Input Conditioned Graph Generation for Language Agents

Anonymous ACL submission

Abstract

Recent progress in the areas of Large Language Models (LLMs) and Language Agents has demonstrated significant promise for various future applications across multiple disciplines. While traditional approaches to language agents often rely on fixed, handcrafted designs, our research aims to develop agents that are both learnable and dynamic. In our method, we use an existing framework that abstracts language agents as graphs. Within this graph framework, we aim to learn a model that can generate edges for every given input to the language agent. This allows us to generate edges that represent the flow of communication within the graph based on the given input, thereby adjusting the internal communication of a language agent. We learn to generate these edges using a pretrained LLM that is fine-tuned with reinforcement learning. This LLM can be fine-tuned on several datasets simultaneously, and we hypothesize that the model learns to adapt to these different domains during training, achieving good overall performance when encountering data from different domains during deployment. We demonstrate that our approach surpasses the previous static approach by nearly 6% accuracy on a combined dataset of MMLU and CMMLU, and by more than 10% when trained with a sparsity-inducing loss. It also shows superior performance in additional experiments conducted with the MMLU and Mini Crossword Puzzles datasets.

1 Introduction

Recent advancements in Large Language Models (LLMs) have significantly expanded their potential applications. Pretrained LLMs can effectively handle a wide range of Natural Language Processing (NLP) tasks with little to no additional training, a capability known as zero-shot or few-shot learning (Brown et al., 2020; Touvron et al., 2023; Team et al., 2024). This enables their use in various critical applications, either to solve complex problems or to support human workflows (Chen et al., 2023; Colombo et al., 2024; Roziere et al., 2023; Xu et al., 2024).

A notable application of LLMs is the development of language agents. These agents use LLMs as their core component and perform various tasks through interactions among multiple LLMs, enhanced by additional operations such as memory retrieval, code execution, and environmental interactions like search (Birr et al., 2024; Hong et al., 2024; Chase, 2022). In contrast to usual LLMs, language agents interact with various environments by leveraging LLMs through actions and observations. Unlike LLMs, language agents can perform internal actions such as reasoning, which may involve multiple LLM queries before interacting with the environment. They also employ various tools to engage with data sources. Language agents can function in single-agent or multi-agent frameworks, using one or more LLMs (Wang et al., 2024). These agents can be trained using Reinforcement Learning (RL) techniques (Zhuge et al., 2024), although much of the existing literature focuses on handcrafted designs that utilize pretrained LLMs without further training (Hong et al., 2024; Chase, 2022; Chen et al., 2024).

1.1 Towards Learned and Dynamic Language Agents

Over the past few decades of deep learning research, a recurring pattern has been the superiority of learned features over handcrafted ones (Mikolov et al., 2013; Hinton et al., 2012; Silver et al., 2017). Notable examples include AlexNet by Krizhevsky et al. (2012), which significantly outperformed the state-of-the-art on ImageNet by employing Convolutional Neural Networks, and Neural Architecture Search algorithms, which improved performance across various benchmarks through architectures discovered via automated searches rather than handcrafted designs (Zoph and Le, 2016). While it remains a topic of debate whether learned approaches consistently surpass handcrafted designs, ample evidence suggests their potential for superiority (Krizhevsky et al., 2012; Mikolov et al., 2013; Hinton et al., 2012; Silver et al., 2017; Zoph and Le, 2016).

In the realm of language agents, many existing approaches incorporate handcrafted designs (Hong et al., 2024; Chase, 2022; Chen et al., 2024; Liu, 2022; Zhuge et al., 2023), often tailored explicitly for specific tasks. For instance, the MetaGPT framework assigns predefined roles to agents, mimicking human workflows (Hong et al., 2024). This strategy has shown promise, particularly in coding benchmarks, but it also introduces significant inductive biases by imposing human-like workflows on language agents, which may limit their potential by constraining their design.

Another important ability is adaptability to input variations, allowing systems to manage different types of data through distinct processing steps. Research on Chain-of-Thought (CoT) prompting has highlighted its advantages for mathematical or reasoning tasks, where several reasoning steps precede the final output (Wei et al., 2022). Similarly, the Tree of Thought approach has demonstrated improved performance in tasks such as crossword puzzles by exploring various potential answers (Yao et al., 2024). Based on this, we hypothesize that applying different strategies or workflows based on the given input can optimize task solutions. A one-size-fits-all solution may serve as a starting point, but over the long term, language agents should have the flexibility to explore various communication flows and apply tailored methods to enhance their performance.

Our work builds upon the framework proposed by Zhuge et al. (2024), which models language agents as Directed Acyclic Graphs (DAGs). This framework allows for an abstract understanding of language agents by representing them as computational graphs where nodes perform specific operations and edges depict the flow of data. We extend this DAG-based approach by introducing adaptive language agents that can modify their internal and external communications based on initial input. Using reinforcement learning, specifically the REINFORCE algorithm (Williams, 1992), we aim to optimize the communication flows within these agents. Unlike previous methods with fixed edge probabilities, our approach learns input-dependent edge probabilities by utilizing a LLM, allowing for dynamic and context-sensitive graph structures.

We aim to assess the performance of our method through three primary experiments using the Crosswords Puzzle dataset (Yao et al., 2024). These experiments evaluate the capability of language agents in solving 5x5 crossword puzzles, with performance measured by the number of correctly predicted words. The second experiment uses the Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2021) dataset for question answering to evaluate reasoning capabilities and detect adversarial agents within the graph. The final experiment combines the MMLU and Chinese Massive Multitask Language Understanding (CMMLU) (Li et al., 2023) datasets to test our method's ability to handle inputs from diverse domains, with performance measured by the number of correctly answered questions.

Our contributions are summarized as follows:

- We propose a novel method for edge optimization in language agents, enabling input-dependent graph generation.
- We provide theoretical justification and demonstrate the superiority of our method through experimental validation.

2 Related Work

Recent language models like GPT-3 (Brown et al., 2020), LLama (Touvron et al., 2023), and Claude[1] excel in diverse NLP tasks through unified architectures (Achiam et al., 2023; Jiang et al., 2023; Team et al., 2024; Touvron et al., 2023). Extensive pretraining allows for zero-shot and few-shot prompting without task-specific fine-tuning (Brown et al., 2020). Few-shot prompting uses example pairs for contextual learning, while zero-shot relies solely on task descriptions. Our research extends this by using language agents that interact with their environment. This approach enhances LLM functionality, enabling complex tasks like reasoning and memory retrieval by interacting with external tools and data sources (Liu, 2022; Chase, 2022; Birr et al., 2024; Reed et al., 2022).

