Exploring Performance Predictability for Multi-Agent System

Anonymous ACL submission

Abstract

Multi-Agent Systems (MAS) built from Large Language Models (LLMs) offer significant potential for complex problem-solving, yet their optimal configuration is challenging, with performance typically evaluable only after resource-intensive execution. Addressing the underexplored area of MAS performance predictability, this paper investigates whether and how accurately MAS outcomes can be forecasted. We propose and evaluate a methodology that involves monitoring MAS operations during execution, capturing agent inputs and outputs, and transforming this data into system-specific statistical indicators. These indicators are then used to train a regression model to predict overall task performance. Conducting experiments across five distinct MAS architectures and three benchmark tasks, we demonstrate that MAS performance is significantly predictable, achieving Spearman rank correlations typically ranging from **0.76** to **0.94** between predicted and actual scores. Notably, our findings indicate that the global statistics required for these predictions can be accurately estimated from as little as 10% of the total operational data-generating events, still yielding a high correlation of **0.82**. Further analysis reveals that metrics quantifying individual agent capabilities are the most influential factors in performance prediction. This work underscores the feasibility of reliably predicting MAS performance, offering a path towards more efficient design, configuration, and deployment of MASs.

1 Introduction

Recently, the rapid development of LLMs has been widely reported (Achiam et al., 2023; Dubey et al., 2024; Gemini et al., 2023). These models exhibit strong capabilities, achieving success in various tasks of Natural Language Processing (NLP) (Radford et al., 2019). Leveraging training processes such as instruction tuning (Longpre et al., 2023), LLMs have demonstrated the ability to articulate reasoning (Wei et al., 2022; Yao et al., 2024), self-correct errors (Madaan et al., 2024), utilize external tools (Schick et al., 2024; Qin et al., 2023), and retain long-term memory (Huang et al., 2023) during inference. By combining these capabilities with various techniques, researchers have successfully built on off-the-shelf LLMs to create single-agent systems capable of solving more complex tasks. Notable examples include AutoGPT (Significant Gravitas, 2024), XAgent (XAgent, 2023), and OpenInterpreter (OpenInterpreter, 2023).

Beyond the aforementioned single-agent applications, research has emerged that focuses on enabling multiple LLMs to collaborate on specific tasks. Inspired by evidence of collective intelligence (Woolley et al., 2010) arising in groups of humans, various multi-agent frameworks have been proposed to mimic human collaborative scenarios. Typically, in these frameworks, each agent is controlled by an LLM with an assigned role, and a predefined executable pipeline is configured. Following the pipeline, agents collaborate towards a common goal. This approach has shown promising results, demonstrating that a well-configured LLM-based **M**ulti-**A**gent **S**ystem (MAS) can outperform a single agent in certain contexts. Notable successes include Generative Agents (Park et al., 2023), which simulates human society, AutoGen (Wu et al., 2023), CAMEL (Li et al., 2023), AgentVerse (Chen et al., 2023) which tackles reasoning tasks, ChatEval (Chan et al., 2023) which tackles evaluation tasks, as well as ChatDev (Qian et al., 2023) and MetaGPT (Hong et al., 2023), which focus on software tasks.

Despite the significant success of these MASs, obtaining the optimal MAS configuration remains an unresolved challenge. The process often requires careful design, relying on prior knowledge of the task and heuristic approaches. The effectiveness of the chosen configuration can only be eval-

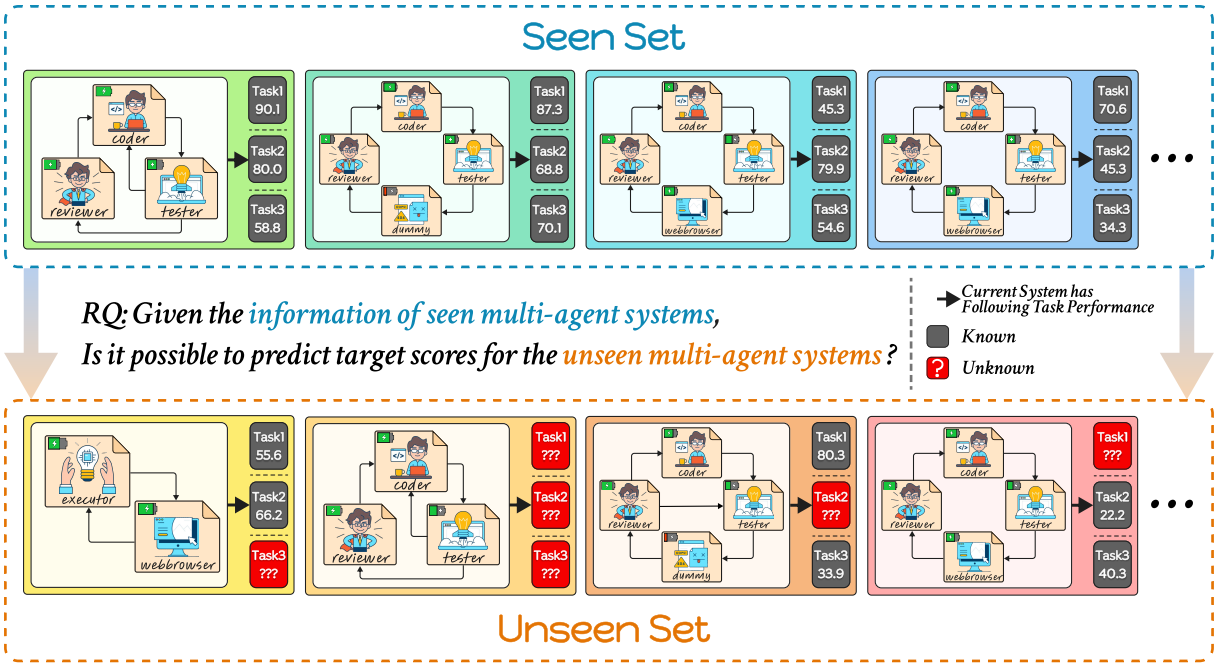


Figure 1: We investigate the research question: "Given knowledge of existing MAS and their corresponding target scores, how accurately can we predict the performance of a new MAS on an unseen task?" As illustrated in the figure, different configurations of MAS are presented. We use to represent the capabilities of the underlying LLM for each agent. For instance, this could be Llama3-8B, Llama3-70B, or other models.

uated after the actual execution, which can be resource intensive and inefficient during production. Inspired by well-studied *scaling laws*(Kaplan et al., 2020) in LLM development – which model target task performance(Isik et al., 2024) or validation loss as functions of model size, data size (Hu et al., 2024), training FLOPs (Hoffmann et al., 2022), or data mixtures (Ye et al., 2024) – we aim to explore whether it is possible to predict downstream task performance given the task and the configuration. Such predictability would enable us to design more reliable and effective MAS without the need for costly trial and error.

