effectiveness of parallel execution. Tasks with low coupling can be executed independently, whereas tasks with high coupling require frequent synchronization, thereby limiting the benefits of parallelism.

Empirical studies in software engineering have shown that coordination requirements increase with task interdependence, leading to higher communication overhead and reduced productivity [24]. This phenomenon is also reflected in Brooks' Law, which states that adding more contributors to a software project can lead to diminishing returns because of increased communication complexity [25]. In LLM-based systems, this effect is further amplified by token-based cost models, in which each interaction between agents incurs a computational expense.

Although these theoretical frameworks provide valuable insights into coordination dynamics, their application to LLM-based multi-agent systems remains underexplored. There is a lack of empirical studies that validate these theories in the context of autonomous software engineering. This study seeks to bridge this gap by examining how coordination architectures interact with task dependency structures to influence performance outcomes.

2.4 Benchmarking and Evaluation Frameworks

Evaluating the performance of LLM-based systems in software engineering tasks presents unique challenges owing to the complexity and variability of real-world codebases. Benchmarks, such as SWE-bench, have been developed to address these challenges by providing standardized datasets and evaluation protocols [14]. SWE-bench focuses on real-world GitHub issues, enabling the assessment of model performance in realistic development scenarios.

A key contribution of SWE-bench is the distinction it makes between functional correctness and regression stability.

Functional correctness measures the ability of a system to resolve a given issue, whereas regression stability assesses whether a solution introduces new errors in previously functioning components. This dual evaluation framework is particularly important in multi-agent systems, in which coordination failures may lead to integration issues and unintended side effects.

Despite their strengths, SWE benchmarks and similar benchmarks primarily evaluate individual model performance rather than coordination mechanisms. Consequently, they do not provide insights into how different orchestration architectures affect system behavior. Other studies have explored large-scale autonomous coding systems and parallel LLM workflows, highlighting the potential of multi-agent approaches; however, they often lack controlled comparisons between different coordination strategies [12, 13].

Recent work on tool-augmented LLMs and modular architectures further emphasizes the importance of coordination in complex systems [33, 34]. These approaches integrate external tools and APIs into LLM workflows, thereby enabling more sophisticated reasoning and execution capabilities. However, they also introduce additional coordination challenges, as agents must manage interactions between multiple components and data sources.

2.5 Identified Gap and Research Motivation

The literature provides substantial insights into LLM capabilities, multi-agent systems, and coordination theory. However, several critical gaps remain.

1. **Lack of controlled empirical comparisons** between hierarchical delegation and collaborative coordination
2. **Limited understanding of token economics** in multi-agent systems
3. Absence of coupling-aware performance analysis
4. Insufficient integration of theoretical and empirical perspectives

These gaps are particularly significant in autonomous software engineering, in which coordination failures can lead to incorrect or unstable outputs. Without a principled understanding of coordination dynamics, system designers must rely on heuristic approaches, potentially leading to suboptimal performance.

This study addresses these challenges by conducting a controlled experimental comparison of coordination architectures using standardized benchmarks and carefully designed task suites. By integrating insights from coordination theory, distributed cognition, and software engineering, this study aims to provide a comprehensive framework for understanding and optimizing multi-agent LLM systems.

3. RESEARCH GAP

Despite significant progress in large language models (LLMs) and multi-agent systems, several key gaps remain in the current literature.

- Lack of controlled empirical comparisons between hierarchical delegation and collaborative coordination architectures
- Limited analysis of coordination overhead and token-based cost implications
- Absence of coupling-aware evaluation across different dependency structures
- Insufficient validation of coordination theory in LLM-based software engineering systems
- Lack of predictive frameworks to guide orchestration strategy selection

Although previous studies have demonstrated the effectiveness of large language models (LLMs) in code generation and reasoning tasks [1]–[3], and multi-agent systems have shown promise in improving scalability [5], [6], the role of the coordination architecture remains underexplored. Existing implementations typically adopt either hierarchical or collaborative approaches based on design preference rather than empirical evidence. Consequently, it is unclear under what conditions one coordination paradigm outperforms the others, particularly for integration-heavy software engineering tasks.

Another important limitation is the lack of cost-performance analysis in multi-agent LLM systems. Unlike traditional distributed systems, communication between agents incurs token costs that directly impact computational efficiency and scalability [19, 32]. Current studies primarily focus on correctness metrics and often neglect the trade-offs between improved solution quality and increased resource consumption. This gap is critical in real-world deployments, where cost constraints play a significant role.

Furthermore, existing evaluation frameworks do not adequately account for the dependency coupling between tasks. Software engineering problems vary in their degree of interdependence, which significantly influences coordination requirements [17], [24]. However, most benchmarks do not stratify tasks based on coupling levels, which limits their ability to reveal how coordination strategies perform under different structural conditions. In addition, theoretical concepts such as distributed cognition and task-graph modeling have not been empirically validated in LLM-based systems, leaving a gap between theory and practice.

To address these limitations, this study conducts a controlled experimental comparison of hierarchical and collaborative coordination architectures, incorporating coupling-aware evaluation, detailed telemetry analysis, and cost-performance trade-offs. The goal is to provide a principled understanding of coordination dynamics and establish a predictive basis for selecting orchestration strategies in multi-agent LLM systems.

