JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 5, pp. 55-66, May 2026
https://doi.org/10.9708/jksci.2026.31.05.055

Window-Augmentation Based Methodology for Improving Error Attribution Performance in Multi-Agent Systems

Seungkeon Lee*, Namgyu Kim**

*Graduate Student, Graduate School of Business IT, Kookmin University, Seoul, Korea
**Professor, Graduate School of Business IT, Kookmin University, Seoul, Korea

[Abstract]

Recent LLM Multi-Agent Systems have demonstrated strong performance across various domains; however, they are limited in that errors can propagate across steps due to their interaction structure. Accordingly, the importance of failure attribution, which aims to identify the root cause of task failure, has been increasingly emphasized. To address the limitation of existing methods that do not sufficiently capture interactions between adjacent steps, this study proposes a window-augmentation-based methodology. The proposed approach constructs a localized context around error steps identified by existing methods and re-identifies the decisive error step and responsible agent. Experimental results show that the proposed method improves both step and agent accuracy over existing approaches. Notably, the best performance is achieved with a window size of 3, highlighting the importance of proper window size selection. These findings demonstrate that localized context-based analysis is effective for failure attribution in multi-agent systems and can be extended to diverse environments.

▸ **Key words**: LLM Multi-Agent Systems, Failure Attribution, Window Augmentation, Local Context

[요 약]

최근 LLM 멀티에이전트 시스템은 다양한 분야에서 활용되며 높은 성능을 보이고 있으나, 상호작용 구조로 인해 오류가 연쇄적으로 전파되는 한계를 갖는다. 이에 따라 작업 실패의 근본 원인을 식별하는 오류 원인 귀속의 중요성이 강조되고 있다. 본 연구에서는 기존 오류 원인 귀속 방법론이 인접 단계 간 상호작용을 충분히 반영하지 못하는 한계를 극복하기 위해, 윈도우 증강(window augmentation) 방법론을 제안한다. 제안 방법론은 기존 방법론에서 식별한 오류를 중심으로 윈도우 증강을 적용한 국소적 맥락을 기반으로 결정적 오류 단계 및 에이전트를 재식별한다. 실험 결과, 제안 방법론은 기존 방법론 대비 단계 정확도와 에이전트 정확도 모두 향상된 성능을 보였으며, 특히 윈도우 크기 3에서 가장 높은 성능을 보여 윈도우 크기 설정이 중요한 요소임을 확인하였다. 본 연구는 멀티에이전트 시스템의 오류 원인 귀속 과정에서 국소적 맥락 분석이 효과적인 접근임을 확인하였으며, 향후 다양한 환경으로의 확장 및 활용이 가능할 것으로 기대한다.

▸ **주제어**: LLM 멀티에이전트 시스템, 오류 원인 귀속, 윈도우 증강, 국소적 맥락

• First Author: Seungkeon Lee, Corresponding Author: Namgyu Kim
*Seungkeon Lee (sgrhee3@kookmin.ac.kr), Graduate School of Business IT, Kookmin University
**Namgyu Kim (ngkim@kookmin.ac.kr), Graduate School of Business IT, Kookmin University
• Received: 2026. 04. 01, Revised: 2026. 05. 02, Accepted: 2026. 05. 07.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

딥 러닝(deep learning) 분야가 빠르게 발전하면서, LLM(Large Language Model)을 활용한 연구가 활발히 진행되고 있다. LLM이란 자연어 처리(natural language processing) 분야의 사전학습 모델(pre-trained model) 중 하나이며, 수십억 개 이상의 파라미터로 구성되어 대규모 텍스트 데이터를 학습한 대형 언어 모델이다. 수년 전 LLM의 성능이 파라미터의 규모, 학습 데이터의 양, 그리고 컴퓨팅 자원과 비례하는 상관관계가 존재한다는 스케일링 법칙(scaling laws)[1]이 제안되었으며, 이에 따라 OpenAI의 GPT[2], Meta의 LLaMA[3], Google의 PaLM[4], 그리고 DeepSeek의 DeepSeek-R1[5] 등 다양한 모델이 등장하며 자연어 처리 분야에서 LLM의 영향력이 커지고 있다.