Additionally, recent literature has made considerable efforts to predict the performance of individual LLM, with studies examining aspects like benchmark performance predictability (Schellaert et al., 2025; Pacchiardi et al., 2025), methods for explaining predicted performance (Drapal et al., 2024), and the ability to forecast success on specific instances from limited data (Pacchiardi et al., 2024). However, the predictability of overall MAS performance, which arises from complex emergent dynamics due to agent interactions, remains comparatively underexplored. This motivates our central research question: ***How predictable is the performance of a MAS?***

To address this, we investigate a method based on monitoring system execution. During the operation of a MAS, we capture inputs and outputs each time an agent communicates. This collected data, reflecting the dynamic interactions, is transformed into system-specific indicators (Section 3) designed to predict target scores. We then train a simple regression model, such as XGBoost (Chen and Guestrin, 2016), on these indicators. This model subsequently allows us to predict the performance of a newly configured MAS on its target tasks.

Specifically, in this paper, we conducted experiments using five distinct, manually designed architectures with varying agent assignments and message flows. These were tested across three benchmark tasks: HumanEval (Chen et al., 2021), MMLU (Hendrycks et al., 2020), and GSM8K (Cobbe et al., 2021). Our findings reveal several key insights into MAS predictability. **Firstly**, the performance scores predicted by our regression model, which is trained on operational statistics, demonstrate a high consistency with actual observed values. We achieved a Spearman rank correlation typically ranging from **0.76** to **0.94** indicating that MAS performance is, to a significant extent, predictable. **Secondly**, while deriving comprehensive operational statistics for the prediction model typically requires extensive MAS execution–a potentially costly and inefficient pro-

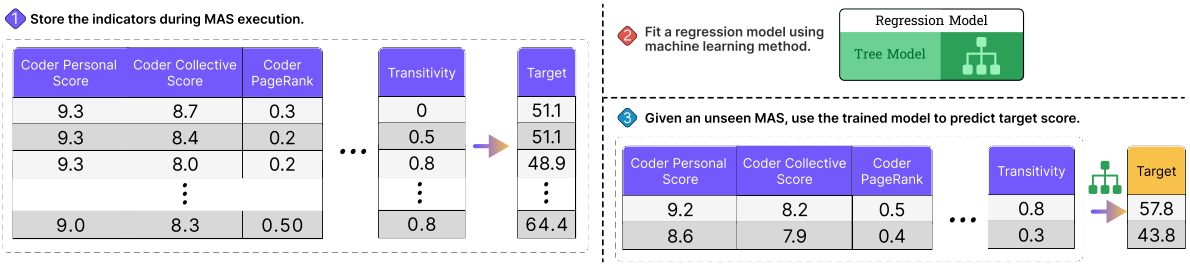


Figure 2: Data collected during MAS execution are transformed into statistics that are used to train a performance prediction model.

cess, our experiments revealed that the key global statistics needed for prediction can be reliably estimated by observing and processing only 10% of the total data-generating events from the MAS. Impressively, using these estimated global statistics as input to the predictive model still yielded a high Spearman rank correlation of **0.82**. **Thirdly**, a finer-grained analysis of the collected statistics revealed that metrics quantifying individual agent capabilities are, intuitively, the most influential factors in predicting overall system performance. Interestingly, many less important statistics still exhibited considerable variance; their lower predictive impact is therefore not merely due to a lack of variation but rather suggests that accurately capturing core agent competencies is of primary importance for the tasks evaluated, with other interaction statistics playing a secondary, though still complex, role.

2 Related Work

2.1 LLM Based Agents and Multi-Agent FrameWork

Recent advances in LLMs, such as GPT-4 (Achiam et al., 2023), have stimulated the development of LLM-based agents. These agents are able to utilize external tools, such as interpreters (OpenInterpreter, 2023), search engines (Luo et al., 2023; Chan et al., 2024), web browsers (Nakano et al., 2021; He et al., 2024), or custom-defined tools (Qin et al., 2023; Schick et al., 2024) through function calling. Leveraging the strong instruction-following abilities of foundation models, these agents have demonstrated significant progress in various domains. For example, the development of OS-Copilot, which integrates with operating systems (Wu et al., 2024), the creation of XAgent for solving complex tasks (XAgent, 2023), and the introduction of SearchGPT to accelerate search experiences (OpenAI, 2024). In line with these advances, frameworks have emerged for efficiently building LLM agents, such as LangChain (LangChain-AI, 2024), AgentGPT (Reworkd, 2024), and AutoGPT (Significant Gravitas, 2024).

Beyond single-agent intelligence, recent research indicates that collaboration among multiple agents, each with different expertise, can enhance downstream task performance. Notable successes include AutoGen (Wu et al., 2023), which facilitates the creation of conversable agents for various pilot applications, such as online decision-making; OpenHands (Wang et al., 2024), a platform for developing powerful and flexible AI agents that interact with the world in ways similar to those of human developers; IOA (Chen et al., 2024), which addresses the challenges of distributed agent deployment by introducing an agent integration protocol, along with a design of an instant messaging architecture. However, constructing reliable MAS with these frameworks often involves trial and error in identifying the optimal configuration.

2.2 LLM Predictablility and Scaling Laws

The vast development of LLMs is closely related to the concept of neuron scaling laws (Kaplan et al., 2020; Rae et al., 2021; Henighan et al., 2020). Previous works have attempted to capture the relationships between factors such as training FLOPS and model size, and their impact on validation loss by first training numerous differently configured models and then proposing a power law to fit the coefficients. Once fitted, this power law can be used to extrapolate and predict the loss for a larger model and further simulated to derive the optimal configuration for target size model. This paradigm has led to several practical and constructive suggestions. For example, Chinchilla law (Hoffmann et al., 2022) suggests that while given a computational budget of 10x, the suggested model size should be 5.5x larger, while training tokens should

be 1.8x more. Similarly, Minicpm (Hu et al., 2024) derive optimal batch size and learning rate configurations from LLM sandbox experiments where they train a multi-set of smaller models, showing that their 2.4B model performs on par with current 7-13B scale models. MM1 (McKinzie et al., 2024) performed a grid search for the optimal learning rate using smaller models and then successfully extrapolated the results to larger scales. BIMIX (Ge et al., 2024) proposed a bivariate law concerning data quantity and mixing proportion, demonstrating that their optimized data mixture outperforms the default mixture.