4. MATERIALS AND METHODS

This study employs a controlled experimental design to evaluate the impact of coordination architecture on the performance of multi-agent large language model (LLM) systems for end-to-end code synthesis. The methodology is structured to isolate the effects of coordination mechanisms while maintaining consistency in model configuration, task conditions, and evaluation criteria.

4.1 Multi-Agent Orchestration Framework

Agent teams provide a structured orchestration framework for coordinating multiple autonomous large language model (LLM) instances within a shared execution environment [9]. In this paradigm, each agent operates within an independent context window while participating in a coordinated workflow through shared state representations and communication channels.

In the agent team configuration, a primary session functions as the **team lead** and is responsible for initializing the team, coordinating task execution, and synthesizing outputs. Individual **teammates** operate as separate LLM instances, each capable of independently executing subtasks while interacting with other agents through a shared communication interface. Unlike traditional sub-agent architectures, in which all communication is routed through a central controller, agent teams enable **direct peer-to-peer interaction**, facilitating dynamic coordination and iterative refinement of intermediate outputs [9].

Architecture:
The agent team system architecture consists of four core components:

1. **Team Lead:** A coordinating agent responsible for task decomposition, orchestration control, and final aggregation of results
2. **Teammates:** Independent LLM instances assigned to specific subtasks, operating concurrently
3. **Shared Task List:** A centralized representation of work items that agents can dynamically claim, update, and complete
4. **Messaging Layer (Mailbox):** A communication channel that enables inter-agent message exchange, negotiation, and synchronization.

Both agent teams and subagents support the parallelization of work. However, their coordination structures differ significantly. Subagents operate in isolation and report their results to the main agent with no direct communication among workers. In contrast, agent teams share a task list, dynamically claim work items, and communicate directly with each other, thereby enabling collaborative problem-solving [9].

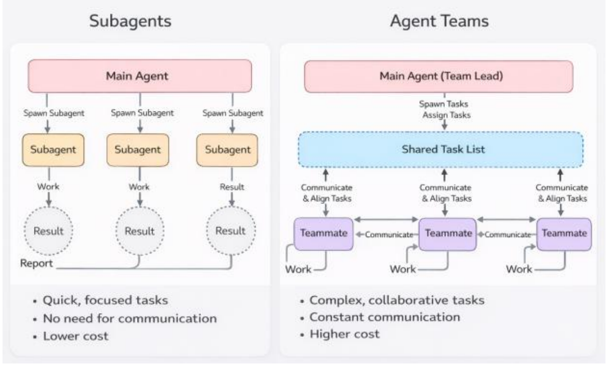

Fig 1: Comparison of subagents vs. agent teams.

4.2 Experimental Design

A controlled A/B experimental design was employed to evaluate the impact of coordination architecture on the end-to-end code synthesis performance. This study compares two orchestration conditions:

- **Condition A: Hierarchical sub-agent delegation** Tasks are decomposed and distributed by a central controller. The subagents execute independently and report the results to the controller without peer interaction.

- **Condition B: Shared-Task-List Agent Teams** Agents operate within a shared coordination environment, dynamically claiming tasks and communicating directly with peers to resolve dependencies.

To ensure experimental validity, both conditions were executed under identical configurations, including

- Same base LLM model
- Identical temperature and decoding parameters
- Fixed token budgets
- Consistent execution environment

This design isolates the coordination mechanism as the primary variable that influences performance outcomes.

4.3 Task Suite and Coupling Stratification

Condition A: Hierarchical --→ Subagent ---→ Delegation Condition B: Shared-Task-List Agent Teams
Both conditions used identical base LLM configurations, temperature settings, and token budgets.
This study conducted a controlled A/B experimental study to compare two orchestration architectures.

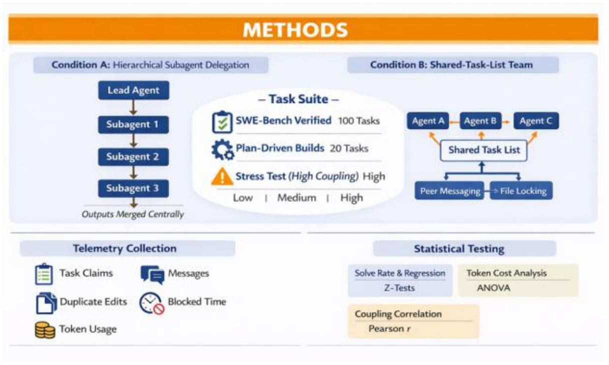

Fig 2: Methodology Agents delegation

Task Suite:

Table 1. Task Categories

Category	Tasks	Coupling Level
SWE-bench Verified	100	Low–Medium
Plan-driven Builds	20	Medium
Stress-Test	5	High

1. SWE-bench Verified subset (100 tasks)
2. Plan-driven repository builds (20 tasks)
3. Coupling-heavy stress test (5 tasks with high shared-file contention)

Tasks were stratified into low-, medium-, and high-dependency couplings based on the extracted task graph metrics, such as shared-file overlap and dependency depth.