2.1 Language Agents in Role Play Setting

In the existing literature, considerable attention has been devoted to exploring the potential of assigning specific roles to language agents to enhance their

[1] `https://docs.anthropic.com/claude/docs/models-overview`

problem-solving capabilities (Zhuge et al., 2023; Li et al., 2024; Hong et al., 2024; Qian et al., 2023). NLSOM by Zhuge et al. (2023) employs a society of mind concept, where multiple LLMs and neural network-based experts operate within a structured society, exchanging information to facilitate complex decisions. Agents in NLSOM follow predefined roles akin to political systems, such as democracies and monarchies, which, while structured, lack flexibility and require handcrafted organizational structures. Similarly, the CAMEL framework by Li et al. (2024) assigns specific roles to agents to guide problem-solving processes, emphasizing role adherence to enhance creativity. However, CAMEL's predefined role assignments and fixed communication schemes limit its adaptability. MetaGPT by Hong et al. (2024) focuses on role specialization and improved communication infrastructure, relying on user-defined roles and human workflow patterns adapted from software engineering. It uses a subscribe-and-publish mechanism for agent communication, offering some flexibility but still constrained by predefined workflows. In contrast, our research introduces a dynamic approach to inter-agent communication using reinforcement learning techniques, enabling agents to learn and adapt their communication strategies over time. This flexibility allows agents to modify their internal communication based on real-time task requirements and performance, leading to more robust and adaptable problem-solving capabilities, enhancing responsiveness to complex and evolving tasks.

2.2 Dynamic Language Agents

Although various methodologies have been developed for dynamically generating language agents tailored to specific task requirements (Team, 2023b; Chen et al., 2024; Liu et al., 2023; Yao et al., 2022), most focus on role assignment and task-dependent agent creation. In contrast, our approach utilizes a fixed set of agents and concentrates on optimizing their communication. XAgent by Team (2023b) is an open-source framework with a Dispatcher, Planner, and Actor, relying on LLMs for planning and dispatching tasks. Our method, however, integrates these functions within graph edge generation, using reinforcement learning for improved task-handling strategies. AgentVerse by Chen et al. (2024) employs a multi-stage problem-solving process with dynamic expert recruitment and structured decision-making, while our approach autonomously learns decision-making procedures using a utility function for feedback, allowing task transferability. DyLAN by Liu et al. (2023) uses inference-time agent selection based on an Agent Importance Score, restricted to multi-round interactions. Our method, instead, employs a generalized graph framework without limiting inter-agent connections, evaluated through a utility function on a dataset. Overall, our approach focuses on an abstracted graph framework that optimizes internal communication between agents using RL techniques, differing from methods that rely on LLMs' internal knowledge for decision-making.

3 Methodology

3.1 Language Agents as Graphs

Language agents, enhanced by pretrained LLMs, have shown significant promise in leveraging their extensive knowledge to handle various tasks. These agents often adopt complex behaviors, such as teamwork and role assignments, to improve task performance (Hong et al., 2024; Chen et al., 2024; Qian et al., 2023). The framework proposed by Zhuge et al. (2024) models language agents as Directed Acyclic Graphs (DAGs). In this approach, a language agent is defined as a graph $G(V, E, F, o)$, where V is the set of nodes, E is the set of edges between these nodes representing the flow of data within the language agent, F is a set of operations with f_i being the operation executed in node v_i, and o is the output node. In their research, they put special emphasis on language agent swarms. These swarms are graphs composed of several subgraphs, where every subgraph represents a single language agent. For their edge optimization techniques, they restricted the optimization to the edges between different agents within the composed graph. This restricted set of edges is $\mathcal{E} \subset E$.

3.2 Static Edge Probabilities

A language agent as a graph can be executed based on the topological order of the nodes within the graph. Every node takes as input the output generated by all its predecessor nodes and generates its own output.

In the method by Zhuge et al. (2024), edge selection within the graph is governed by a single parameter vector θ, where θ_i represents the probability of sampling edge e_i. This probability is optimized using RL techniques, specifically the REINFORCE algorithm. The objective is to find the optimal θ that maximizes the expected utility of the graph structures generated by this parameterization:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{G' \sim D_\theta}[u_{\mathcal{T}}(G')], \quad (1)$$

where $\theta^\star$ represents the optimal parameter vector that maximizes the expected utility. Here, Θ is the set of all possible parameter vectors, G' denotes a graph structure sampled from the distribution D_θ parametrized by θ, and $u_{\mathcal{T}}(G')$ is the utility function that evaluates the performance of the graph G'.

3.3 Input-Conditional Edge Probabilities

Given the increasing complexity of tasks that language agents must handle, there is a need for these agents to exhibit a high degree of adaptability. This adaptability involves dynamically adjusting their computational routines based on the specific input, akin to the flexibility seen in Mixture of Experts (MoE) architectures, where different "experts" are selected based on the input to optimize processing (Jacobs et al., 1991).

To enhance the adaptability and efficiency of language agents, we propose a novel method where edge probabilities are conditional on the input x. Instead of using a fixed parameter vector, we introduce a function f that maps an input x to a vector of probabilities θ, tailored for that specific input:

$$f(x) = \theta, \quad x \sim D, \quad (2)$$

where f is a function from the set of all possible functions $\mathcal{F}$. The input x is sampled from the distribution D, and θ is the vector of probabilities generated by f for the input x.

This approach allows the graph structure to dynamically adjust based on the input, improving the agent's ability to handle diverse tasks effectively. The optimization objective is then redefined to maximize the expected utility across different inputs and their corresponding graph structures:

$$f^\star = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{x \sim D}[\mathbb{E}_{G' \sim D_{f(x)}}[\hat{u}(G'(x))]], \quad (3)$$

where $f^\star$ is the optimal function that maps inputs to edge probabilities. Here, $\mathcal{F}$ is the set of all possible functions, x is an input sampled from the distribution D, $G' \sim D_{f(x)}$ represents a graph structure sampled from the distribution $D_{f(x)}$ parametrized by the output of $f(x)$, and $\hat{u}(G'(x))$ is a utility function that evaluates the output of the graph G' executed on input x. This method is designed to be at least as effective as the previous approach, which directly updates edge probabilities. This is because the function can always be learned to be a constant function. We provide a proof of this in Appendix A.