Inspired by the fruitful results in the construction of LLM, our work aims to investigate the predictability of MAS by capturing relevant indicators to predict target scores. Another study (Qian et al., 2024) explored collaborative scaling laws by increasing the number of agents in a system, finding that normalized solution quality follows a logistic growth pattern as the number of agents increases. However, given the versatility required in building different MAS, it is challenging to determine a single-variable law for the entire system.

The broader goal of predicting the performance of individual LLM has been explored in previous literature (Schellaert et al., 2025; Drapal et al., 2024; Pacchiardi et al., 2024, 2025; Ye et al., 2023). While such research focusing on individual LLM capabilities is valuable, the distinct challenge of forecasting the emergent, overall performance of MAS–a domain we've highlighted as comparatively underexplored–requires a dedicated approach. Our work shares the high-level objective of performance prediction with these studies; however, to the best of our knowledge, it is the first systematic effort to specifically investigate and forecast the task performance in the context of building and configuring MAS. We hope that this research provides valuable insights and paves the way for the community to build better MAS.

3 Indicators Used to Predict Performance for MASs

In this section, we introduce the indicators used to train the prediction model. Our indicators fall into two main categories. The first group consists of scores generated by using an LLM to assess performance, including the agent's personal score and the collective score. Intuitively, the personal score measures how well an agent completes its own task following the given instruction, while the collective score evaluates how the agents' behaviour contributes to the overall system. For example, an agent given the instruction to generate helpless or nonsense responses might excel at its specific task and receive a high personal score. However, it would earn a low collective score, as it does not contribute significantly to the final result. Specifically, We records the input to each agent, each agent's output, and the conversation history. We then use these records to prompt an LLM, using the prompts detailed in Appendix E.1, to generate the corresponding scores. Both scores are rated on a scale from 0 to 10, with higher scores indicating better performance. Note that the scores are averaged across all turns and instances.

The second category includes indicators that are either inherited from or reflect the configuration of the MAS. These indicators are fixed after the system's construction (e.g., number of nodes) or are strongly influenced by the configuration (e.g., each agent's PageRank).

The indicators are detailed as follows:

- **Number of Nodes**: Each agent in the execution graph is represented as a node, so the number of nodes corresponds to the total number of agents in the system.

- **Number of Edges**: We use directed edges to represent the information flow between agents. For example, if Agent A communicates with Agent B, a directed edge is drawn from A to B, and vice versa.

- **Agent Capability**: We assign an integer to represent the capability of each agent, depending on the level of LLM controlling it. In our experiments, we assign Llama3-70B-Instruct a score of 3, Llama3-8B-Instruct and its uncensored variant a score of 2, and GPT-3.5-turbo-1106 a score of 1. These rankings are intuitively derived from the leaderboard at `https://tatsu-lab.github.io/alpaca_eval/`, though the ranking may vary slightly across different benchmarks.

- **Agent PageRank**: We calculate the weighted PageRank for each agent, treating the edge weight as the number of tokens sent and received by the agent. PageRank (Page et al., 1999) is an algorithm that measures the importance of web pages, based on the idea that a

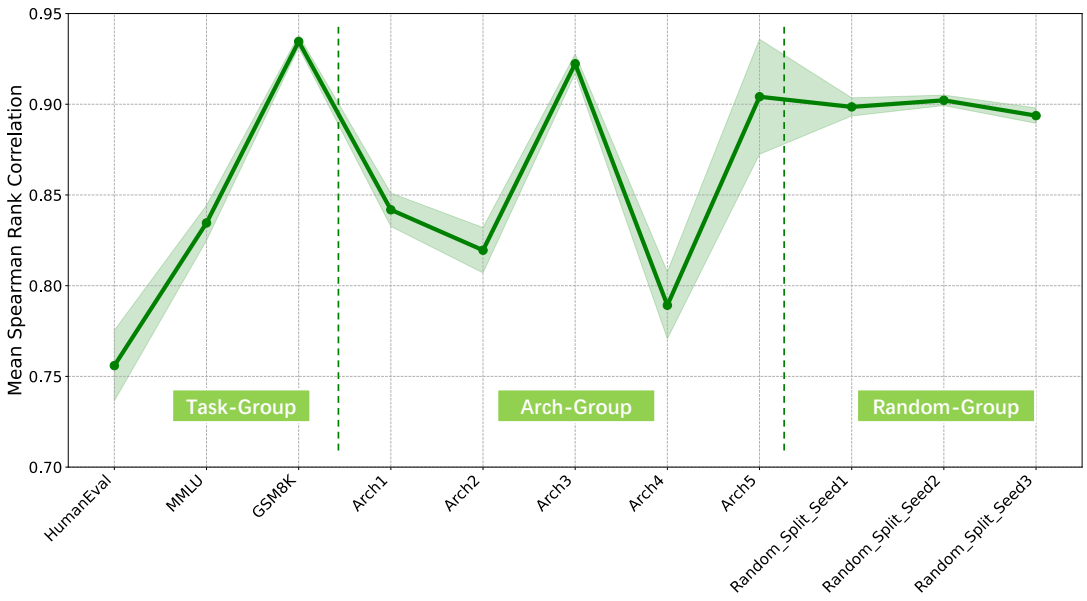


Figure 3: Spearman rank correlation between the predicted score and the observed score.

page with many incoming links is more important. Additionally, pages that are linked by other high-PageRank pages further increase their own importance. Here, we use agent PageRank to indicate the importance of each agent within the system.

$$PR(i) = \frac{1-\alpha}{N} + \alpha \sum_{j \in M(i)} \frac{w_{ji} \cdot PR(j)}{\sum_{k \in L(j)} w_{jk}} \tag{1}$$

Where:

- $PR(P_i)$ is the PageRank of agent P_i.
- d is the damping factor (set to 0.85 in our paper).
- $M(P_i)$ is the set of agents that link to P_i.
- w_{ji} is the weight of the link from agent P_j to agent P_i (we use token sent and received as weight in our paper).
- $L(P_j)$ is the set of agents that P_j links to.