LLM은 다양한 분야의 대규모 텍스트 데이터를 학습하며, 문장의 의미, 문법, 흐름 및 구조를 정확하게 이해하고 생성할 수 있다. 이를 통해 LLM은 텍스트 생성, 요약, 추론, 그리고 감정 분석과 같은 광범위한 분야의 작업을 수행하며, 최근에는 LLM을 평가자(judge)로 활용하여 인간 평가를 대체하는 사례[6]도 증가하며 그 우수성을 입증하고 있다. 그러나 단일 LLM은 뛰어난 성능에도 불구하고, 복잡한 계획 수립(planning)[7], 다단계 추론[8] 및 긴 문맥 추론[9] 등 단계적 의사결정이나 긴 문맥 입력을 요구하는 작업에서 취약성을 보인다는 한계를 갖는다. 이는 단일 LLM만으로는 복잡한 작업을 수행하는 데 구조적 제약이 존재함을 의미하며, 이를 보완하기 위한 새로운 접근 방식이 요구되고 있다.

단일 LLM의 한계를 극복하기 위한 다양한 시도 가운데, 최근 멀티에이전트 시스템에 대한 관심이 집중되고 있다. 멀티에이전트 시스템이란 다수의 LLM 에이전트가 역할을 분담하고 상호작용하여 공동의 목표를 달성하는 구조를 의미하며, 이러한 접근은 단일 LLM의 한계를 보완하기 위한 연구 흐름으로 확장되어 왔다[10]. 구체적으로, 멀티에이전트 연구는 토론(debate) 및 다중 관점 제시(multi-perspective reasoning)[11], 검증(verification) 및 수정 과정을 반복하는 순환적 개선[12] 등 다양한 형태로 발전해 왔다. 이러한 구조적 발전을 바탕으로 AutoGen[13], CAMEL[14], ChatDev[15], 그리고 MetaGPT[16]와 같은 멀티에이전트 프레임워크(framework)에 대한 연구가 활발히 진행되고 있으며, 시스템을 체계적으로 설계하고 운영하기 위한 연구 또한 활발히 진행되고 있다. 이와 같은 연구 흐름은 실제 복잡한 환경에서 멀티에이전트 시스템의 문제 해결 능력을 향상시키는 방향으로 이어지고 있으며, 그 결과 복잡한 작업 수행에서 단일 LLM 대비 멀티에이전트 시스템의 성능 우위가 보고되고 있다. 이러한 성능에 힘입어 멀티에이전트 시스템은 의료[17], 금융[18], 그리고 소프트웨어 개발[19] 등 다양한 분야에서 활용되며 그 우수성을 입증하고 있다. 그러나 기존 연구는 주로 멀티에이전트 시스템의 성능 향상에만 초점을 맞추어 왔으며, 작업 실패가 왜 발생하는지 체계적으로 분석하는 연구는 상대적으로 충분히 다루어지지 않았다.

이에 최근 연구에서는 멀티에이전트 시스템의 성능 향상뿐만 아니라, 작업 실패 시 실패 원인을 탐색하는 오류 원인 귀속(failure attribution)에 대한 논의가 활발하게 이루어지고 있다. 멀티에이전트 시스템의 순차적 협력 구조에서는 초기 에이전트에서 발생한 오류가 이후 에이전트의 입력으로 전달되어 연쇄적인 오류로 전파될 수 있으며, 이는 전체 작업 실패로 이어질 수 있다. 따라서 실패가 관측된 최종 단계가 아닌, 작업 실패의 근본적인 원인을 제공한 단계를 분석하기 위한 오류 원인 귀속 연구의 중요성이 강조되고 있다.

오류 원인 귀속은 오류 수정 전략을 수립하여 최종적으로 시스템 성능을 향상시킬 수 있는 핵심적 역할을 수행한다. 그러나 대부분의 오류 원인 탐색은 전문가의 수작업 분석에 의존하고 있어, 많은 시간과 인적 자원을 필요로 하는 한계를 갖는다. 최근 이러한 한계를 극복하기 위해 오류 원인 식별을 자동화하려는 시도가 등장하고 있으며, 특히 LLM 멀티에이전트 시스템에서 연쇄적인 실패 과정에서 특정 단계 에이전트의 행동을 수정하였을 때 작업 실패가 성공으로 전환되는 가장 이른 단계를 결정적 오류(decisive error)로 정의하고, 이를 자동으로 식별하는 방법론을 제안한 연구[20]가 주목받고 있다. 해당 연구에서 제안한 기존 오류 원인 귀속 탐색 방식은 각 에이전트의 행동 로그를 주변 맥락과 분리된 독립 단위로 간주하거나, 에이전트 행동 로그 전체에서 전역적으로 오류 원인 귀속 탐색을 수행한다. 이러한 접근은 인접 단계 간 상호작용을 충분히 반영하지 못하기 때문에, 연쇄적 오류가 발생하는 상황에서 결정적 오류를 정확히 식별하기 어렵다는 한계를 갖는다. 기존 방법론의 결정적 오류 식별 결과를 기준으로 허용 오차 범위를 확장할수록 정답 단계 일치율이 상승하였으며, 이는 인접 단계 내에 결정적 오류가 존재할 가능성이 높음을 시사한다.

