The Ted Lasso of Visual Analytics

Robert Specht [*] Qianwen Wang[†]

University of Minnesota
Computer Science and Statistics

Abstract

The Ted Lasso System (TLS) is a multi-agent system (MAS) inspired by the football coach, designed to guide agents in generating novel data visualizations for tabular data. These visualizations integrate techniques from machine learning, statistical learning, and data science. TLS employs a unique coaching strategy that combines specialized agent architectures, dynamically coded agents, advanced prompting methods, and retrieval-augmented generation (RAG) tools. Together, these components enable TLS to produce rich and effective data visualizations. The prompting strategies are grounded in key data visualization principles, ensuring both clarity and impact. While TLS consumes more tokens and requires longer runtime than single-agent systems, it generates more compelling, educational, and interactive HTML-based visualizations. These results consistently outperform those produced by single-agent systems.

1 Introduction

Multi-agent systems (MAS) have not consistently outperformed single-agent systems in discrete tasks [3], and single-agent systems and MAS continue to struggle with data science and visualization tasks [7]. This raises the question: can a coordinated system of agents be coached to generate meaningful data visualizations from tabular data?

2 Challenges and Solutions

Inspired by the success of MAS systems in drug rediscovery [4], spatial biology [12, 9], and neurology [8], it could be hypothesized that an MAS system could also uncover novel insights in data visualization and data science. The key lies in the system architecture, which includes specialized agents, dynamic edges, customized prompting strategies, and curated tool selection. But how can MAS systems be compared against one another? The simplest comparison method is to analyze their outputs. To facilitate faster LLM calls and easier output aggregation, we propose abstracting the output from HTML and Python code into ideas. However, quantifying"ideas" poses unique challenges. Thus, a two-step algorithm was implemented to structure and evaluate them.

The first step was aggregation. Each MAS system was run for 50 iterations, with the prompt to generate six data science ideas per iteration. Once the 50 iterations were completed, another LLM was used to perform a basic mapping algorithm that aggregated the ideas into buckets based on their descriptions. After aggregating the bucket counts, we required a method to compare MAS systems. The Pearson correlation coefficient (PC) was selected as the measurement. This coefficient quantifies the similarity of ideas between two MAS: a high PC indicates that the systems generated similar ideas, reflecting overlap in both breadth and depth. In contrast, our goal was to achieve a lower PC, encouraging the system to explore a wider range of educational ideas for the reader.

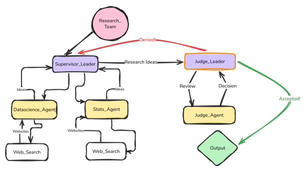

Figure 1: Research_Team

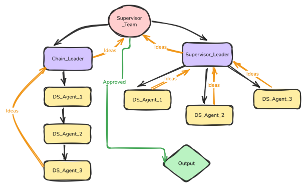

Figure 2: Supervisor_Team

The correlation was calculated by constructing vectors using the buckets defined during the aggregation step. If a bucket was absent from a given MAS system, a zero value was assigned to maintain consistent vector dimensions. The variance of each MAS vector was then calculated along with the covariance across all pairings of the system. Finally, the Pearson correlation coefficient was calculated for each pairing.

Together, aggregation and correlation allowed us to identify and compare the effectiveness of different MAS architectures. Multiple systems were then constructed to explore variations in novelty, breadth, and depth of idea generation. These architectural designs were inspired by proven techniques in prior MAS literature, including systems with supervisors [11], reflective agents [1], retrieval-augmented generation (RAG) tools [2], and chain-of-thought prompting [13].

- Idea 1:Chain — Chain up to n dependent agents. Each agent's prompt builds upon the output of the previous one.
- Idea 2:Supervisor — Deploy up to n independent agents, each of which can be called between 1 and N times.
- Idea 3:Judge — An agent accepts or rejects input. If rejected, the agent provides feedback.
- Idea 4:RAG — Agents are given access to web search or scraping tools to acquire external knowledge.