- **Average Clustering** is the mean of the local clustering coefficients of all the nodes in the network where the clustering coefficient measures the degree to which nodes in a network tend to cluster together. The local clustering coefficient C_i for a node i with degree k_i is:

$$C_i = \frac{2 \times e_i}{k_i(k_i - 1)} \tag{2}$$

where e_i is the number of edges between the neighbors of node i.

The average clustering coefficient is:

$$\text{Average Clustering} = \frac{1}{N} \sum_{i=1}^{N} C_i \tag{3}$$

- **Transitivity** measures the overall tendency of a network to form triangles. It is the ratio of the number of closed triplets (triangles) to the total number of triplets (open and closed) and is defined as:

$$T = \frac{3 \times \text{Number of Triangles}}{\text{Number of Connected Triplets of Nodes}} \tag{4}$$

- **Degree Centrality** is the mean of the degree centralities of all the nodes in the network. where the degree centrality is the number of edges connected to a node defined as D_i for a node i is:

$$D_i = \frac{k_i}{N-1} \tag{5}$$

where k_i is the degree of node i, and N is the number of nodes in the network.

$$\text{Average Degree Centrality} = \frac{1}{N} \sum_{i=1}^{N} D_i \tag{6}$$

- **Closeness Centrality** is the mean of the closeness centralities of all the nodes in the network, where Closeness centrality is the reciprocal of the average shortest path distance from a node to all other nodes.

  The closeness centrality C_i for a node i is:

$$C_i = \frac{N-1}{\sum_{j \neq i} d(i,j)} \tag{7}$$

  where $d(i,j)$ is the shortest path distance between nodes i and j.

$$\text{Closeness Centrality} = \frac{1}{N} \sum_{i=1}^{N} C_i \tag{8}$$

- **Betweenness Centrality** is the mean of the betweenness centralities of all the nodes in the network where betweenness centrality measures how often a node appears on the shortest paths between pairs of nodes in the network.

  The betweenness centrality B_i for a node i is:

$$B_i = \sum_{s \neq i \neq t} \frac{\sigma_{st}(i)}{\sigma_{st}} \tag{9}$$

  where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(i)$ is the number of those paths that pass through node i.

$$\text{Betweenness Centrality} = \frac{1}{N} \sum_{i=1}^{N} B_i \tag{10}$$

- **Heterogeneous Score**: here we define the heterogeneous score to examine the diversity of LLM used in the MAS. The higher score means that the LLM used in the MAS is more different.

$$\text{Heterogeneous Score} = \frac{\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \mathbf{1}(e_i \neq e_j)}{\binom{n}{2}} \tag{11}$$

  Where:

  - n is the total number of agents.
  - e_i represents the i-th agent's backbone LLM.
  - $\mathbf{1}(e_i \neq e_j)$ is an indicator function that equals 1 if $e_i \neq e_j$, and 0 otherwise.

4 Experiments

We experiment with the use of the indicators introduced in Section 3 to train a performance prediction model to predict the target scores. Specifically, we evaluate the MAS on three downstream tasks: HumanEval (Chen et al., 2021), MMLU (Hendrycks et al., 2020), and GSM8K (Cobbe et al., 2021), which test coding, reasoning, and math skills, respectively. We use a sampled version from MINT (Wang et al., 2023), where the queries are complex enough to require multi-agent collaboration. Furthermore, we configure each agent with different LLMs chosen from **Llama3-8B**, **Llama3-70B**, and **ChatGPT**, for details, see Appendix B.1. That is, we can perturb the selection of LLMs, generating a new combination that is a new data point [1] for training a regression model. By treating a different combination as a different data point, we can obtain 3^4 data points from this architecture in total if there are 4 agents in the system and 3 optional LLMs. In this paper, we collect a total of 1,796 data points. We then perform a grid search to train an XGBoost model (Chen and Guestrin, 2016), using *reg:squarederror* as the objective function.

Additionally, we experiment with the following settings:

- **Task-Group:** We group the tuples by task, then divide them into training and test sets.
- **Arch-Group:** We group the tuples by architecture, then divide them into training and test sets.
- **Random-Group:** We randomly divide all data into training and test sets.

4.1 Overall Results for Predicting Target Scores

The mean Spearman rank correlation with error bars is shown in Figure 3. The error bars are plotted using a 5-fold cross-validation. We observe a clear pattern: (1) The performance prediction model tends to achieve relatively high correlations and low variances. (2) Compared to Task-Group and Arch-Group, the Random-Group setting–which includes all tasks and architectures–achieves a higher overall mean score and a relatively lower variance. This indicates that access to information from other

[1] Here, each data point is a tuple of {various indicators, downstream task performance}.

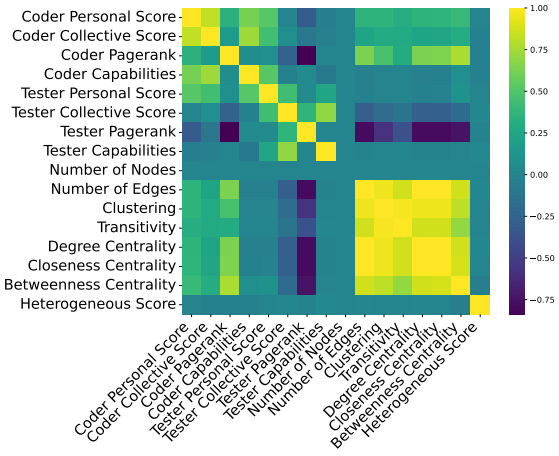


Figure 4: Feature Heatmap of Arch1 on MMLU.