이에 본 연구에서는, 윈도우 증강(window augmentation)을 통해 결정적 오류 탐색 성능을 향상시

키는 방법론을 제안한다. 구체적으로 제안 방법론은 기존 방법론에서 도출한 잠재적 오류 단계 및 에이전트를 기준으로, 인접 단계의 상호작용을 함께 고려할 수 있도록 윈도우 증강을 수행한다. 이후 윈도우 집중 식별(window focused identification)을 수행하여 증강된 윈도우 내 문맥을 기반으로 결정적 오류를 보다 정확히 식별함으로써, 오류 원인 귀속 성능을 향상시키고자 한다.

본 논문의 이후 구성은 다음과 같다. 우선 다음 2장에서는 본 연구와 관련된 선행 연구를 소개하고, 3장에서는 본 연구가 제안하는 윈도우 증강 및 윈도우 집중 식별 방법을 설명한다. 4장에서는 제안 방법론의 실험 설계와 실험 결과를 소개하고, 윈도우 크기에 따른 성능 변화를 분석하며, 다양한 LLM 및 데이터 환경에서의 일반화 성능을 검증한다. 마지막으로 5장에서는 본 연구의 기여와 한계를 정리하고 향후 연구 방향을 논의한다.

II. Preliminaries

1. Large Language Models

LLM(Large Language Model)이란 자연어 처리(natural language processing) 분야에서 널리 활용되어 온 사전학습 언어 모델(pre-trained language model)[21]을 대규모로 확장한 형태로, 방대한 텍스트 데이터를 기반으로 학습된 대형 신경망 언어 모델을 의미한다. LLM의 성능이 파라미터의 수, 학습 데이터의 양, 그리고 컴퓨팅 자원과 비례한다는 스케일링 법칙(scaling laws)[1]이 제안되면서, GPT[2], LLaMA[3], PaLM[4], 그리고 DeepSeek-R1[5]과 같은 LLM 모델이 등장하며 LLM의 규모가 커지고 있다[22]. LLM은 방대한 텍스트 데이터를 통해 언어의 문법적 구조와 의미적 관계를 학습함으로써, 단어 수준을 넘어 문장 및 문단 단위의 문맥 정보를 효과적으로 반영할 수 있다. 그 결과 텍스트 생성, 요약, 질의응답, 분류, 그리고 추론 등 다양한 자연어 처리 작업에서 높은 성능을 보인다[23].

LLM의 중요한 특징 중 하나는, 추가적인 파인튜닝(fine-tuning)이나 사전학습 없이 프롬프트 입력으로 제공되는 문맥만을 활용하여 새로운 작업을 수행할 수 있는 문맥 내 학습(in-context learning)[24]이다. 문맥 내 학습은 대표적으로 예시 없이 지시만으로 작업을 수행하는 제로-샷 학습(zero-shot learning)[25]과 소수의 예시만을 활용하는 퓨-샷 학습(few-shot learning)[2]이 있으며, 이를 통해 LLM을 다양한 도메인에 적용 가능한 범용적인 추론 모델로 활용하고 있다. 또한, 입력 프롬프트의 구성 방식은 문맥 내 학습 환경에서 LLM이 생성하는 추론 과정과 최종 출력에 직접적인 영향을 미치는 요소[26]로 알려져 있으며, 이에 따라 프롬프트를 보다 정교하게 설계하려는 연구가 활발하게 진행되고 있다. 이러한 연구 결과의 대표적인 예로 CoT(Chain-of-Thought)[27]를 들 수 있다. CoT는 문제 해결 과정을 여러 단계의 중간 추론으로 분해하여 중간 사고 과정을 순차적으로 표현하도록 유도함으로써, 복잡한 추론 과제에서 LLM의 성능을 향상시키는 대표적인 접근으로 알려져 있다.