These four ideas, along with several combinations, were tested against one another. Previously successful MAS systems informed the combinations selected [4, 8, 12, 9], and given the following names:

- The Supervisor_Team (Figure 2) incorporates Ideas 1 and 2.
- The Research_Team (Figure 1) integrates Ideas 3 and 4.
- The MachineLearning_Team consists of a single agent using a richly detailed prompt grounded in modern learning theory and algorithms [5, 6].

Unless otherwise noted (e.g., labeled as "(n agents)"), Ideas 1 and 2 were implemented with three agents. The objective was to evaluate whether the additional token and runtime costs of these more complex MAS architectures translated into measurable improvements in novelty, depth, and breadth of ideas. The results of this evaluation, represented as Pearson correlation heatmaps, are shown in Figure 3.

The heatmap reveals several key insights: (1) The addition of a judge agent (Idea 3) has a marked impact, as evidenced by the difference between Idea 2 + 4 and Idea 2 alone. (2) More agents do not

[*]e-mail: spech065@umn.edu
[†]e-mail:qianwen@umn.edu

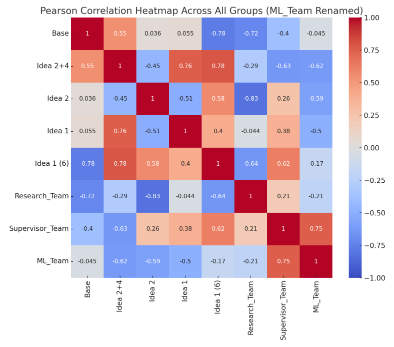

Figure 3: Pearson Correlation Coefficient Heatmap

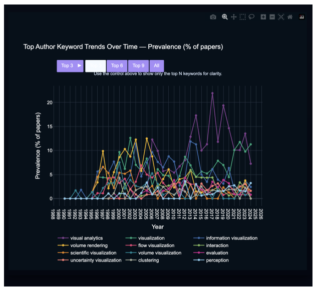
Top Author Keyword Trends Over Time — Prevalence (% of papers)
Prevalence (% of papers)
Year

Figure 4: HTML Plotly

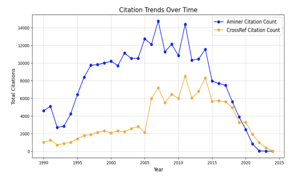

Figure 5: Python PDF

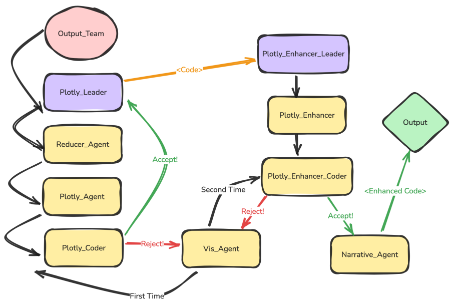

Figure 6: System architecture of `Output_Team`.

necessarily yield better results. Idea 1 with three agents performs comparably to Idea 1 with six agents. (3) Distinct architectures produce distinct outputs: the three teams (MachineLearning, Supervisor, and Research) generate notably different outputs, with some overlap observed between the ML_Team and Supervisor_Team. (4) The base model produces outputs that are substantially different from team-based systems. However, upon closer inspection of the individual ideas within the aggregated buckets, the base model's ideas (though varied) were relatively shallow and lacked the sophistication needed for educational visualizations. By contrast, the MAS teams produced outputs that were both more diverse and conceptually richer. For this reason, all three MAS teams were selected over the baseline as the foundation of TLS, ensuring breadth, depth, and novelty in the generated visualizations.

3 Output

The next challenge was to translate the output of the Ted Lasso System (TLS) into compelling visualizations. Three visualization strategies were tested: Plotly, PDF generation, and VegaLite. Generalized prompts were adapted for each rendering strategy.