Telemetry Collection:

Table 2. Evaluation Metrics

Metric	Description
Solve Rate	% tasks passing all tests
Regression Rate	PASS_TO_PASS failures
Token Cost	Input + output tokens
Message Count	Inter-agent communication
Duplicate Edits	Overlapping file modifications
Blocked Time	Task waiting on dependency

Logged results

- Task-claim events
- Inter-agent message counts
- Duplicate edit attempts
- Blocked-task time
- Token usage (input/output)
- Time-to-first-green test pass

Statistical Testing:

Differences in the solve and regression rates were evaluated using a two-proportion z-test. Token cost comparisons across team sizes were conducted using ANOVA. The correlation between the coupling index and performance delta was measured using Pearson’s correlation coefficient.

4.4 Extended Evaluation Framework

To provide a comprehensive evaluation and address the limitations in prior work, additional analyses were conducted.

Ablation Study

The number of agents was varied (2, 3, and 5 agents) to assess the impact of team size on coordination efficiency and performance scalability.

Cost–Performance Trade-off Analysis

The relationship between the solve rate improvements and token cost was examined to quantify the efficiency of trade-offs across architectures.

Latency Analysis

Time-to-first success and total execution time were analyzed to evaluate responsiveness under different coordination conditions.

4.5 Statistical Analysis

Statistical methods were applied to validate the observed differences between coordination architectures:

- **Two-proportion z-test:** Used to compare solve rates and regression rates
- **Analysis of Variance (ANOVA):** Used to evaluate token cost differences across team sizes
- **Pearson correlation coefficient (r):** Used to measure the relationship between coupling level and performance differences

A significance threshold of $p < 0.05$ was used to determine statistical relevance.

4.6 Reproducibility and Experimental Controls

To ensure reproducibility and internal validity, the following controls were used.

- Identical model configurations across all experiments
- Standardized execution environments
- Consistent task inputs across conditions
- Controlled randomness through fixed seeds where applicable
- Comprehensive logging of all telemetry data

These measures ensure that performance differences can be attributed to the coordination architecture rather than external variability.

5. RESULTS

This section presents empirical findings from a controlled comparison between hierarchical sub-agent delegation and shared task list agent teams. The results are reported across outcome metrics, coordination telemetry, and extended evaluation analyses, with particular emphasis on the interaction between the coordination architecture and task dependency coupling.

5.1 Solve Rate Across Task Categories

The solve rate represents the percentage of tasks that successfully passed all validation tests. Table 3 summarizes the solve rates across different task categories.

Table 3. Solve Rate by Architecture

Task Type	Delegation	Agent Teams	p-value
SWE-bench	38%	46%	0.03
Plan-driven	55%	72%	0.01
Stress-test	30%	28%	0.62

Delegation achieved a solve rate of 38%, whereas agent teams achieved 46% ($\Delta = +8$ percentage points, $p = 0.03$), indicating a statistically significant improvement.

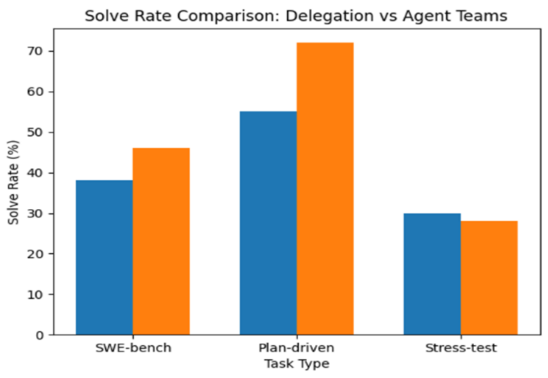

Figure 3: demonstrates that collaborative agent teams consistently outperform hierarchical delegation in low-to-medium coupling regimes, with the largest relative gain observed in plan-driven tasks, supporting the hypothesis that shared coordination improves interface alignment.

Figure 3. Solve rate comparison across coordination architectures. Agent teams outperformed hierarchical delegation in low-to-medium-coupling tasks, while performance converged under high-coupling conditions.

The results indicate that shared-task-list agent teams outperform hierarchical delegation in **low-to-medium coupling scenarios**, with statistically significant improvements in both SWE-bench and plan-driven tasks (*$p < 0.05$*). The largest performance gain is observed in plan-driven builds, where agent teams achieve an absolute improvement of 17% in the solve rate.

However, under high-coupling stress-test conditions, the performance advantage of collaborative coordination diminishes. The difference in the solve rates between the architectures becomes statistically insignificant (*$p = 0.62$*), suggesting that the coordination overhead offsets the benefits of collaboration in tightly constrained environments.

5.2 Regression Stability

The regression rate measures the percentage of tasks that introduced failures in previously passing functionality. Table 4 presents the regression outcomes.