LLM이 뛰어난 언어 이해 및 추론 능력을 보임에도 불구하고, 복잡한 계획 수립[7]이나 다단계 추론[8], 그리고 긴 문맥 추론[9]이 요구되는 작업 환경에서는 단일 모델 구조로 인해 이를 효과적으로 처리하는 데 제약이 존재한다. 즉, 단일 LLM은 입력된 문맥을 연속(sequence)으로 처리하며 추론을 수행하기 때문에, 작업이 여러 단계로 구성되거나 장기간에 걸쳐 수행되는 경우 전체 작업 흐름을 일관되게 유지하고 관리하는 데 어려움이 발생할 수 있다. 이러한 한계는 특히 계획 수립, 실행 및 검증과 같이 서로 다른 역할을 하나의 추론 과정에서 동시에 처리해야 하는 상황에서 두드러지게 나타나며, 이는 단일 LLM이 각 단계의 역할을 명확히 분리하여 수행하기 어렵다는 구조적인 특성에 기인한다[8]. 이러한 구조적 특성은 단일 LLM 기반 접근이 복잡한 작업을 처리하는 데 근본적인 한계를 갖고 있음을 의미하며, 이를 극복하기 위한 새로운 접근의 필요성을 제기한다.

2. Multi-Agent Systems

단일 LLM의 한계를 보완하기 위한 접근으로, 최근 LLM 멀티에이전트 시스템(multi-agent systems)이 주목받고 있다. LLM 멀티에이전트 시스템이란 다수의 자율적인 LLM 에이전트(agent)가 특정 역할을 분담하여 작업을 수행하고, 각 에이전트의 출력이 다른 에이전트의 입력으로 활용됨으로써 전체 문제 해결 과정을 구성하는 시스템을 의미한다[28]. 이러한 접근은 단일 LLM이 독립적으로 처리하기 어려운 복합적인 문제를 구조적으로 분해하여 처리할 수 있도록 한다는 점에서, 전통적인 멀티에이전트 시스템 연구에서 제안되어 온 분산적 문제 해결 구조[29]를 LLM 기반 환경으로 확장한 형태로 이해할 수 있다.

LLM 멀티에이전트 시스템에서 에이전트의 동작은 기능적 관점에서 프로파일(profile), 메모리(memory), 계획(planning), 그리고 행동(action)과 같은 구성 요소들로 구분할 수 있다[10]. 프로파일은 에이전트의 역할, 성향,

또는 목표와 같은 정체성 정보를 정의하며, 메모리는 에이전트가 문제 해결 과정에서 획득한 정보를 저장하고 활용하는 기능을 담당한다. 계획 모듈은 현재 상태와 목표를 바탕으로 수행할 행동을 결정하며, 행동 모듈은 계획된 내용을 기반으로 외부 환경과 상호작용하거나 출력을 생성한다. 이러한 구성 요소의 분리는 에이전트가 과거 정보를 활용하고 미래 행동을 계획할 수 있도록 하여, 복잡한 작업을 수행할 수 있게 한다.

LLM 멀티에이전트 시스템의 동작은 개별 에이전트의 독립적인 추론 결과의 단순한 집합이 아닌, 에이전트 간 메시지 전달(message passing)을 통해 정보와 상태를 교환하는 복합적인 상호작용 과정으로 이해할 수 있다. LLM 멀티에이전트 시스템은 메시지 기반 상호작용에 의존하며, 문제 해결 전략 방식에 따라 대표적으로 계층(hierarchical)[30]과 병렬(parallel)[31] 구조로 구분된다. 계층 구조에서는 상위 에이전트가 전체 계획을 조율하고 하위 에이전트가 세부 작업을 수행하는 방식으로 상호작용이 이루어지며, 병렬 구조에서는 다수의 에이전트가 동일한 문제에 대해 독립적으로 추론한 결과를 통합한다. 이러한 상호작용 구조는 문제 해결 과정을 여러 구성 요소에 걸쳐 분산시키는 동시에, 구성 요소 간의 결합을 통해 전체 시스템의 동작을 형성한다는 특징을 갖는다.

에이전트 간 상호작용 방식을 확장함으로써 LLM 멀티에이전트 시스템의 성능을 향상시키기 위한 다양한 연구가 제안되어 왔으며, 대표적으로 토론(debate) 및 다중 관점 제시(multi-perspective reasoning)[11], 검증과 수정 과정을 반복하는 순환적 개선[12] 구조 등이 있다. 이러한 발전을 바탕으로, AutoGen[13], CAMEL[14], ChatDev[15], 그리고 MetaGPT[16]와 같이 에이전트의 역할과 상호작용 절차를 명시적으로 정의하여 복잡한 LLM 멀티에이전트 시스템을 체계적으로 구성하려는 프레임워크(framework) 연구가 활발히 진행되고 있다. 이와 같은 연구 흐름은 실제 복잡한 환경에서 멀티에이전트 시스템의 문제 해결 능력을 향상시키는 방향으로 이어지고 있으며, 그 결과 의료[17], 금융[18], 소프트웨어 개발[19], 그리고 게임[32] 등 다양한 분야에 활용되면서 그 우수성을 입증하고 있다.