3.4 Design of Our Methodology

We employ the REINFORCE algorithm for training, providing a solid base for our initial experiments. Our goal is to learn the function f, which dynamically adjusts graph structure using a pretrained LLM.

Given the pivotal role of LLMs in processing textual input, f must comprehend and reason about text to accurately assess task requirements and devise optimal strategies.

We use a learnable linear transformation layer to map the last hidden dimension of the LLM's output to a probability vector $\theta \in \mathbb{R}^{|\mathcal{E}|}$ for edge selection in our graph-based model: $\theta = W \cdot h + b$, where $W \in \mathbb{R}^{|\mathcal{E}| \times d}$ is the weight matrix, $b \in \mathbb{R}^{|\mathcal{E}|}$ is the bias vector, and $h \in \mathbb{R}^d$ is the output from the last hidden layer of the LLM.

Following (Zhuge et al., 2024), we initialize the weights W to zero and set the bias b to reflect initial probabilities, guiding the initial learning phase.

However, initializing W with a normal distribution facilitates more effective gradient updates by breaking symmetry among neurons (LeCun et al., 2002; Glorot and Bengio, 2010): $W_{ij} \sim \mathcal{N}(0, \sigma^2)$. This diversity in initial weights allows neurons to learn different aspects of the input data, leading to more robust neural network models (Glorot and Bengio, 2010).

4 Experiments

We replicated the experiments from (Zhuge et al., 2024) to demonstrate that our method is at least as effective. Additionally, we introduced a new experimental framework to show that our method can significantly outperform the previous approach. In the following, we will refer to the previous method proposed by Zhuge et al. (2024) as Static Graph, and our method as Dynamic Graph. The code implementation we use is publicly available on GitHub, building upon the existing code base by Zhuge et al. (2024)[2].

[2]https://github.com/metauto-ai/GPTSwarm

Table 1: Accuracy results for $k = 5$ runs on the Crosswords Puzzle dataset by Dynamic Graph and Static Graph. We used LLama 3 8B instruction finetuned for LLM inference.

Method	Accuracy x_i for Run i (in %)					**Std (in %)**	**Mean Acc. (in %)**
	x_1	x_2	x_3	x_4	x_5		
Static Graph	23.5	18.0	21.5	19.5	17.5	2.2	20.0
Dynamic Graph	21.5	18.5	19.5	21.5	21.0	**1.2**	**20.4**

Table 2: Comparison of Dynamic Graph and Static Graph in terms of accuracy on the test sets. The expected number of edges refers to edges from the agents to the final decision node. We split the tasks into static tasks, which are reproduced from (Zhuge et al., 2024), and dynamic tasks, specifically designed to require language agents to change based on the input. The static tasks include accuracy results for the Crosswords Puzzle dataset and the Adversarial Agent experiment on the MMLU dataset, while the dynamic tasks include accuracy with additional loss to reduce computational costs on the combined MMLU and CMMLU dataset. All accuracies are reported in percentages. Ratio refers to how much more likely it is to sample an edge from a truthful agent compared to an adversarial agent.

Method	**Static Task**			**Dynamic Task**			
	Crosswords	Adversarial Agents		Specialized Agents		+Edge Reduction	
	Acc. (%)	**Acc. (%)**	**Ratio**	**Acc. (%)**	**Edges**	**Acc. (%)**	**Edges**
Static Graph	20.0	44.6	1.52	44.7	5.01	42.1	**2.60**
Dynamic Graph	**20.4**	**48.6**	**2.73**	**50.3**	**4.38**	**52.4**	2.72

4.1 Crosswords Puzzle Experiment

In our initial experiment, we aimed to replicate the crosswords puzzle experiments to demonstrate that our method performs better than the previous approach. We utilized the instruction-finetuned version of LLM known as LLama 3 8B (Meta AI, 2024), which has shown robust performance across various benchmarks despite its relatively smaller parameter size. This model was selected for its suitability for the crosswords puzzle task. For learning the edge probabilities (Eq. 3), we employed a Gemma-2B LLM (Team et al., 2024), balancing model size and language understanding capabilities. The prompts used for this experiment are provided in Appendix G.

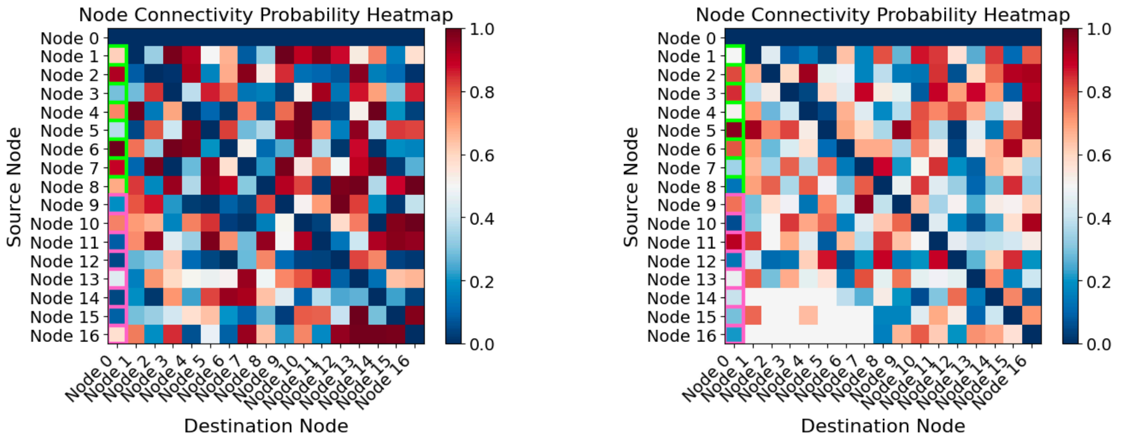
Following Zhuge et al. (2024), we explored two types of agents: the CoT agent (Wei et al., 2022) and the Reflexion Agent (Shinn et al., 2024). The **Chain of Thought** agent processes tasks in three logical steps: it initially analyzes the task in the first two steps and then outputs a solution based on the derived information in the final step. The **Reflexion Agent** initially solves the task with a greedy solution, receives feedback from an LLM, and then outputs a new solution based on the feedback. The outputs of these agents are collected by a **Return All** agent, which returns all solutions. There was a third agent, the **Tree of Thought** agent (Yao et al., 2024), but it was excluded due to the quadratic scaling of potential edges with the increase in nodes within a graph, in order to maintain computational feasibility.