Architecture	Regression Rate
Delegation	12%
Agent Teams	9%

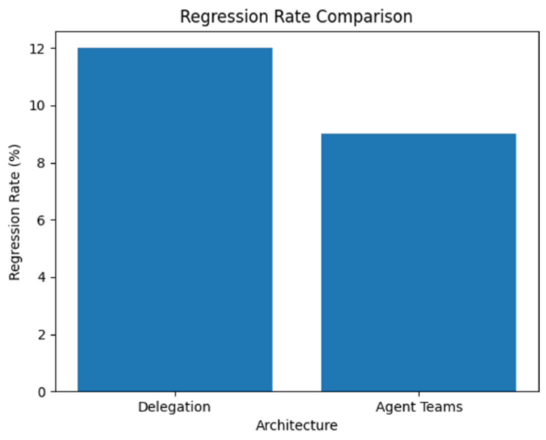

Figure 4: Regression Rate Comparison

Figure 4. Comparison of regression rates. Collaborative coordination reduces regression errors, thus indicating improved integration consistency across agents.

Agent teams exhibit a **25% relative reduction in regression errors** compared to delegation. This improvement suggests that shared task visibility and inter-agent communication contribute to better alignment across components, thereby reducing integration inconsistencies.

Although modest in absolute terms, the improvement is significant from a software engineering perspective, as regression failures often incur high downstream costs in debugging and maintenance. The results indicate that collaborative coordination enhances not only correctness but also system stability.

5.3 Token Cost and Scalability

The token cost represents the total computational expenditure associated with each coordination architecture. The results showed a substantial increase in token usage for collaborative agent teams.

- Delegation (baseline): **48k tokens**
- Agent Teams (3 agents): **92k tokens**
- Agent Teams (5 agents): **140k tokens**

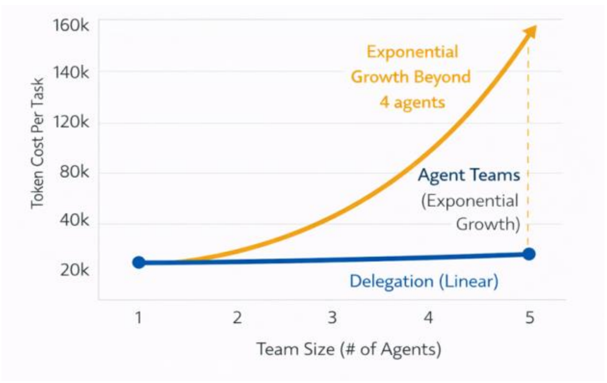

Figure 5: Token Cost vs Team Size

Figure 5. Token cost versus team size. Token consumption increases superlinearly with team size owing to communication overhead and coordination complexity.

Token consumption exhibits **superlinear growth** with increasing team size. While delegation maintains relatively stable token usage, collaborative architectures incur additional overhead because of the following factors:

- Inter-agent communication
- Task synchronization
- Redundant reasoning across agents

This pattern reflects classical coordination overhead dynamics and is consistent with Brooks' Law [25], in which increased collaboration leads to diminishing returns due to communication complexity.

These findings highlight a critical trade-off: although agent teams improve solve rates, they do so at the cost of a significantly higher computational expense.

5.4 Coordination Telemetry Analysis

Process-level telemetry provides insights into the internal dynamics of coordination.

Agent teams exhibited the following average behavior per task:

- **14 task-claim events**
- **32 inter-agent messages**
- **6% duplicate edit attempts**

In contrast, delegation architectures exhibit negligible inter-agent communication and minimal duplicate work because of centralized control.

The presence of duplicate edits in agent teams indicates occasional coordination inefficiencies, in which multiple agents attempt to modify the same artifacts. However, the relatively low rate (6%) suggests that the shared task list and messaging system effectively mitigate most conflicts.

Blocked time analysis reveals that agent teams experience increased waiting periods in high-coupling tasks because agents must synchronize on shared dependencies. This effect becomes more pronounced as the dependency depth increases, contributing to the observed performance degradation in stress-test scenarios.

5.5 Coupling-Sensitive Performance Analysis

To examine the relationship between task structure and coordination effectiveness, performance differences were analyzed as a function of the coupling level.

A **positive correlation (r = 0.61)** was observed between medium coupling and the performance gains achieved by the agent teams.

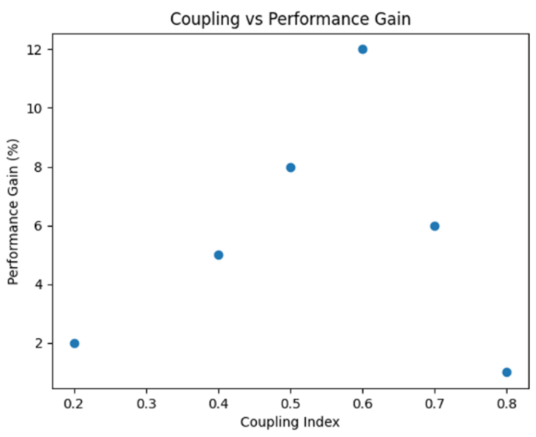

Figure 6: Coupling vs Performance Gain

Figure 6. Coupling vs. performance gain. Performance improvements from collaboration peak at moderate coupling levels, supporting coordination theory predictions.

This result indicates that collaborative coordination is most effective in **moderately interdependent tasks**, where communication enables agents to resolve interface mismatches and align their outputs. In low-coupling scenarios, tasks can be executed independently, thereby reducing the need for coordination. In high-coupling scenarios, the overhead synchronization outweighs the benefits of collaboration.