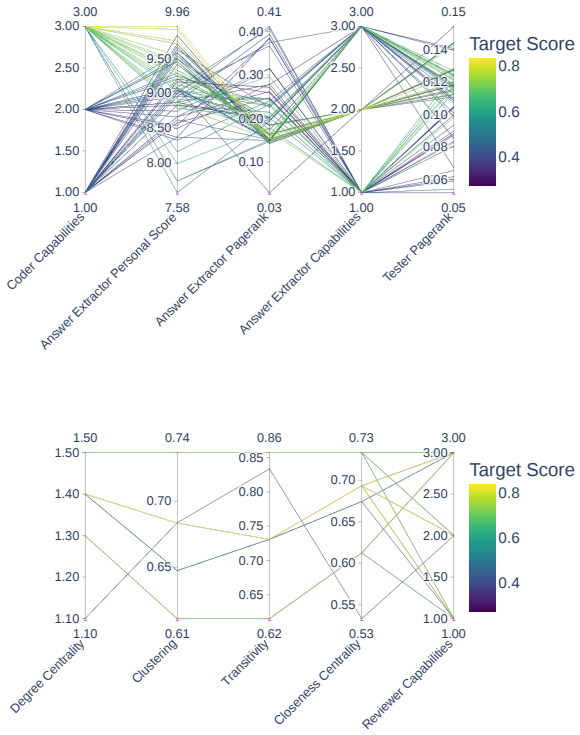


Figure 5: Parallel Coordinate Plot of Top5 (**Top**) and Bottom5 (**Bottom**) Important Features of Arch1 on GSM8K.

tasks or architectures improves predictive performance, and indicators are transferable between different settings.We also show the boxplot in Figure 8 which shows the distribution of errors in all settings. In general, the median lies close to zero, suggesting that the predicted values are generally close to the observed values. Furthermore, although the correlation for HumanEval in Task-Group setting is lower, the error is not, being smaller overall compared to the other two tasks. This suggests that while predicting the rank is more challenging for this task, the regression model can still predict values that are reasonably closer to the observed values.

4.2 A Closer Look into Relationships of Each Indicator

In this section, we take a closer look at the relationships between each indicator. As shown in Figure 4, the correlation heatmap of the indicators in MMLU for Arch1 is presented. We can identify some subtle patterns in this setting, where scores evaluated by the LLM, such as the coder personal score and the coder collective score, exhibit a higher correlation with each other but exhibit a lower correlation with other graph attributes, such as the total number of nodes and transitivity (indicated by the lighter color near the diagonal).

Furthermore, in Figure 5, we present the parallel coordinate plot of the features with the top five and bottom five importance scores of the features. The feature importance is calculated by XGBoost, which measures how much each indicator contributes to the model's inference. In particular, it is observable that most indicators do not exhibit monotonicity with respect to the target score, meaning that a higher indicator value does not necessarily result in a higher target score. Another important finding is that: Metrics that quantify individual agent capabilities emerged as the most influential predictors of overall system performance. Notably, other statistics with less predictive importance still demonstrated substantial variance, indicating that their diminished predictive influence is not due to limited variation. Instead, this suggests that effectively capturing core competencies of agents is paramount for accurately evaluating task performance, while other interaction metrics serve a secondary yet intricate role.

4.3 Estimated Statistics from Subset is Sufficient for Prediction

Another intuitive question is to what extent we need prior information to calculate the indicators described in Section 3. For example, given a prebuilt MAS and a downstream task, it is impractical to obtain all the required information only after completing all the instances in the test set. Ideally, we would only need a few instances to gather

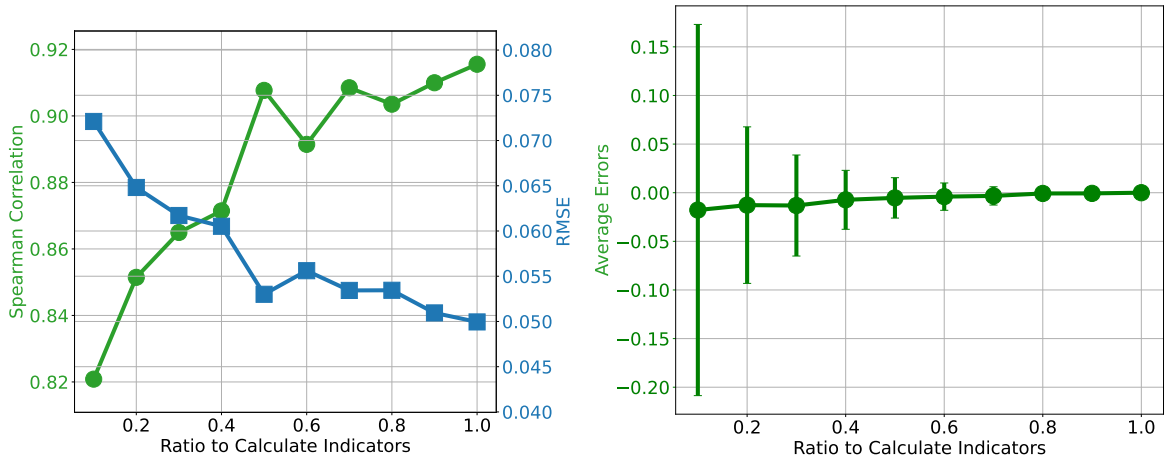


Figure 6: **Left**: Spearman Correlation and RMSE vs ratio to calculate indicators ; **Right**: Average Errors of Indicators vs ratio to calculate indicators.

enough information for a "sneak peek" at the system's potential performance. In contrast to the experiments detailed in the previous section, where the indicators were calculated by averaging over all instances, in this experiment tailored for *"RQ: How the number of instances that we use to calculate indicators affects the predictive results?"*, we use a subset of the total instances to calculate the "approximated indicators." We then analyze the effect of the number of instances used to calculate these indicators.

We begin by using the best-trained XGBoost model to perform inference on the Random-Group test set, retaining instances with an absolute error smaller than 0.05 as a new test set. The rationale behind this is that, these samples are more predictable for the trained model, and they better illustrate the usefulness of the indicators. Otherwise, the samples that are poorly predicted might not be explained by our model and could hinder the interpretation of the approximated indicators.

As shown in the left and right part of Figure 6, we observe the following: (1) There is a clear trend that as the ratio of instances used to calculate the indicators increases, the Spearman correlation continually rises, and the RMSE decreases. This suggests that increasing the number of instances used to calculate the indicators improves predictive performance. As expected, when the ratio increases, the predictive performance of the "approximated" indicators converge toward that of the "accurate" indicators. (2) Even when using only 10% of the total instances to calculate the indicators, the Spearman correlation is still around 0.82, supporting the claim that we can use a relatively small subset of data to gain an early glimpse of the final performance. This approach can guide the construction of MAS without fully executing the entire dataset. (3) The average error and variance decrease as the ratio increases. Additionally, we observe that when the ratio is low, the approximated indicator values tend to be smaller than the accurate values, suggesting that the main source of error may stem from certain indicators being underestimated by the LLM judger.