We used a mini crosswords dataset of 156 5x5 puzzles from GooBix (Yao et al., 2024) and evaluated performance based on correct words for direct puzzle-solving assessment. Details of the dataset and language agent's graph structure, along with training parameters, are provided in Appendix C and B.1.

Results and Analysis: Due to the inherent randomness in the graph sampling process during evaluation, we tested our method on the test set five times. For the Static Graph approach we sampled five graphs. The results, detailed in Table 1, demonstrate that Dynamic Graph not only achieves higher average accuracy but also exhibits less variance across these runs, indicating enhanced performance and consistency from language agents using our method.

4.2 Adversarial Agent Detection

We replicated experiments from Zhuge et al. (2024) with the MMLU benchmark to validate



(a) Probabilities for sampling an edge in the graph learned by Dynamic Graph.

(b) Probabilities for sampling an edge in the graph learned by Static Graph.

Figure 1: Comparative visualization of edge probabilities in graphs learned by Dynamic Graph and Static Graph. Node 0 is the final decision node, nodes 1 to 8 are truthful agents, and nodes 9 to 16 are adversarial agents. Self-loops and connections from the final decision node to any other node are set to 0.

our method's ability to identify adversarial agents and compare it with Static Graph.

Experimental Setup: We used the instruction-finetuned Gemma-7B model, configuring seventeen agents: eight adversarial, eight truthful, and one final decision agent. Adversarial agents output lies, while truthful agents provide honest answers. The final decision agent performs majority voting on the inputs. Smaller models like Gemma-7B struggled with untruthful responses, frequently defaulting to truthful answers. To address this, adversarial agents were modified to consistently respond with "A".

The dataset used was the MMLU dataset (Hendrycks et al., 2021), which includes multiple-choice questions with four answer options, covering 57 topics. This dataset is a standard for assessing the world knowledge and problem-solving abilities of LLMs. Samples of this dataset are in Appendix D.

Following the original experiment's configuration, we conducted 200 iterations with a batch size of 4, a learning rate of 0.0001, and used the Adam optimizer. Both methods were trained on the MMLU dev set and tested on 1000 questions from the val set. Prompts used for this experiment are in Appendix F.

Results and Analysis

The Dynamic Graph effectively identified and excluded most adversarial agents. The average probability across the test set was calculated as

$$\bar{\theta} = \frac{1}{|D|} \sum_{x \in D} f(x),$$

where $\bar{\theta}$ is the average probability, D is the test set, and $f(x)$ is the function outputting edge probabilities.

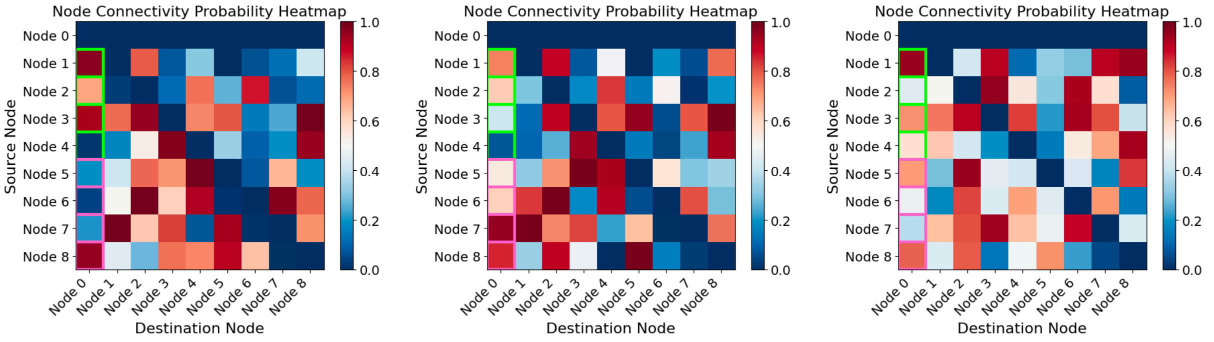
Figures 1a and 1b show heatmaps of edge probabilities in the graph. Our method demonstrated higher effectiveness in identifying adversarial agents, with probabilities for edges from truthful agents close to 1 and from adversarial agents near 0.

Critical edges are those from agents (nodes 1 to 16) to the final decision agent (node 0). The heatmaps illustrate that our method assigns high probabilities to edges from truthful agents and low probabilities to those from adversarial agents, enhancing robustness and accuracy. Outliers at nodes 5 or 10 had minimal impact due to the majority vote mechanism. This is evident from the ratio between the probability of sampling a critical edge from truthful agents compared to adversarial agents, which increased from 1.52 in the Static Graph to 2.73 in the Dynamic Graph.

Overall, the Dynamic Graph improves the detection and exclusion of adversarial agents, leading to higher accuracy, as shown in Table 2.

4.3 Specialized Agents Experiment

This experiment aimed to demonstrate the superiority of our method over the Static Graph approach. We trained a language agent on a diverse dataset requiring input-specific adaptation. Dynamic Graph adjusts to the specific characteristics of the input, unlike the Static Graph method, which aims for a generalized solution, often resulting in decreased



(a) Probabilities for sampling an edge on the MMLU dataset in the graph learned by Dynamic Graph.

(b) Probabilities for sampling an edge on the CMMLU dataset in the graph learned by Dynamic Graph.

(c) Probabilities for sampling an edge in the graph by Static Graph.

Figure 2: Comparative visualization of edge probabilities on MMLU and CMMLU datasets. Node 0 is the final decision node, nodes 1 to 4 are truthful agents using Gemma-7B-it, and nodes 5 to 8 are truthful agents using BlueLM-7B-chat. Notably, self-loops as well as connections from the final decision node to any other node are not allowed and thereby 0.

performance.

Experimental Setup: We configured eight truthful agents and a final decision agent, evenly divided between two language models: four based on Gemma-7B-it (Team et al., 2024) and four on BlueLM-7B-Chat (Team, 2023a). Despite their similar sizes, these LLMs show varying performance (Appenidx B.2) depending on the dataset used. Gemma-7B-it excels with the MMLU dataset (Hendrycks et al., 2021), while BlueLM-7B-Chat performs better on the CMMLU dataset (Li et al., 2023).

Results and Analysis: Our experiment demonstrated that Dynamic Graph effectively identified performance differences between the two LLMs and assigned higher probabilities to agents better suited for specific questions. Visualizations of adjacency matrices showed that the Static Graph found a more general solution, incorporating almost all agents with high probabilities (Figure 2c).