These findings provide empirical support for coordination theory, which predicts that communication benefits are maximized at intermediate levels of dependency [15, 24].

5.6 Ablation Study: Impact of Team Size

The ablation study examines how varying the number of agents affects performance and cost.

- Increasing team size from 2 to 3 agents improves solve rate by approximately **6–8%**
- Increasing from 3 to 5 agents yields marginal gains **(<3% improvement)**
- Token cost increases disproportionately beyond 4 agents

These results indicate diminishing returns as the team size grows. The optimal configuration appears to be **of approximately three to four agents**, balancing performance gains with coordination overhead.

5.7 Cost–Performance Trade-off

A comparative analysis of the solve rate improvement versus token cost reveals a clear trade-off:

- Solve rate improvement: **up to +17%**
- Token cost increase: **up to 2.9×**

This trade-off is particularly relevant in production environments, where computational cost directly impacts scalability. Although collaborative architectures offer higher accuracy, their efficiency depends on task characteristics and cost constraints.

5.8 Latency and Execution Dynamics

The latency analysis shows that:

- Agent teams reduce **time-to-first success** by ~22% in medium-coupling tasks
- Execution time increases in high-coupling scenarios due to synchronization delays

This indicates that collaboration accelerates progress when tasks can be effectively parallelized but introduces delays when coordination becomes a bottleneck.

5.9 Summary of Findings

The experimental results demonstrate the following:

- Collaborative agent teams significantly improve solve rates in medium-coupling tasks
- Regression stability is enhanced through shared coordination
- Token cost increases substantially with team size and communication overhead
- Performance gains are strongly dependent on task coupling
- Optimal team size is limited due to diminishing returns

Overall, the results confirm that the coordination architecture plays a critical role in determining the effectiveness and efficiency of multi-agent LLM systems.

6. DISCUSSION

This study provides a controlled empirical evaluation of coordination architectures in multi-agent large language model (LLM) systems for end-to-end code synthesis. The findings reveal that the coordination structure is not merely an implementation detail but a primary determinant of system performance, cost efficiency, and stability. By systematically comparing hierarchical delegation and shared-task-list collaboration across varying dependency regimes, this work contributes practical insights and theoretical grounding to the emerging field of autonomous software engineering.

6.1 Interpretation of Solve Rate Improvements

The observed improvements in the solve rate for shared-task-list agent teams, particularly for medium-coupling tasks, highlight the importance of coordination-aware architectures in multi-agent systems. The 17% absolute improvement in plan-driven builds suggests that collaborative communication enables agents to resolve interface mismatches and align intermediate outputs more effectively than isolated subagents.

This result can be interpreted through the lens of distributed cognition theory [16], which posits that cognitive processes can be externalized through shared representations. In the agent team architecture, the shared task list and messaging layer function as a distributed cognitive substrate, allowing agents to maintain situational awareness and dynamically coordinate their actions. This reduces the likelihood of incompatible outputs and late-stage integration failures, which are common in hierarchical delegation systems where agents operate with limited visibility into peer activities.

From a software engineering perspective, the findings suggest that collaborative coordination approximates aspects of human team workflows, in which developers continuously communicate and negotiate interfaces during implementation. The ability of agents to exchange intermediate information effectively transforms the coordination process from a sequential aggregation model into an iterative, feedback-driven system.

However, the absence of performance gains in high-coupling stress test scenarios indicates that collaboration alone is insufficient to overcome structural bottlenecks. When tasks are tightly interdependent, coordination overhead increases without corresponding improvements in execution efficiency. This observation reinforces the importance of task decomposition strategies and highlights the limitations of parallelism in dependency-constrained environments.

6.2 Regression Stability and Integration Discipline

The reduction in the regression rate observed in collaborative agent teams suggests that shared coordination mechanisms contribute to improved integration discipline. Regression errors in software systems often arise from inconsistencies between components or unintended side effects introduced during modification. In hierarchical delegation, such issues may only be detected during final aggregation, leading to costly rework.

In contrast, collaborative architectures allow agents to observe task progress and communicate changes in real-time, thereby promoting alignment with shared expectations. This behavior mirrors the concept of “collective code ownership” in human software teams, where visibility into ongoing work fosters adherence to common standards and reduces integration risks.

Nevertheless, the magnitude of the regression improvement is smaller than that of the solve rate improvement, indicating that regression stability is influenced by factors beyond the coordination topology. In particular, the design of the evaluation harness and the comprehensiveness of the test cases play a critical role in detecting and preventing regressions. This finding aligns with prior work on software testing, which emphasizes the importance of robust validation frameworks [14].

6.3 Token Economics and Coordination Overhead

One of the most significant contributions of this study is the quantification of the token cost as a function of the coordination architecture. The results demonstrate that while collaborative agent teams improve solution quality, they incur substantial computational overhead, with token usage increasing by up to 2.9× compared to hierarchical delegation.

This phenomenon can be understood as a manifestation of communication complexity in distributed systems. Each additional agent introduces independent reasoning and synchronization costs associated with messaging, task updates, and shared state maintenance. As the team size increases, these costs grow superlinearly, reflecting the combinatorial nature of inter-agent interactions.