5 Conclusion

In conclusion, this paper confronts the significant challenge of performance predictability in the configuration of MAS, an area less explored than individual LLM predictability. We systematically investigated the question of *"How predictable MAS performance is?"*, by developing and evaluating a methodology rooted in monitoring system execution, capturing operational statistics from agent interactions, and utilizing these to train regression models for performance forecasting.

As a pioneering effort in systematically forecasting overall task performance in MAS, this research provides a foundational step towards moving beyond costly and time-consuming trial-and-error approaches to system configuration. The ability to anticipate system behavior, understand key performance drivers, and do so efficiently has profound implications for the design, development, and deployment of more reliable and multi-agent systems.

6 Limitations

Our empirical validation was based on five distinct, manually designed architectures and three specific benchmark tasks (HumanEval, MMLU, and GSM8K). Although these provide a solid foundation and cover diverse capabilities, the generalizability of our findings to the vast landscape of possible MAS configurations–including those with a significantly larger number of agents, different underlying LLMs powering the agents, or a broader array of real-world applications and collaborative paradigms–warrants further extensive investigation. Future work could address these limitations by leveraging more substantial computational resources to enable broader experimentation and validation.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, and 1 others. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2024. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv preprint arXiv:2407.07061*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Patricia Drapal, Ricardo BC Prudêncio, and Telmo M Silva Filho. 2024. Towards explainable evaluation: Explaining predicted performance using local performance regions. *Applied Soft Computing*, 167:112351.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ce Ge, Zhijian Ma, Daoyuan Chen, Yaliang Li, and Bolin Ding. 2024. Data mixing made efficient: A bivariate scaling law for language model pretraining. *arXiv preprint arXiv:2405.14908*.

Gemini, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, and 1 others. 2020. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, and 1 others. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Yunpeng Huang, Jingwei Xu, Zixu Jiang, Junyu Lai, Zenan Li, Yuan Yao, Taolue Chen, Lijuan Yang, Zhou Xin, and Xiaoxing Ma. 2023. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv preprint arXiv:2311.12351*.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. 2024. Scaling laws for downstream task performance of large language models. *arXiv preprint arXiv:2402.04177*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

LangChain-AI. 2024. Langchain. https://github.com/langchain-ai/langchain.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, and 1 others. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pages 22631–22648. PMLR.

Hongyin Luo, Tianhua Zhang, Yung-Sung Chuang, Yuan Gong, Yoon Kim, Xixin Wu, Helen Meng, and James Glass. 2023. Search augmented instruction learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3717–3729.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Brandon McKinzie, Zhe Gan, Jean-Philippe Fauconnier, Sam Dodge, Bowen Zhang, Philipp Dufter, Dhruti Shah, Xianzhi Du, Futang Peng, Floris Weers, and 1 others. 2024. Mm1: Methods, analysis & insights from multimodal llm pre-training. *arXiv preprint arXiv:2403.09611*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2024. Searchgpt prototype. https://openai.com/index/searchgpt-prototype/.

OpenInterpreter. 2023. Openinterpreter. Accessed: 2023-08-06.

Lorenzo Pacchiardi, Lucy G Cheke, and José Hernández-Orallo. 2024. 100 instances is all you need: predicting the success of a new llm on unseen data by testing on a few instances. *arXiv preprint arXiv:2409.03563*.

Lorenzo Pacchiardi, Konstantinos Voudouris, Ben Slater, Fernando Martínez-Plumed, José Hernández-Orallo, Lexin Zhou, and Wout Schellaert. 2025. Predictaboard: Benchmarking llm score predictability. *arXiv preprint arXiv:2502.14445*.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford infolab.

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

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, and 1 others. 2021. Scaling language

models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Reworkd. 2024. Agentgpt. https://github.com/reworkd/AgentGPT.

Wout Schellaert, Fernando Martínez-Plumed, and José Hernández-Orallo. 2025. Analysing the predictability of language model performance. *ACM Transactions on Intelligent Systems and Technology*, 16(2):1–26.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Significant Gravitas. 2024. Autogpt. https://github.com/Significant-Gravitas/AutoGPT.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. 2010. Evidence for a collective intelligence factor in the performance of human groups. *science*, 330(6004):686–688.

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

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

XAgent. 2023. Xagent: An autonomous agent for complex task solving.

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

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Yunhua Zhou, Jun Zhan, and Xipeng Qiu. 2024. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. *arXiv preprint arXiv:2403.16952*.

Qinyuan Ye, Harvey Yiyun Fu, Xiang Ren, and Robin Jia. 2023. How predictable are large language model capabilities? a case study on big-bench. *arXiv preprint arXiv:2305.14947*.

A MAS configuration

Table 1 shows the configuration of five MAS used in our paper, along with the used indicators.

B Details of Experiments Setting

In this section, we detail the LLMs used in various sections (Section B.1) and provide an introduction to the evaluation tasks and corresponding metrics (Section B.2).

B.1 LLM used in different sections

As shown in Table 1, we design various architectures for the MAS. However, if we use a system with only one LLM, the total number of runs remains low, which may not be sufficient to conduct the predictive experiment. An intuitive approach is to assign different LLMs to each agent in the system and permute them. For example, with an architecture of three agents and three different LLMs, we would have a total of 27 possible combinations (3^3).

In the experiment described in Section 4.1, we select LLMs from the following: GPT-3.5-turbo-1106, Llama3-8B-Instruct, and Llama3-70B-Instruct. These LLMs are chosen to represent varying levels of capability, thereby forming a diverse group of expertise, allowing us to construct MAS with greater diversity.

To reduce cost and improve throughput, we use the AWQ quantized version of Llama3-70B-Instruct from `https://huggingface.co/casperhansen/llama-3-70b-instruct-awq`. AWQ (Lin et al., 2024) is a training-free low-bit weight-only quantization method that does not rely on backpropagation or reconstruction, making it more efficient during inference. GPT-3.5-turbo and Llama3-8B-Instruct were obtained from their official providers, `https://platform.openai.com/docs/models/gpt-3-5-turbo` and `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`, respectively.