Heatmaps for the CMMLU (Figure 2b) and MMLU (Figure 2a) datasets displayed the average probability of sampling edges. Edges from Gemma-7B-it agents are highlighted in green, and those from BlueLM-7B-chat agents in pink. The model consistently assigned higher probabilities to the LLMs that performed better on these datasets, aiming for higher accuracy. Detailed edge probabilities are listed in Table B.2.

Our approach selected relevant agents based on input, reducing computational load by minimizing the number of agents needed for final decisions, thus saving resources. Table 2 shows that our method improved accuracy by nearly 6% over Static Graph on the test set.

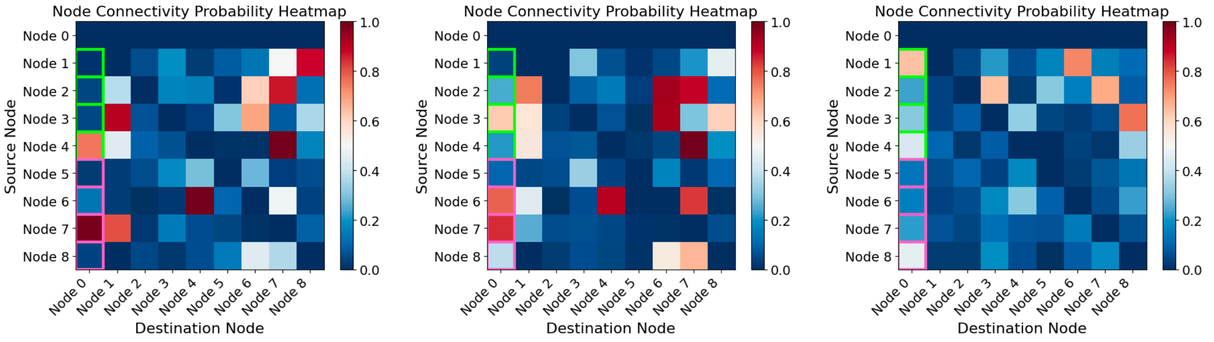
These results highlight the effectiveness of task-dependent graph construction for edge optimization. By dynamically adapting the graph and leveraging agent specifications, Dynamic Graph outperforms Static Graph, improving the performance of language agents on diverse tasks.

4.4 Edge Reduction on Specialized Agents

In this graph framework, reducing the number of edges can lower computational costs. We introduced an additional loss function during training to prioritize key nodes and reduce unnecessary internal communications. The loss function used was: $L(\theta) = \delta \cdot \sum_{i=1}^{|\mathcal{E}|} |\theta_i|$ with δ set to 0.1. This sparsity-inducing loss (Tibshirani, 1996) was applied using backpropagation to guide the model toward fewer edges, enhancing computational efficiency. The experimental setup mirrored Section 4.3, maintaining identical agent and training parameters.

Results and Analysis: Dynamic Graph showed enhanced performance over Static Graph. While the earlier method's performance declined, our approach improved, indicating that edge reduction helps identify relevant nodes and generate input-dependent graphs effectively.

Figure 3a shows edge probabilities with Static Graph, while Figures 3b and 3c show edge probabilities with Dynamic Graph on the MMLU and CMMLU datasets. Dynamic Graph adapts preferences based on input, especially noticeable in the CMMLU dataset. Table 2 shows our method outperformed the previous one by over 10%, identifying relevant agents and demonstrating robustness



(a) Edge probabilities on MMLU with Dynamic Graph and additional loss.

(b) Edge probabilities on CMMLU with Dynamic Graph and additional loss.

(c) Edge probabilities with Static Graph and additional loss.

Figure 3: Comparative visualization of edge probabilities learned by different methods on old and MMLU datasets. Node 0 is the final decision node, nodes 1 to 4 are Gemma-7B-it agents, and nodes 5 to 8 are BlueLM-7B-chat agents. Self-loops and connections from the final decision node to any other node are not allowed.

to changes in loss functions. It should be noted that Dynamic Graph sampled slightly more edges from truthful agents to the final decision agents.

4.5 Ablation Study

We investigated the role of the LLM in predicting edge probabilities. We compared our method, which uses transformer layers for textual representation, with a baseline model using a pretrained embedding matrix from Gemma 2B. The baseline averages embeddings and maps them to edge probabilities via a linear layer. This aligns with prior studies suggesting that averaging embeddings can capture essential textual information (Mikolov et al., 2013; Arora et al., 2017). The experiment setup was the same as in Section 4.3.

Results and Analysis: The baseline method struggled to differentiate based on input text, often defaulting to a constant vector output. Despite this, it performed better than earlier methods, achieving an accuracy of 45.7%. The heatmap of probabilities is in Appendix B.3.

These results highlight the importance of a component capable of comprehending language and text to generate task-dependent graphs accurately.

5 Conclusion

This work presents a new approach for edge optimization in graph frameworks for language agents, as introduced by Zhuge et al. (2024). Unlike Static Graph, which used a single vector of probabilities for edge sampling, Dynamic Graph learns a function f that dynamically maps the agent's input to edge probabilities. This generalizes the previous approach, which is a special case where f is constant. We train this function using a neural network built on a pretrained LLM.

Experimental results in Section 4 show Dynamic Graph consistently outperforms Static Graph across all tasks. Specifically, in a task-dependent graph construction experiment (Section 4.3), Dynamic Graph exceeded Static Graph by nearly 6% accuracy. This flexibility allows language agents to adjust their communication strategies based on input, processing input more effectively and enhancing their adaptability across various tasks.

6 Limitations

Our work establishes an experimental foundation demonstrating the potential benefits of dynamically adjusting language agents based on their input. However, further research is necessary to explore these concepts on a larger scale.

To deepen understanding, the effect of dynamic language agents should be investigated using agent swarms comprising a greater number of agents and employing larger LLMs. Additionally, while our research aimed to demonstrate the comparative efficacy of our method against that proposed by Zhuge et al. (2024), with superiority demonstrated through an additional experiment utilizing a mixed dataset, our experiments were confined to those conducted by Zhuge et al. (2024) employing edge optimization. While this comparison suffices to showcase the superior performance of our method in those specific experiments, future research should assess our method's capabilities across a broader spectrum of datasets and tasks, such as code generation.

Furthermore, exploring input-dependent edge generation could empower agents to dynamically adjust graph complexity based on input difficulty. For instance, a language agent might utilize a subgraph induced by nodes $V' \subseteq V$ to process an input x efficiently, thereby conserving computational resources.