The initial experiment employed a single-agent approach to convert ideas directly into the corresponding code format. However, this method proved unreliable, frequently producing bug-filled outputs. The solution involved implementing a dynamic feedback loop using a try-except block: the generated code was executed locally, and any runtime errors were captured and sent back to the LLM agent for refinement. A recursion limit of 25 iterations was imposed, which significantly improved the success rate of runnable code.

Despite this, recurring error patterns continued to trigger the recursion limit. Upon further investigation, the root cause was identified: the LLM lacked sufficient contextual understanding of the input data. To address this, a series of helper functions was introduced to extract low-token-cost metadata from the input CSV files, such as null values, data types, and basic dataset info. This lightweight data summary was then passed to the LLMs, enabling them to overcome frequent errors related to null values, one-hot encoding, singular feature dimensions, and more.

With these changes, the output became consistently bug-free for both the PDF and Python (Plotly) rendering pipelines. VegaLite, however, continued to produce unstable code and failed to remain within the recursion limit, leading to its removal from further experiments.

Another challenge was the excessive runtime caused by complex or redundant idea outputs. This was addressed by introducing the Reducer_Agent, which filtered out redundant, overly complex, or slow-to-execute ideas using a generalized prompt enhanced by modern prompting strategies [14].

The next challenge involved the simplicity of the generated data visualizations. The initial plots were rudimentary, lacking legends, meaningful color palettes, chart complexity, titles, and axis labels. To address this, an additional output layer was introduced. This layer followed the same structure as the prior coding layer, but instead of translating ideas into code, the new Enhancer Agent was designed to improve existing visualizations. It received a generalizable prompt informed by best practices in data visualization [10], resulting in markedly more detailed and immersive visual outputs.

At this stage, it became clear that Plotly was the optimal tool. As shown in Figure 4,5, Python produces static PDF images and plotly enabled user interactivity, zoom in and out, hover information, and dynamic layout adjustments, along with enhanced color and presentation quality. However, the use of Plotly alone lacked a cohesive narrative. To solve this, a Narrative Agent was introduced. It was given a generalized prompt to generate coherent, explanatory storytelling to accompany the visualization, leveraging advanced prompting techniques [14]. This consistently resulted in complete, educational, and engaging visual narratives.

Figure 4 illustrates the full Output_Team architecture. Coder Agents use try-except blocks to handle errors. Vis Agent refines and corrects any faulty Plotly or Python code. Plotly Agent transforms ideas into runnable Plotly visualizations. The ML_Team, Research_Team, and Supervisor_Team all feed their generated ideas into the Output_Team, which synthesizes them into a cohesive and interactive final HTML file.

4 Conclusion

Much like Ted Lasso, a successful MAS system is built by being a great coach. First, you need a balance of stars and team players. This means using different model types and companies, like mixing GPT-5 with GPT-4o-mini and Claude 3.5, to get a range of strengths. Second, a great coach understands the value of repetition. The same applies to MAS development. It takes repeated testing to find common pitfalls that can be worked through with in-depth domain knowledge applied to specific agents in the form of prompts, tools, and helper functions. Third, great coaches know how to communicate. They change their tone, phrasing, or approach until the message lands. In MAS systems, that's prompting; sometimes it takes adjusting the wording, syntax, or diction to fix persistent issues. Finally, a coach experiments with different team structures. Some players work well together, others don't, and the only way to find out is to try. MAS architects should do the same: mix and match agents, try unconventional pairings, and see what clicks. You never know. If you're curious, creative, and a little optimistic, you might be the next Ted Lasso.

Final Selected Output

https://www.visagent.org/api/output/743bb5d5-a3de-42b8-8ac0-97a6356f005f
https://www.visagent.org/api/submission/743bb5d5-a3de-42b8-8ac0-97a6356f005f

Figure Credits

Figure 3 were created with the assistance of **OpenAI** and Figure 1,2,6 were created with the open-source tool **Excalidraw** (https://excalidraw.com).