B.2 Evaluation Tasks Introduction

In this section, we introduce the tasks used in our paper. As shown in Table 2, the selected tasks for code generation, reasoning, and math follow MINT (Wang et al., 2023), where the sampled instances are more complex and require multi-turn interactions to solve.

C Training Details of Performance Prediction Model

We primarily utilize XGBoost (Chen and Guestrin, 2016) as the performance prediction model. XGBoost is a gradient boosting framework widely recognized for its efficiency and superior performance in regression and classification tasks. XGBoost optimizes the following squared-error objective function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{12}$$

where y_i represents the true value and $\hat{y}_i$ is the predicted value for instance i.

We conducted hyperparameter tuning using grid search combined with 5-fold cross-validation to identify the best-performing hyperparameters. The hyperparameters explored in our experiments include:

- Number of estimators: $\{50, 100, 200\}$
- Learning rate: $\{0.01, 0.1, 1.0\}$
- Maximum depth: $\{3, 5, 7\}$

The optimal hyperparameters were selected based on minimizing the mean squared error (MSE), evaluated as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{13}$$

D Additional Experimental Results

D.1 Average Errors Between the Predicted Score and the Observed Score

In addition to focusing on the relative ranking of the scores, performance prediction also emphasizes minimizing the actual error between predicted and observed scores. Figure 8 illustrates the average errors across different tasks. Notably, although the correlation for HumanEval in the Task-Group setting is lower compared to the other two tasks, the average error is smaller overall. This suggests that while predicting the rank is more challenging for this particular task, the regression model is still able to produce predictions that are reasonably close to the observed values. This highlights the model's ability to capture absolute trends effectively, even when relative rankings are harder to discern.

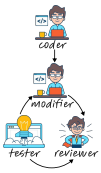
Architectures	Agents	Example Execution Graph	Indicators
Arch1	The coder is responsible for providing a self-contained code that can solve the task. The reviewer is responsible for writing review and provide comments. The tester is responsible for writing unit test to ensure the code functionality.		Indicators Judged by LLM Each Agent's Personal Score
Arch2	The coder is responsible for providing a self-contained code that can solve the task. The reviewer is responsible for writing review and providing comments. The dummy agent does not have specific responsibility but is acting as a black sheep. The tester is responsible for writing unit test to ensure the code functionality.		Each Agent's Collective Score Indicators Determined by Configuration Each Agent's Pagerank
Arch3	The coder is responsible for providing a self-contained code that can solve the task. The reviewer is responsible for writing review and providing comments. The web browser is responsible for using the Internet to retrieve useful information. The tester is responsible for writing unit test to ensure the code functionality.		Each Agent's Capabilities Number of Nodes Number of Edges Average Clustering
Arch4	The coder is responsible for providing a self-contained code that can solve the task. The modifier is responsible for modifying the code written by the coder. The reviewer is responsible for writing review and providing comments. The tester is responsible for writing unit test to ensure the code functionality.		Transitivity Average Degree Centrality Average Closeness Centrality Average Betweenness Centrality
Arch5	The executor is provided with an executable interpreter and should execute the code. The web browser is responsible for using the Internet to retrieve useful information.		Heterogeneous Score

Table 1: The configuration of five MAS used in our paper, along with the used indicators.

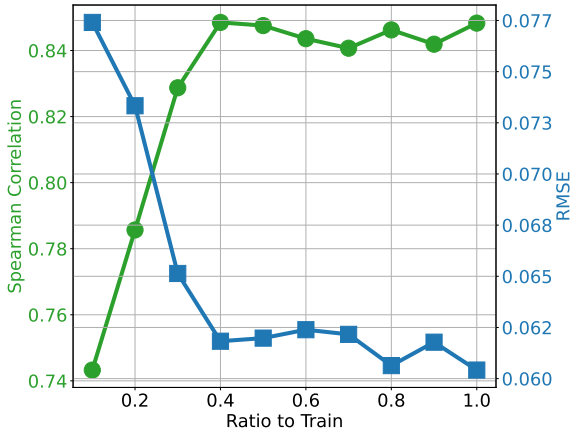


Figure 7: Spearman Correlation and RMSE vs ratio to train.

D.2 Training Size Ablation

In this section, we analyze how the training size affects the predictive results.

In this section we focus on the Random-Group setting and randomly sample 10%, 20%, 30%, $\cdots$, up to 100% of the original training set to form new training sets, while keeping the test set constant. As shown in Figure 7, as the training size increases, we observe a noticeable improvement in Spearman correlation and a reduction in RMSE, as expected. This indicates that more training data contribute to better predictive performance. Notably, when the training set reaches 50%, the results plateau, approaching those achieved with the entire dataset. This suggests that in this specific setting, half of the data contains sufficient information to achieve acceptable results.

Task Type	Task Name	Original Size	Sampled Size
Code Generation	HumanEval (Chen et al., 2021)	164	45
Reasoning	MMLU (Hendrycks et al., 2020)	13,985	48
Math	GSM8K (Cobbe et al., 2021)	1,319	48

Table 2: Used tasks in our paper.

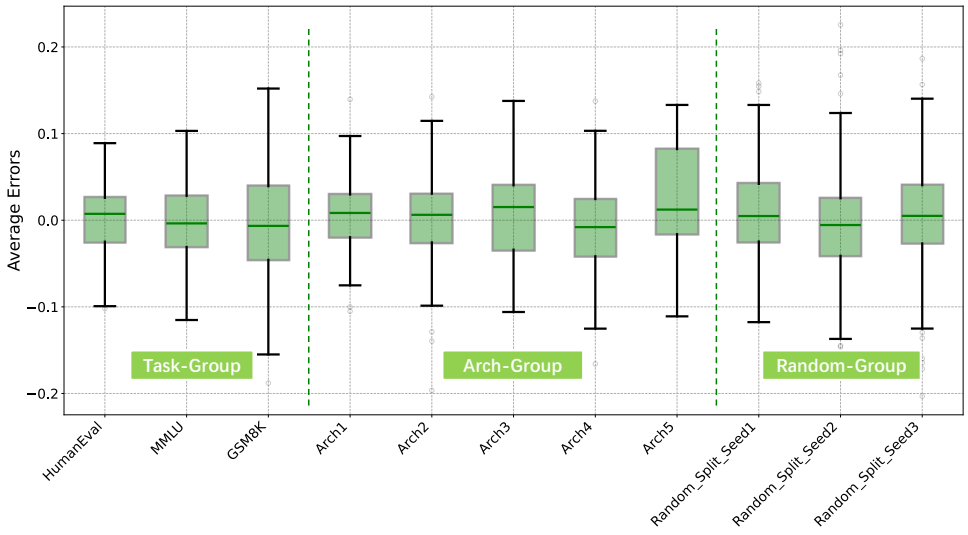


Figure 8: Errors between the predicted score and the observed score.