Moreover, we propose investigating dynamic node generation, which would enable agents to generate, add, or remove nodes and edges during execution. This capability could enhance flexibility and empower graphs to effectively handle diverse inputs.

7 Potential Risks

Language Agents can substantially extend the capabilities of LLMs, allowing them to interact with their environment through a multitude of ways. Consequently, there is a concern that these advancements could lead to widespread automation, potentially displacing human labor on a large scale (Brynjolfsson and McAfee, 2015). Moreover, there is a risk that these language agents could be exploited for illegal or dangerous activities (Brundage et al., 2018).

We acknowledge the potential for our research, contributing to this field, to have harmful effects on society. Therefore, we advocate for more work on AI safety measures and the controlled deployment of AI technologies (Amodei et al., 2016).

References

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

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations*.

Timo Birr, Christoph Pohl, Abdelrahman Younes, and Tamim Asfour. 2024. Autogpt+ p: Affordance-based task planning with large language models. *arXiv preprint arXiv:2402.10778*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, Hyrum Anderson, Heather Roff, Gregory C. Allen, Jacob Steinhardt, Carrick Flynn, Seán Ó hÉigeartaigh, Simon Beard, Haydn Belfield, Sebastian Farquhar, Clare Lyle, Rebecca Crootof, Owain Evans, Michael Page, Joanna Bryson, Roman Yampolskiy, and Dario Amodei. 2018. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *Preprint*, arXiv:1802.07228.

Erik Brynjolfsson and Andrew McAfee. 2015. Will humans go the way of horses. *Foreign Aff.*, 94:8.

Harrison Chase. 2022. LangChain.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ArXiv preprint arXiv:2309.10848.

Zeming Chen, Alejandro Hernández Cano, Angelika Romanou, Antoine Bonnet, Kyle Matoba, Francesco Salvi, Matteo Pagliardini, Simin Fan, Andreas Köpf, Amirkeivan Mohtashami, et al. 2023. Meditron-70b: Scaling medical pretraining for large language models. *arXiv preprint arXiv:2311.16079*.

Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre FT Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, et al. 2024. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2024.

Metagpt: Meta programming for multi-agent collaborative framework. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ArXiv preprint arXiv:2308.00352.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2002. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2024. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2023. Cmmlu: Measuring massive multitask language understanding in chinese. *Preprint*, arXiv:2306.09212.

Jerry Liu. 2022. LlamaIndex.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

Meta AI. 2024. Introducing meta llama 3: The most capable openly available LLM to date.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

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*.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175*.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

BlueLM Team. 2023a. Bluelm: An open multilingual 7b language model. `https://github.com/vivo-ai-lab/BlueLM`.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

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

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26.

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

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Haoran Xu, Young Jin Kim, Amr Sharaf, and Hany Hassan Awadalla. 2024. A paradigm shift in machine translation: Boosting translation performance of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ArXiv preprint arXiv:2309.11674.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. 2023. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jurgen Schmidhuber. 2024. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

A Theoretical Justification of Our Method

Our approach is naturally designed to be as effective as or better than the previous approach, particularly when learning a constant function defined as:

$$f(x) = \theta^\star \in \mathbb{R}^{|\mathcal{E}|} \tag{4}$$

where $\theta^\star$ is derived from Equation (1), ensuring that the performance matches or exceeds that of the existing methodology.

The original optimization goal set forth by Zhuge et al. (2024) is:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{G' \sim D_\theta}[u_{\mathcal{T}}(G')] \tag{5}$$

Considering the utility function as the average utility over the current batch, we can rewrite this as:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{G' \sim D_\theta}\left[\frac{1}{B}\sum_{i=1}^{B} \hat{u}(G'(x))\right] \tag{6}$$

Assuming a sufficiently large batch size, this average utility represents an unbiased estimator of the expected utility, allowing us to reframe it as:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{G' \sim D_\theta}[\mathbb{E}_{x \sim D}[\hat{u}(G'(x))]] \tag{7}$$

Using the commutativity of expected values, it simplifies to:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{x \sim D}[\mathbb{E}_{G' \sim D_\theta}[\hat{u}(G'(x))]] \tag{8}$$

Defining $\theta^\star$ as the solution to this optimization.

By constraining the function set $\mathcal{F}$ to constant functions, $\mathcal{F}_c$, we align our new optimization goal with the original objective:

$$\begin{aligned} &\arg\max_{f \in \mathcal{F}_c} \mathbb{E}_{x \sim D}[\mathbb{E}_{G' \sim D_{f(x)}}[\hat{u}(G'(x))]] \\ &= \arg\max_{\theta \in \Theta} \mathbb{E}_{x \sim D}[\mathbb{E}_{G' \sim D_\theta}[\hat{u}(G'(x))]] \\ &= \theta^\star \end{aligned} \tag{9}$$

Since the set of constant functions $\mathcal{F}_c \subseteq \mathcal{F}$, our solution is at least as good as the solution found by the method introduced by Zhuge et al. (2024).

B Experiments

B.1 Crosswords Puzzle Experiment

We provide an example of the language agent graph of the Crosswords Puzzle experiment in Figure 4.

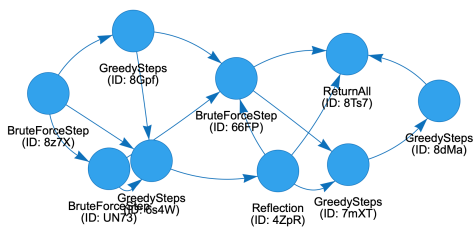


Figure 4: This is a sample graph for the crosswords experiment setup.

We used a mini crosswords dataset comprising 156 5x5 crossword puzzles collected from GooBix[3] as described in (Yao et al., 2024). Performance on this dataset was evaluated using three primary metrics: correct letters, correct words, and correct games. Consistent with previous studies, our evaluation focused on the number of correct words, allowing for a direct assessment of puzzle-solving effectiveness based on the provided clues. Samples of this dataset are available in Appendix C.

We used the same subset of 20 crossword puzzles for training and testing as (Zhuge et al., 2024; Yao et al., 2024). Our approach, with a greater number of learnable parameters and a reduced learning rate, required extended training iterations. We increased the iteration count from 10 to 40 and decreased the batch size from 20 to 5, ensuring a total of 200 examples were presented during training. The

[3] https://www.goobix.com/crosswords/

initial edge-sampling probability was set at 0.1, with a learning rate of 0.0001 using the Adam optimizer. We reported the best state word accuracy (Yao et al., 2024; Zhuge et al., 2024), indicating the accuracy of the best-proposed solution.