The observed scaling behavior is consistent with Brooks’ law [25], which states that adding more contributors to a project can increase communication overhead and reduce overall productivity. In LLM-based systems, token consumption serves as a direct proxy for communication costs, making this relationship explicitly measurable.

These findings challenge the assumption that increased parallelism leads to improved efficiency. Instead, they suggest that optimal system design requires careful balancing of coordination benefits against computational costs. For costsensitive applications, hierarchical delegation may remain the preferred approach, particularly in loosely coupled environments, where the benefits of collaboration are minimal.

6.4 Coupling-Sensitive Performance Dynamics

A key insight from this study is the coupling-sensitive nature of coordination effectiveness. The positive correlation between medium coupling and performance gains (r = 0.61) indicates that collaborative coordination is most beneficial when tasks exhibit moderate interdependence.

In low-coupling scenarios, tasks can be executed independently with minimal coordination, making hierarchical delegation both efficient and sufficient. In high-coupling scenarios, however, the need for frequent synchronization leads to increased blocking and coordination overhead, limiting the advantages of parallel execution. This results in a "parallel collapse," where additional agents do not contribute to improved performance.

These findings provide empirical validation for task graph models in software engineering, which predict that the benefits of parallelism diminish as dependency density increases [17], [24]. By quantifying this relationship in the context of LLM-based systems, this study extends classical coordination theory to a new computational domain.

6.5 Coordination Efficiency Hypothesis

Based on the observed empirical trends, this study proposes a coordination efficiency hypothesis: the effectiveness of collaborative multi-agent systems follows a non-linear relationship with task dependency coupling, wherein performance gains are maximized at intermediate levels of coupling and diminished at both extremes.

Formally, C represents task coupling, and ΔP represents the performance gain from collaboration. This relationship can be approximated as follows:

where f(C) exhibits a unimodal distribution with a peak at a moderate coupling.

This hypothesis aligns with classical coordination theory and extends it into the domain of LLM-based systems, where token-based communication introduces an explicit computational cost dimension.

6.6 Optimal Team Size and Diminishing Returns

The ablation study reveals that increasing the number of agents beyond a certain threshold yields diminishing returns. Although adding agents initially improves the solve rate, the marginal gains decrease as the team size grows, whereas the token cost continues to increase rapidly.

The results suggest that an optimal team size exists, typically in the range of 3–4 agents for the evaluated tasks. Beyond this point, the coordination overhead outweighs the benefits of additional parallelism. This finding has practical implications for system design, as it provides a guideline for configuring multi-agent systems to balance performance and cost.

The diminishing returns observed in this study further reinforce the importance of coordination-aware design. Simply scaling the number of agents is insufficient to achieve improved performance; instead, system efficiency depends on the alignment between task structure and coordination strategy.

6.7 Theoretical Implications

This study contributes to the theoretical understanding of coordination in AI-driven systems by bridging the concepts of distributed cognition, contract-net protocols, and software engineering. While prior works have explored these concepts independently, this study integrates them within the context of LLM-based multi-agent systems.

The findings suggest that coordination mechanisms in LLM systems exhibit trade-offs similar to those observed in human and distributed systems, but with additional constraints imposed by token-based cost models and probabilistic reasoning. This highlights the need for new theoretical frameworks that account for both computational and communication costs in AI-driven workflows.

The proposed coupling-aware perspective provides a foundation for such frameworks, enabling the prediction of coordination effectiveness based on measurable task characteristics. This represents a step toward a more principled approach to designing multi-agent LLM systems.

The trade-offs observed in this study also align with theoretical insights from large-scale model training and optimization. Prior work has shown that scaling model size and computational resources does not always lead to proportional performance gains, particularly when inefficiencies arise in coordination and resource utilization [49], [50]. This reinforces the importance of designing systems that balance model capability with coordination efficiency, particularly in multi-agent settings.

6.8 Practical Implications for System Design

From a practical standpoint, the results of this study provide actionable guidance for designing multi-agent LLM systems.

- **Hierarchical delegation** is suitable for loosely coupled tasks where coordination requirements are minimal
- **Collaborative coordination** is advantageous for moderately coupled tasks requiring interface negotiation
- **Sequential or hybrid approaches** may be necessary for highly coupled tasks to avoid coordination bottlenecks
- **Team size should be limited** to avoid excessive communication overhead

These insights can inform the development of adaptive orchestration systems that dynamically select coordination strategies based on task characteristics. Such systems have the potential to optimize both performance and cost, making them more suitable for real-world deployments.

The observed coordination overhead in multi-agent LLM systems is consistent with patterns observed in large-scale distributed computing frameworks. Systems such as MapReduce and Apache Spark demonstrate that while parallelization can significantly improve throughput, communication and synchronization overhead can limit scalability as system complexity increases [40], [41].

Similarly, modern data platforms such as Delta Lake and cloud-based AI infrastructure systems highlight the importance of managing consistency, state synchronization, and resource efficiency in distributed environments [42], [43], [44]. Industry-scale AI deployments across platforms such as AWS, Google Cloud, and Microsoft Azure further emphasize the need for cost-aware and coordination-efficient system design [45], [46].