References

[1] X. Bo, Z. Zhang, Q. Dai, X. Feng, L. Wang, R. Li, X. Chen, and J.-R. Wen. Reflective multi-agent collaboration based on large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, eds., *Advances in Neural Information Processing Systems*, vol. 37, pp. 138595–138631. Curran Associates, Inc., 2024. 1

[2] D. A. Boiko, R. MacKnight, B. Kline, and et al. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023. doi: 10.1038/s41586-023-06792-0 1

[3] L. Chen, J. Q. Davis, B. Hanin, P. Bailis, I. Stoica, M. Zaharia, and J. Zou. Are more llm calls all you need? towards scaling laws of compound inference systems, 2024. 1

[4] J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, A. Palepu, P. Sirkovic, A. Myaskovsky, F. Weissenberger, K. Rong, R. Tanno, K. Saab, D. Popovici, J. Blum, F. Zhang, K. Chou, A. Hassidim, B. Gokturk, A. Vahdat, P. Kohli, Y. Matias, A. Carroll, K. Kulkarni, N. Tomasev, Y. Guan, V. Dhillon, E. D. Vaishnav, B. Lee, T. R. D. Costa, J. R. Penadés, G. Peltz, Y. Xu, A. Pawlosky, A. Karthikesalingam, and V. Natarajan. Towards an ai co-scientist, 2025. 1

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, NY, 2nd ed., 2009. doi: 10.1007/978-0-387-84858-7 1

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer, New York, NY, 2nd ed., 2021. doi: 10.1007/978-1-0716-1418-1 1

[7] L. Jing, Z. Huang, X. Wang, W. Yao, W. Yu, K. Ma, H. Zhang, X. Du, and D. Yu. Dsbench: How far are data science agents from becoming data science experts?, 2025. 1

[8] Z. Lin, A. Marin-Llobet, J. Baek, Y. He, J. Lee, W. Wang, X. Zhang, A. J. Lee, N. Liang, J. Du, J. Ding, N. Li, and J. Liu. Spike sorting ai agent. *bioRxiv*, 2025. doi: 10.1101/2025.02.11.637754 1

[9] Z. Lin, W. Wang, A. Marin-Llobet, Q. Li, S. D. Pollock, X. Sui, A. Aljovic, J. Lee, J. Baek, N. Liang, X. Zhang, C. K. Wang, J. Huang, M. Liu, Z. Gao, H. Sheng, J. Du, S. J. Lee, B. Wang, Y. He, J. Ding, X. Wang, J. R. Alvarez-Dominguez, and J. Liu. Spatial transcriptomics ai agent charts hpsc-pancreas maturation in vivo. *bioRxiv*, 2025. doi: 10.1101/2025.04.01.646731 1

[10] S. R. Midway. Principles of effective data visualization. *Patterns*, 1(9):100141, 2020. doi: 10.1016/j.patter.2020.100141 2

[11] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, 2024. doi: 10.1101/2024.11.11.623004 1

[12] H. Wang, Y. He, P. P. Coelho, M. Bucci, A. Nazir, B. Chen, L. Trinh, S. Zhang, K. Huang, V. Chandrasekar, D. C. Chung, M. Hao, A. C. Leote, Y. Lee, B. Li, T. Liu, J. Liu, R. Lopez, T. Lucas, M. Ma, N. Makarov, L. McGinnis, L. Peng, S. Ra, G. Scalia, A. Singh, L. Tao, M. Uehara, C. Wang, R. Wei, R. Copping, O. Rozenblatt-Rosen, J. Leskovec, and A. Regev. Spatialagent: An autonomous ai agent for spatial biology. *bioRxiv*, 2025. doi: 10.1101/2025.04.03.646459 1

[13] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. 1

[14] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. Association for Computing Machinery, New York, NY, USA, 2023. doi: 10.1145/3544548.3581388 2