3. Failure Attribution in Multi-Agent Systems

LLM 멀티에이전트 시스템에서 오류 원인 귀속(failure attribution)이란, 시스템 수행 과정에서 관측된 실패를 기준으로 해당 실패가 발생한 원인이 되는 단계, 에이전트, 또는 상호작용을 식별하는 문제이다[20]. LLM 멀티에이전트 시스템에서는 에이전트 간 상호작용이 단계적으로 연결된 실행 흐름을 형성하며, 개별 에이전트의 판단은 이후 에이전트의 의사결정에 영향을 미치는 구조적 의존성을 갖는다[33]. 이러한 구조로 인해 특정 단계에서 생성된 부정확한 정보나 판단은 이후 에이전트에게 전달되며, 최종 출력에서 관측되는 작업 실패는 단일 에이전트의 오류가 아닌, 여러 단계에 걸쳐 누적된 결과로 나타난다.

LLM 멀티에이전트 시스템의 연쇄적 상호작용 특성으로 인해, 시스템에서 오류가 발생한 원인을 명확히 식별하는 데 추가적인 분석이 필요하다. 실제 시스템 운용 환경에서는 최종 출력의 성공 또는 실패 여부만 관측되는 경우가 일반적이며, 어떤 에이전트의 어떤 판단이 실패에 기여했는지 식별하기 위해서는 실행 로그 전반에 대한 분석이 요구된다[34]. 실행 로그를 통한 오류 분석은 주로 전문가가 수작업으로 검토하는 방식에 의존하고 있으며, 에이전트 수와 상호작용 단계가 증가할수록 분석에 소요되는 시간과 인적 자원은 크게 증가한다. 그럼에도 불구하고, 기존 LLM 멀티에이전트 시스템 연구는 주로 성능 향상과 작업 성공률 개선을 목표로 설계 및 평가되어 왔으며, 작업 실패 이후의 원인 분석은 상대적으로 덜 주목받아 왔다[35]. 시스템 규모와 에이전트 간 상호작용 복잡성이 증가함에 따라, 실패 원인을 설명하고 수정 지점을 규명하기 위한 오류 원인 귀속의 필요성이 최근 들어 신뢰성 및 유지보수 관점에서 점차 강조되고 있다.

이러한 문제의식을 바탕으로, 최근 LLM 멀티에이전트 시스템에서 발생하는 실패 원인을 체계적으로 분석하고 분류하려는 연구가 진행되었다[36]. 해당 연구에서는 시스템에서 발생하는 실패 원인을 계획 오류, 의사소통 실패 및 정보 불일치 등 14개의 유형으로 분류 및 구조화함으로써, 시스템에서 관측되는 실패 양상을 체계적으로 정리하였다. 이러한 분석적 접근에 더해, 시스템의 실행 로그를 기반으로 오류 원인 귀속을 자동화하려는 연구가 활발히 진행되고 있으며, 대표적으로 실패가 관측된 시점을 기준으로 이전 단계들 중 어떤 단계 및 에이전트의 행동이 작업 실패에 기여하였는지 식별하는 연구[20]가 있다. 해당 연구에서는 시스템의 연쇄적인 오류 전파 과정에서, 특정 단계 에이전트의 행동을 수정하였을 때 작업 실패가 성공으로 전환되는 가장 이른 단계를 결정적 오류(decisive error)로 정의하였다. 이를 통해, 시스템 실행 로그의 시간적 흐름을 기반으로 결정적 오류를 자동으로 식별하는 방법을 제안하였다. 이러한 접근은 LLM 멀티에이전트 시스템에서 오류 원인 귀속을 자동화하는 데 기여하였으나, 결정적 오류 식별에 필요한 상호작용 정보가 여러 인접 단

계에 분산되어 있을 경우, 해당 정보를 충분히 활용하지 못해 정확한 오류 원인 귀속에 한계가 존재한다.

III. The Proposed Method

1. Overall Research Process

본 장에서는 LLM 멀티에이전트 시스템의 오류 원인 귀속 성능을 향상시키기 위한 윈도우 증강 기반 방법론을 새롭게 제안하고, 전체 방법론의 주요 단계를 소개한다. 제안 방법론은 시스템 실행 로그로부터 작업 실패를 유발한 결정적 오류 단계 및 에이전트를 식별하는 것을 목표로 하며, 전체 과정은 Fig. 1과 같다.