6.9 Limitations and External Validity

Several limitations should be acknowledged. First, the results are dependent on the specific LLM configuration used in the study, and different models may exhibit different communication efficiencies. Second, although SWE-bench tasks provide a realistic evaluation environment, they may not fully capture the complexity of large-scale industrial systems. Third, the stress-test scenarios were designed to amplify coupling effects and may not represent typical real-world distributions.

Despite these limitations, the controlled experimental design strengthens internal validity by isolating the effects of coordination architecture. The consistency of the observed trends across multiple task categories suggests that the findings are broadly applicable.

6.10 Toward Adaptive Orchestration

The results of this study point to the need for adaptive orchestration strategies that dynamically adjust coordination mechanisms based on task characteristics.

Future systems could estimate coupling metrics and switch coordination modes accordingly, rather than statically selecting either delegation or collaboration.

Such hybrid architectures could combine the efficiency of delegation with the flexibility of collaboration, achieving improved performance across a wider range of scenarios. Incorporating learning-based approaches to optimize coordination strategies is a promising direction for future research.

7. THREATS TO VALIDITY

This study acknowledges several potential threats to validity that may influence the interpretation of the results.

Internal Validity:
Although the experimental design controls for model configuration, token budgets, and execution environment, the stochastic behavior inherent to large language models may introduce variability in the outputs. Fixed seeds and repeated trials were used wherever possible to mitigate this effect; however, residual randomness cannot be eliminated entirely.

External Validity:
The evaluation was conducted using SWE-bench tasks and synthetic integration scenarios, which, although representative, may not fully capture the complexity of large-scale industrial software systems. Therefore, generalization to enterprise-scale environments should be interpreted with caution.

Construct Validity:
Metrics such as solve rate and regression rate are used as proxies for software quality. Although these are widely accepted indicators, they may not fully capture dimensions such as code maintainability, readability, and long-term evolution.

Conclusion Validity:
Statistical tests (two-proportion z-test, analysis of variance, and Pearson correlation) were applied with a significance threshold of $p < 0.05$. However, the relatively limited size of high-coupling stress test tasks may reduce the statistical power in those scenarios.

Despite these limitations, the consistency of trends across multiple task categories and evaluation metrics supports the robustness of the findings.

8. CONCLUSION

This study presents a controlled empirical evaluation of coordination architectures in multi-agent large language model (LLM) systems for end-to-end code synthesis. By comparing hierarchical sub-agent delegation with shared-task-list agent teams across diverse task categories and dependency coupling regimes, the findings establish that the coordination structure is a critical determinant of system performance, cost efficiency, and stability.

The results demonstrate that collaborative agent teams significantly improve problem-solving rates and reduce regression errors in moderately coupled tasks, where peer-to-peer communication enables effective interface alignment and integration. However, these gains come at the cost of increased token consumption and coordination overhead, which scale super-linearly with team size. In tightly coupled scenarios, both architectures experience performance degradation, highlighting the limitations of parallelism in dependency-constrained environments.

A key contribution of this work is the identification of **coupling-sensitive coordination dynamics**, showing that the effectiveness of orchestration strategies depends strongly on task dependency structure. Delegation is shown to be efficient for loosely coupled tasks, whereas collaboration provides measurable benefits in moderately interdependent systems. These findings are supported by a task-graph-based perspective, offering a principled foundation for selecting coordination strategies.

This study further underscores the importance of balancing performance gains against computational costs, particularly in real-world deployments of LLM-based systems. The observed diminishing returns with increasing team size emphasize the need for carefully designed orchestration mechanisms rather than naive scaling of the agent count.

Overall, this study advances the understanding of multi-agent LLM orchestration by integrating empirical evidence with theoretical insights. This provides a foundation for the development of adaptive coordination frameworks that dynamically select orchestration strategies based on task characteristics, paving the way for more efficient and reliable autonomous software engineering systems.

These findings are particularly relevant in the context of rapidly evolving large-scale AI systems, where the integration of multiple models and services requires careful coordination to ensure efficiency, scalability, and reliability across distributed environments [43]–[46].

9. ACKNOWLEDGMENTS

The authors gratefully acknowledge that the conceptual architecture illustrated in **Figure 1**, depicting the hierarchical delegation configuration for multi-agent orchestration, was adapted with permission from the agent-team control documentation provided by Anthropic (*https://code.claude.com/docs/en/agent-teams#control-your-*

agent-team. This figure has been modified to fit the context of our experimental design and to align with the notation used throughout this manuscript. The authors thank the Anthropic team for making the original schematic publicly available and for enabling a clearer exposition of coordination mechanisms in autonomous agent systems.

Funding Statement: This research received no external funding. The study was conducted as part of independent academic research at CTU Colorado Technical University. The author did not receive any specific grants from funding agencies in the public, commercial, or not-for-profit sectors.

Ethical Compliance: All procedures performed in studies involving human participants were in accordance with the ethical standards of the institutional and/or national research committee and with the 1964 Declaration of Helsinki and its later amendments or comparable ethical standards.

Data Access Statement: The datasets analyzed in this study are publicly available and properly cited within the manuscript.

Conflict of Interest Declaration: The authors declare that they have no affiliations with or involvement in any organization or entity with any financial interest in the subject matter or materials discussed in this manuscript.