E Prompt Template Used in the Paper

E.1 Prompts of Agents Designed in Different Architectures

Table 3, 4, 5, 6, 7, 8, 9 and 10 present the prompts used in our designed multi-agent architecture, as outlined in Table 1. These agents include the Coder, Modifier, Reviewer, Tester, Dummy Agent, Executor, Web Browser, and Answer Extractor. Note that we do not show the Answer Extractor in the table, as it is utilized in all tasks requiring a final answer to be extracted from conversation history, except for HumanEval.

E.2 Prompts used to evaluate the personal score and collective score

Table 11 and 12 present the prompts used to evaluate each agent's personal score and collective score, respectively.

Finish the following python function as prompted:
{Instruction}
Below is the conversation history, you can use it as context to help you modify or maintain your original answer.
{Conversation History}
Please provide a self-contained python function that can solve the task and response it in a markdown code block.
For example:
Your code:
“‘Python
your code here
“‘
—
Your code:

Table 3: Coder Prompt Template.

{Instruction}
{Conversation History}
You are given the above instructions and conversation history. You are acting as an engineer to modify the code. Your peers have proposed the initial code and some have also reviewed and tested it. Please take this information into account and provide a refined and self-contained Python function that can solve the task. Please respond using a markdown Python code block.
For example:
Your code:
“‘Python
your code here
“‘
—
Your code:

Table 4: Modifier Prompt Template.

{Conversation History}
Review the test cases and provide critical comments:

Table 5: Reviewer Prompt Template.

{Conversation History}
Write k unit tests using pytest for the given function, assuming you have imported it. Return a python code in a markdown code block.

Table 6: Tester Prompt Template.

{Conversation History}
Above is a team’s conversation history;
Say some nonsense to disrupt the conversation:

Table 7: Dummy Agent Prompt Template.

Finish the following python function as prompted:
{Instruction}
Below is the conversation history, you can use it as context to help you modify or maintain your original answer.
{Conversation History}
Please provide a self-contained python function that can solve the task and response it in a markdown code block. And remember that your code will be actually executed, so make sure it is correct and safe.
For example:
Your code:
“‘Python
your code here
“‘
—
Your code:
After receiving the above code block, we then utilize a sandbox environment to execute the code, and return the results as follows;
Executed Code:
{Code Block}
Output:
{Interpreter Output}

Table 8: Executor Prompt Template.

{Instruction}
{Previous Search Results}
You are given the above instruction, and the corresponding histories of previous searched results. Please check whether it is expected and provide a more appropriate query for searching on the internet. Please directly output your refined query without any explanation.
Refined Query:
We first use the above template to prompt the llm for generate the query suitable for search engine.
—
{Instruction}
{Information}
You are given the instruction and also the relevant documents retrieved from the internet website, please give your suggestions towards solving the task.
Your suggestions:

Table 9: Web Browser Prompt Template.

{Conversation History}
Based on the upper information, provide an answer for the original task. If you are not sure, provide an answer anyway. Return your answer only , do not contain other irrelevant words.
Your Answer:

Table 10: Answer Extractor Prompt Template.

You are a judge tasked with evaluating the performance of the agent. Your evaluation should be based on the agent's adherence to its expected duties, as well as its behavior throughout the execution process of the entire MAS.
Evaluation Criteria for Agent Performance (0-10):
Score: 0-1
Agent failed to perform its expected duties. Significant errors or omissions in execution. Negatively impacted the overall system performance.
Score: 2-3
Agent performed some expected duties but with frequent errors. Multiple omissions or incorrect actions. Minor positive impact or neutral impact on the overall system performance.
Score: 4-5
Agent performed expected duties with occasional errors. Some omissions or incorrect actions, but they were not critical. Moderate positive impact on the overall system performance.
Score: 6-7
Agent performed expected duties with few errors. Minor omissions or incorrect actions that were not impactful. Significant positive impact on the overall system performance.
Score: 8-9
Agent performed expected duties with negligible errors. No critical omissions or incorrect actions. Very significant positive impact on the overall system performance.
Score: 10
Agent performed expected duties flawlessly. No errors, omissions, or incorrect actions. Exceptional positive impact on the overall system performance.
—
Carefully review the expected duties, execution process record and evaluation criteria. Based on your review, assign a score between 0 and 10 to measure how well the agent performed its job.
Now, focus on the behaviors and actions of the target agent: {Agent ID and Name}
{Expected Duties}
{Conversation History}
Please directly respond your score and do not followed by other text:

Table 11: Personal Score Judger Template.

You are a judge tasked with evaluating the contribution of the agent identified as {Agent ID and Name} to the final goal of the MAS. Your evaluation should be based on the agent's behavior throughout the execution process, as well as their impact on the overall system's success.
Evaluation Criteria for Agent Performance (0-10):
Score: 0-1
The agent not only failed to support the group goal effectively but also hindered the performance of other agents.
Score: 2-3
The agent contributed to the group in some scenarios, but overall contribution was limited and often flawed.
Score: 4-5
The agent contributed to the group in most cases but occasionally made mistakes.
Score: 6-7
The agent made significant contributions to the group, with only minor errors, and overall performed well.
Score: 8-9
The agent contributed greatly to the group, with almost no mistakes, and demonstrated excellent performance.
Score: 10
The agent made an outstanding contribution to the group and was a key driving force behind the system's success.
—
Consider whether the agent's actions were meaningful and directly supported the accomplishment of the primary objective, rather than just following instructions.
Now, focus on the behaviors and actions of the target agent: {Agent ID and Name}.
{System Goal}
{Conversational History}
Please directly respond your score and do not followed by other text:

Table 12: Collective Score Judger Template.