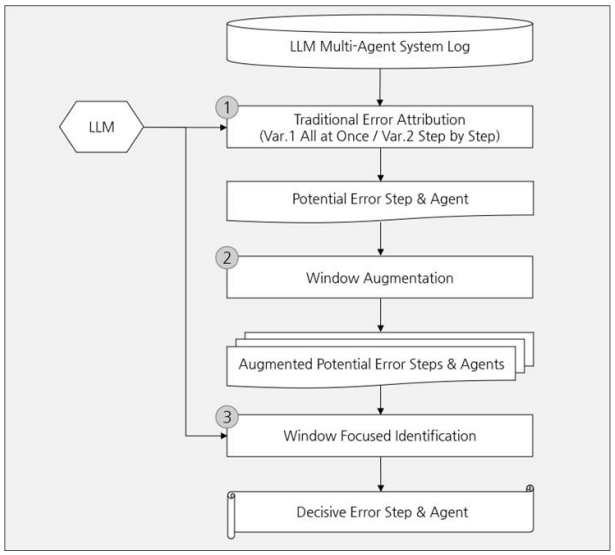

Fig. 1. Overall Research Process

먼저 기존 LLM 멀티에이전트 시스템의 전통적인 오류 원인 귀속 방법론[20]을 적용하여, 시스템 실행 로그로부터 잠재적 오류 단계 및 에이전트(potential error step & agent)를 일차적으로 식별한다(①). 이후 도출된 잠재적 오류 단계 및 에이전트를 기준으로 분석 범위를 확장하는 윈도우 증강(window augmentation)을 수행하여(②), 증강된 잠재적 오류 단계 및 에이전트를 도출한다. 마지막으로, 증강된 윈도우를 기반으로 윈도우 집중 식별(window focused identification)을 수행하여, 결정적 오류 단계 및 에이전트를 식별한다(③). 각 단계에 대한 세부적인 프로세스는 다음 절에서 설명하며, 실제 데이터를 적용한 제안 방법론의 성능 평가는 4장에서 소개한다.

2. Traditional Error Attribution

본 절에서는 기존 LLM 멀티에이전트 시스템의 전통적인 오류 원인 귀속 방법론을 적용하여 잠재적 오류 단계 및 에이전트를 식별하는 단계를 소개한다(①). 본 단계는 다음 단계(②)의 탐색 시작점으로 사용하기 위한 잠재적 오류 단계 및 에이전트를 식별하는 것을 목표로 한다. 본 연구에서 잠재적 오류란 전통적인 오류 원인 귀속을 통해 도출된 결과인 후보 단계 및 에이전트를 의미한다.

전통적인 오류 원인 귀속에서는 Fig. 2와 같이 LLM을 이용한 두 가지 분석 방식을 제안하였다. 두 방식은 모두 동일한 실행 로그를 분석 대상으로 사용하지만, 오류를 탐색하는 방식과 분석 단위에서 차이를 갖는다.

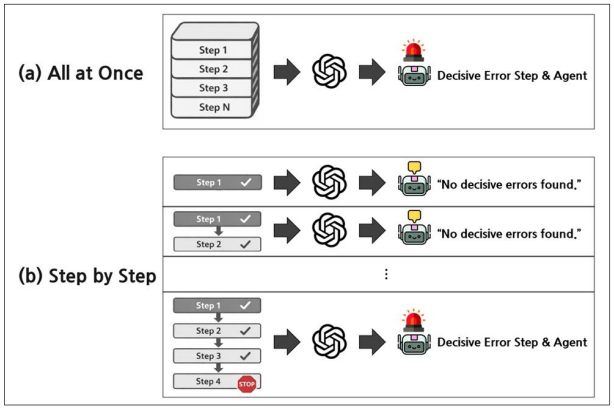

Fig. 2. Illustration of Baseline Methodologies[20]