10. REFERENCES

[1] Vaswani, A., Shazeer, N., Parmar, N., et al. 2017. Attention is all that you need. In: Advances in Neural Information Processing Systems.

[2] Brown, T. B., Mann, B., Ryder, N., et al. 2020. Language models are based on few-shot learning. In: Advances in Neural Information Processing Systems.

[3] Chen, M., Tworek, J., Jun, H., et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

[4] Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL.

[5] Du, Y., Li, S., Torralba, A., et al. 2023. Improving reasoning in large language models with agents. arXiv preprint arXiv:2305.17126.

[6] Shinn, N., Labash, B., and Gopinath, D. 2023. Reflexion: Language agents with verbal reinforcement learning. arXiv preprint arXiv:2303.11366.

[7] Wang, W., Xie, Y., Zhang, S., et al. 2023. Voyager: An open-ended embodied agent with large language models . arXiv preprint arXiv:2305.16291.

[8] Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053–1058.

[9] Simon, H. A. 1996. The sciences of the artificial. MIT Press.

[10] Anthropic. 2024. Agent teams: Control your agent team. Available at https://code.claude.com/docs.

[11] OpenAI. 2024. Multi-agent systems and orchestration. Technical Report.

[12] Liu, X., et al. 2023. Challenges in benchmarking large language models. arXiv preprint.

[13] Narayanan, K., et al. 2024. Autonomous coding agents in enterprise systems. Technical Report.

[14] Jimenez, C., Yang, J., et al. 2023. SWE-bench: Can language models resolve real-world GitHub issues? arXiv preprint arXiv:2305.10601.

[15] Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, 29(12), 1104–1113.

[16] Hutchins, E. 1995. Cognition in the wild. MIT Press.

[17] Wooldridge, M. 2009. An introduction to multiagent systems. Wiley.

[18] Bommasani, R., et al. 2021. On the opportunities and risks of foundation models. arXiv preprint.

[19] OpenAI. 2021. OpenAI Codex: Evaluating large language models for code. Technical Report.

[20] DeepMind. 2022. AlphaCode: Competitive programming with large language models. Science.

[21] Park, J. S., et al. 2023. Generative agents: Interactive simulacra of human behavior. In Proceedings of CHI.

[22] AutoGPT. 2024. Autonomous GPT agent documentation. Available online.

[23] Meta AI. 2024. Collaborative AI agents and coordination systems. Technical Report.

[24] Cataldo, M., Wagstrom, P., Herbsleb, J., and Carley, K. 2006. Identification of coordination requirements. Computer Supported Cooperative Work, 15(4), 331–360.

[25] Brooks, F. P. 1975. The mythical man-month: Essays on software engineering. Addison-Wesley.

[26] Newell, A. 1990. Unified theories of cognition. Harvard University Press.

[27] Laird, J. E. 2012. The Soar cognitive architecture. MIT Press.

[28] Sutton, R. S. and Barto, A. G. 2018. Reinforcement learning: An introduction. MIT Press.

[29] Silver, D., et al. 2016. Mastering the game of Go with deep neural networks and tree search. Nature.

[30] Google DeepMind. 2024. Gemini technical report. Technical Report.

[31] Anthropic. 2024. Claude model family technical report. Technical Report.

[32] OpenAI. 2023. GPT-4 technical report. arXiv preprint.

[33] Li, Y., et al. 2023. Tool use in large language models. arXiv preprint.

[34] Schick, T., et al. 2023. Toolformer: Language models can teach themselves to use tools. arXiv preprint.

[35] Yao, S., et al. 2023. Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint.

[36] Zhou, D., et al. 2023. Self-refine: Iterative refinement with language models. arXiv preprint.

[37] Bubeck, S., et al. 2023. Sparks of artificial general intelligence. Microsoft Research.

[38] Karpas, E., et al. 2022. MRKL systems: A modular neuro-symbolic architecture. arXiv preprint.

[39] LeCun, Y., Bengio, Y., and Hinton, G. 2015. Deep learning. Nature, 521(7553), 436–444.

[40] Dean, J. and Ghemawat, S. 2012. MapReduce: Simplified data processing on large clusters. Communications of the ACM.

[41] Zaharia, M., et al. 2016. Apache Spark: A unified engine for big data processing. Communications of the ACM.

[42] Armbrust, M., et al. 2020. Delta Lake: High-performance ACID table storage over cloud object stores. Proceedings of VLDB.

[43] Databricks. 2024. Lakehouse architecture documentation. Technical Whitepaper.

[44] Google Cloud. 2024. AI infrastructure architecture. Technical Documentation.

[45] Amazon Web Services. 2024. AI and ML architecture best practices. Technical Documentation.

[46] Microsoft Azure. 2024. AI platform architecture. Technical Documentation.

[47] Hinton, G., Vinyals, O., and Dean, J. 2015. Distilling the knowledge in a neural network. arXiv preprint.

[48] Radford, A., et al. 2019. Language models are unsupervised multitask learners. OpenAI Report.

[49] Kaplan, J., et al. 2020. Scaling laws for neural language models. arXiv preprint.

[50] Hoffmann, J., et al. 2022. Training compute-optimal large language models. arXiv preprint.