B.2 Specialized Agents Experiment

Training Details: We used a combined dataset from MMLU and CMMLU. Each dataset contains multiple-choice questions, testing agents on a wide range of domains and languages. The CMMLU benchmark, with questions in Mandarin Chinese, assesses LLMs across 67 topics and includes linguistic and culturally specific content (Appendix E).

Training was conducted over 200 iterations with a batch size of 4 and a learning rate of 0.0001 using the Adam optimizer. We trained on the dev sets of MMLU and CMMLU, and tested on 1000 questions from the MMLU validation set and the CMMLU test set.

We provide the performance of both LLMs used for Section 4.3 on the MMLU and CMMLU benchmarks. Since we couldn't find a reported value for the performance of Gemma on the CMMLU dataset, we evaluated the model ourselves in Table 3.

Table 3: Comparison of LLM performance on the MMLU and CMMLU benchmarks.

Model Name	Accuracy (%)	
	CMMLU	MMLU
BlueLM-7B-Chat	**72.7**	50.7
Gemma-7B	37.0	**64.3**

Results: We provide a detailed breakdown of the relevant edge probabilities from agents to the final decision node in Table 4. The table shows that our method was able to detect the differences in the LLMs' abilities and adjust their contribution to the final result based on the input's origin dataset.

B.3 Ablation Study

In this appendix, we provide the heatmap depicting the edge probabilities for the ablation study described in 4.5. The heatmap visualizes the probabilities assigned to different edges in the graph based on the input text.

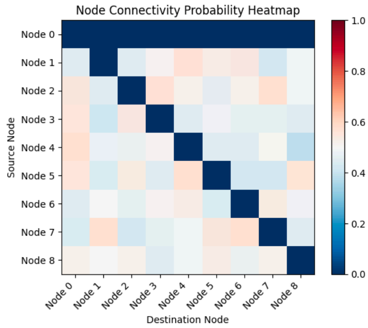


Figure 5: Probabilities for sampling an edge in the graph by Dynamic Graph with the reduced model. Node 0 is the final decision node, nodes 1 to 4 are truthful agents using Gemma-7B-it, and nodes 5 to 8 are truthful agents using BlueLM-7B-chat. Notably, self-loops as well as connections from the final decision node to any other node are not allowed and thereby 0.

C Mini Crosswords Puzzle Dataset

We provide samples (Table 5) of the Mini Crosswords Puzzle dataset used for the experiments in 4.1.

Below is the raw input formatted for clarity, ↪ indicates a line break:

Input Data

```
[
    [
        "To stamp; to brand; to impress; to
        ↪  put into type",
        "A scarf; a cymar; a loose dress",
        "To cut",
       "To perceive; wisdom; reason; feeling",
        "The ridges on a tire; to walk
        ↪  heavily",
        "A signaling sound",
        "A rice processor; an implement for
        ↪  ricing potatoes",
        "A chemical compound",
        "A dog whelk or its shell",
        "Chased up a tree"
    ],
    [
        "P", "R", "I", "N", "T",
        "S", "I", "M", "A", "R",
        "S", "C", "I", "S", "E",
        "S", "E", "N", "S", "E",
        "T", "R", "E", "A", "D"
    ]
]
```

D MMLU Dataset

The following question is a data sample from the MMLU data set, used in the experiments in 4.2 and 4.3. Specifically sampled from the test set category **College Mathematics**.

Table 4: Comparison of Dynamic Graph with Static Graph method. We report the probabilities for sampling edges from the agents to the final decision node. For our method we further report the average probability over the test set of both the MMLU and CMMLU datasets.

Node	Dynamic Graph			Static Graph
	CMMLU	MMLU	Difference	
Gemma-7B-It				
1	0.792	0.974	-0.182 ↓	0.942
2	0.629	0.716	-0.087 ↓	0.456
3	0.460	0.936	-0.476 ↓	0.717
4	0.059	0.023	0.036 ↑	0.567
BlueLM-7B-Chat				
5	0.519	0.211	0.308 ↑	0.701
6	0.556	0.063	0.493 ↑	0.471
7	0.938	0.075	0.863 ↑	0.374
8	0.852	0.956	-0.104 ↓	0.782

Table 5: Sample Data from the Mini Crosswords Dataset

P	R	I	N	T
S	I	M	A	R
S	C	I	S	E
S	E	N	S	E
T	R	E	A	D

Clue	ID
To stamp; to brand; to impress; to put into type	H1
A scarf; a cymar; a loose dress	H2
To cut	H3
To perceive; wisdom; reason; feeling	H4
The ridges on a tire; to walk heavily	H5
A signaling sound	V1
A rice processor; an implement for ricing potatoes	V2
A chemical compound	V3
A dog whelk or its shell	V4
Chased up a tree	V5

Dataset Question

```
Question: Let V and W be 4-dimensional
↪  subspaces of a 7-dimensional vector space
↪  X.
Which of the following CANNOT be the dimension
↪  of the subspace V intersect W?
Options:
A) 0
B) 1
C) 2
D) 3
Correct Answer: A
```

Below is the raw input formatted for clarity, ↪ indicates a line break:

Dataset Question

```
Let V and W be 4-dimensional subspaces of a
↪  7-dimensional vector space X. Which of the
↪  following CANNOT be the dimension of the
↪  subspace V intersect W?,0,1,2,3,A
```

E CMMLU Dataset

The following question is a data sample from the CMMLU data set, used in the experiments in 4.3. Specifically sampled from the test set category **Chinese Food Culture**.

Dataset Question

```
Question: 传统名菜“松鼠桂鱼”是典型的什么菜?
Options:
A) 川菜
B) 粤菜
C) 淮扬菜
D) 鲁菜
Correct Answer: C
```

Below is the raw input formatted for clarity, ↪ indicates a line break:

Input Data

```
传统名菜“松鼠桂鱼”是典型的什么菜? ,川菜,粤
↪  菜,淮扬菜,鲁菜,C
```

In this appendix we provide samples for the prompts used in our experiments.

F MMLU Prompt Set

This appendix provides detailed examples of prompts used in the study. These prompts are part of the `MMLUPromptSet` for a 4-option question answering framework. Except the constraint prompt, all prompts are adapted from (Zhuge et al., 2024).