전역 분석(all at once)은 Fig. 2(a)와 같이 시스템 실행 로그 전체를 한 번에 입력하여, 잠재적 오류 단계 및 에이전트를 탐색하는 접근이다. 이 과정에서 전체 실행 로그를 개별 단계로 분리하여 탐색하지 않으며, 전체 실행 로그에 나타나는 모든 에이전트의 판단과 상호작용을 하나의 맥락으로 간주하여 탐색한다. 순차 분석(step by step)은 Fig. 2(b)와 같이 시스템 실행 로그를 시간 순서에 따라 단계별로 누적하여 분석하며, 각 시점에서 해당 단계가 잠재적 오류에 해당하는지 순차적으로 판단하는 방식이다. 해당 방식에서는 LLM 멀티에이전트 시스템에서 오류가 연쇄적으로 형성될 수 있다는 점을 고려하여, 특정 단계가 잠재적 오류로 판단되면 그 시점에 로그 분석을 종료하고 해당 단계와 에이전트를 잠재적 오류로 식별한다. 이와 같이 전역 분석과 순차 분석은 서로 다른 분석 단위의 시스템 실행 로그 처리를 통해 잠재적 오류 단계, 에이전트, 그리고 판단 근거(reason)를 생성하며, 정규표현식 기반 추출(parsing) 과정을 통해 각 요소를 구조화된 형태로 도출한다.

그러나 전통적인 오류 원인 귀속 방법론은 인접 단계 간 상호작용을 충분히 반영하지 못하는 한계를 갖는다. Fig. 3은 순차 분석에서 식별한 결정적 오류 단계를 기준으로 허용 오차 범위를 5까지 확장했을 때 정답 단계 일치율을 분석한 결과이다.

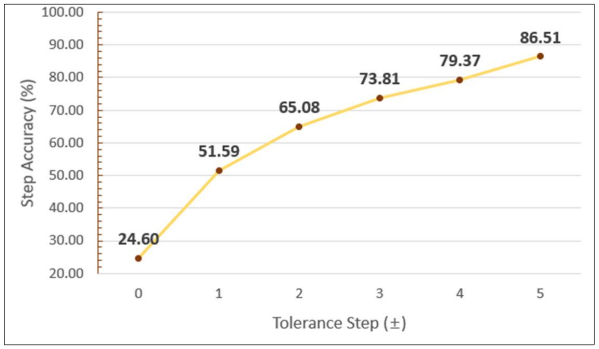

Fig. 3. Relaxed Step Accuracy by Tolerance Step of Step by Step Method

분석 결과, 허용 오차 범위를 1에서 5까지 확장했을 때, 정답 단계 일치율이 24.60%에서 86.51%까지 상승함을 보였다. 이는 전통적인 오류 원인 귀속 방법론의 결정적 오류 단계 식별 결과가 정답 단계와 일치하지 않더라도, 인접 단계 내에 결정적 오류 단계가 존재할 가능성이 높음을 시사한다.

이에 본 연구에서는 전통적인 오류 원인 귀속 방법론을 통해 도출된 잠재적 오류 식별 결과를 기준으로 분석 범위를 확장하여(②), 확장된 윈도우를 기반으로 프롬프트를 생성하고, LLM을 이용하여 결정적 오류 단계 및 에이전트를 재식별하는(③) 방법론을 제안한다.

3. Window Augmentation

본 절에서는 Fig. 1에서 이루어지는 과정 중, 본 연구에서 새로 제안하는 윈도우 증강 과정(②)을 소개한다. 윈도우 증강은 전통적인 오류 원인 귀속 방법론을 통해 도출된 잠재적 오류 단계 주변의 맥락을 함께 고려하기 위한 절차이다. LLM 멀티에이전트 시스템에서는 특정 단계에서 발생한 오류가 인접 단계와 상호작용을 통해 형성되거나 확대될 수 있다. 그러나 기존 방법론은 잠재적 오류 단계의 전후 상호작용 맥락은 별도의 분석 대상으로 재검토하지 않으며, 결과적으로 오류의 형성 및 전파 과정을 충분히 반영하지 못하는 한계를 갖는다.

따라서 잠재적 오류 단계를 결정적 오류 단계로 확정하는 것이 아닌, 인접 단계까지 포함하는 국소적 맥락으로 확장하여 결정적 오류를 재식별할 필요가 있다. 이에 본 연구에서는 잠재적 오류 단계를 중심으로 인접 단계까지 포함하도록 윈도우 증강을 수행하여 국소적 맥락을 구성하고자 한다.

구체적으로, 본 단계에서는 잠재적 오류 단계 s_i를 중심으로 양방향으로 a단계씩 확장하여 윈도우를 구성한다. 전체 실행 로그의 단계 집합을 $\{s_1, s_2, \ldots, s_T\}$라 할 때, 단계 s_i에 대한 윈도우 W_i는 (식 1)과 같이 정의된다.

$$(\text{식}1)\ W_i = \{s_j \mid \max(1, i-a) \le j \le \min(T, i+a)\}$$

Fig. 4은 전체 로그 길이 T값이 10, 윈도우 범위 a값이 3으로 설정된 경우를 가정하며, 잠재적 오류 단계 s_i에 따라 윈도우가 이동하는 예를 보인다.