F.1 Role of the Prompt

The role prompt is defined as follows:

```
    a knowledgeable expert in question
↪   answering
```

F.2 Constraints of the Prompt

The constraints are outlined to ensure the response format and correctness. We changed the required output format to JSON format. This made parsing easier and helped the models to adhere better to a given format:

```
"I will ask you a question.
I will also give you 4 answers enumerated as
↪   A, B, C, and D.
Only one answer out of the offered 4 is
↪   correct.
You must choose the correct answer to the
↪   question.
Answer with only a single letter (A, B, C, or
↪   D).
Do not include any other information in your
↪   answer except the letter.
Your response should be in JSON format, with
↪   the key 'answer' and the value being one
↪   of the 4 letters: A, B, C, or D,
↪   corresponding to the correct answer.
Here is an example of the correct format:
{
    'answer': 'A'
}"
```

F.3 Formatting of the Response

The expected format of the response is:

one of the letters: A, B, C or D

In JSON format.

F.4 Example Prompts

Here are some specific prompts used for different scenarios within the framework:

F.5 Adversarial Answer Prompt

Designed to receive a deceptive answer to the given question. We did not use this prompt for our experiments because the LLMs seemed to be unable to output a lie:

```
Answer a lie to the following question:
↪ {question}.
```

F.6 Reflective Prompt

Encourages reflection on the provided question and answer, assessing correctness and accuracy:

```
Reflect on the following question and answer:
Question: {question}
Answer: {answer}
What are your thoughts on the correctness and
↪   accuracy of the answer? Do you agree or
↪   disagree? Why? Please provide a brief
↪   explanation.
```

G Crosswords Prompt Set

This appendix provides detailed examples of prompts used in the mini crossword puzzle game framework. These prompts are part of the `CrosswordsPromptSet`, designed to guide LLMs in solving a 5 x 5 crossword puzzle. The prompts are mainly adapted from the framework and experiments by Zhuge et al. (2024).

G.1 Propose Prompt

The prompt used for to start the game is the following:

```
Let's play a 5 x 5 mini crossword, where each
↪   word should have exactly 5 letters.

{board}

Given the current status, list all possible
↪   answers for unfilled or changed words, and
↪   your confidence levels
↪   (certain/high/medium/low), using the JSON
↪   format with the position of the word as
↪   key and a list of lists consisting of
↪   possible answer and your confidence about
↪   the solution, as shown in this example
↪   {{"<position>": [["<answer>" ,
↪   "<confidence>"]]}}. Use "certain"
↪   cautiously and only when you are 100\%
↪   sure this is the correct word. You can
↪   list more than one possible answer for
↪   each word. Each word should have a length
↪   of exactly 5 characters. Consider the
↪   intersection of horizontal and vertical
↪   words.
```

G.2 Prompt to Test Correctness

The prompt used to verify the correctness of an answer by an LLM:

```
"Does {word} has meaning "{meaning}"? Responde
↪   only Yes or No."
```

G.3 Formatting of the Response

The expected format of the response is:

> a JSON object containing the position, possible answers, and their confidence levels

G.4 Suggest Prompt

A prompt to inform an LLM about a previous game and ask it to plan the next game:

```
    You are playing a 5 x 5 mini crossword, where each word should have exactly 5 letters.
Given the current status: The target words are classified as {List of words and their categorization into "correct", "incorrect", and "impossible"} You will retry the game. Write a plan for the next time.
Respond at most five sentences, one sentence per line.
Do not include the phrase "next time" in your response.
```

G.5 Evaluation Prompt

A prompt to make the LLM evaluate the current state of the board and find possible solutions based on letters that have already been filled in.:

```
    Evaluate if there exists a five letter word of some meaning that fit some letter constraints (sure/maybe/impossible).

Incorrect; to injure: w _ o _ g
The letter constraint is: 5 letters, letter 1 is w, letter 3 is o, letter 5 is g.
Some possible words that mean "Incorrect; to injure":
wrong (w r o n g): 5 letters, letter 1 is w, letter 3 is o, letter 5 is g. fit!
sure

A person with an all-consuming enthusiasm, such as for computers or anime: _ _ _ _ u
The letter constraint is: 5 letters, letter 5 is u.
Some possible words that mean "A person with an all-consuming enthusiasm, such as for computers or anime":
geek (g e e k): 4 letters, not 5
otaku (o t a k u): 5 letters, letter 5 is u
sure

Dewy; roscid: r _ _ _ l
The letter constraint is: 5 letters, letter 1 is r, letter 5 is l.
Some possible words that mean "Dewy; roscid":
moist (m o i s t): 5 letters, letter 1 is m, not r
humid (h u m i d): 5 letters, letter 1 is h, not r
I cannot think of any words now. Only 2 letters are constrained, it is still likely
maybe

A woodland: _ l _ d e
The letter constraint is: 5 letters, letter 2 is l, letter 4 is d, letter 5 is e.
Some possible words that mean "A woodland":
forest (f o r e s t): 6 letters, not 5
woods (w o o d s): 5 letters, letter 2 is o, not l
grove (g r o v e): 5 letters, letter 2 is r, not l
I cannot think of any words now. 3 letters are constrained, and _ l _ d e seems a common pattern
maybe

An inn: _ d _ w f
The letter constraint is: 5 letters, letter 2 is d, letter 4 is w, letter 5 is f.
Some possible words that mean "An inn":
hotel (h o t e l): 5 letters, letter 2 is o, not d
lodge (l o d g e): 5 letters, letter 2 is o, not d
I cannot think of any words now. 3 letters are constrained, and it is extremely unlikely to have a word with pattern _ d _ w f to mean "An inn"
impossible

Chance; a parasitic worm; a fish: w r a k _
The letter constraint is: 5 letters, letter 1 is w, letter 2 is r, letter 3 is a, letter 4 is k.
Some possible words that mean "Chance; a parasitic worm; a fish":
fluke (f l u k e): 5 letters, letter 1 is f, not w
I cannot think of any words now. 4 letters are constrained, and it is extremely unlikely to have a word with pattern w r a k _ to mean "Chance; a parasitic worm; a fish"
impossible

{input}
```

H CMMLU and MMLU Prompt Set

Direct translations of the prompts into Chinese were considered; however, such translations did not influence the performance outcomes of the models. Therefore, for simplicity and to streamline the implementation process, identical prompts were employed during training on both the MMLU and CMMLU datasets.