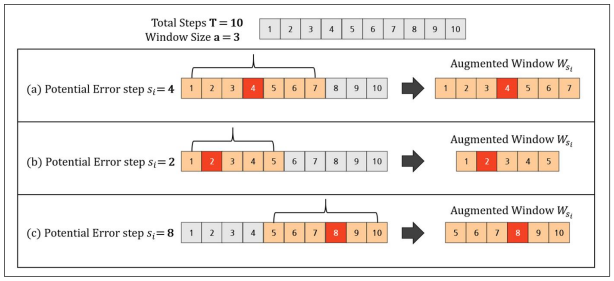

Fig. 4. Illustration of Window Augmentation Process

Fig. 4(a)와 같이 s_i값이 4인 경우, 윈도우는 1부터 7까지 확장되며, Fig. 4(b)와 같이 s_i값이 2인 경우에는 하한 경계 조건에 의해 윈도우는 1부터 5까지로 구성된다. 한편, Fig. 4(c)와 같이 s_i값이 8일 경우, 상한 경계 조건에 의해 윈도우는 5부터 10까지로 구성된다. 이와 같이 전체 로그 길이와 잠재적 오류 단계의 위치에 따라 윈도우 범위는 동적으로 조정된다.

이러한 과정을 통해 구성된 윈도우는 이후 단계(③)의 입력으로 사용되며, 확장된 국소적 맥락을 기반으로 프롬프트를 생성하고, 결정적 오류 단계를 재식별하는 오류 원인 귀속을 수행한다.

4. Window Focused Identification

본 절에서는 Fig. 1의 과정 중 프롬프트 생성 및 윈도우 집중 식별을 설명한다(③). 해당 과정은 단계 ②에서 확장된 윈도우 Ws_i를 기반으로 결정적 오류를 탐색하기 위한 프롬프트를 생성하고, 결정적 오류 단계를 재식별하는 절차이다. Fig. 5는 윈도우 증강을 통해 국소적 맥락을 구성하고, 윈도우 집중 식별을 통해 최종 결정적 오류 단계 및 에이전트를 도출하는 윈도우 집중 식별 단계를 나타낸다.

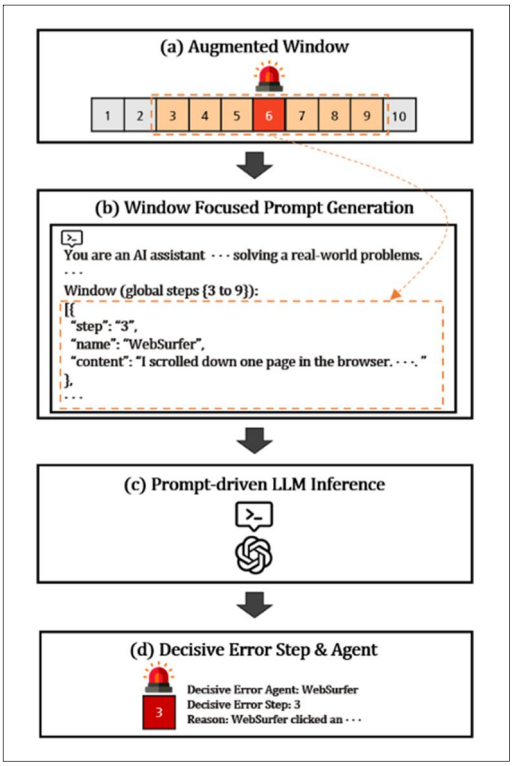

Fig. 5. Window Focused Identification Process

Fig. 5(a)는 단계 ②에서 잠재적 오류 단계를 중심으로 윈도우 증강을 마친 결과이며, 예시에서는 단계 6을 중심으로 단계 3부터 9까지 국소적 맥락이 형성되었다. Fig. 5(b)는 증강된 윈도우에 포함된 실행 로그와 잠재적 오류 단계 및 에이전트를 기반으로 LLM 입력 프롬프트를 구성하는 과정을 나타낸다. 해당 프롬프트는 윈도우 내 단계 중 가장 이른 오류 단계를 선택하도록 하는 지시문과 출력 형식 제약을 포함한다. Fig. 5(c)는 구성된 프롬프트를 LLM에 입력하여 결정적 오류 탐색을 수행한다. 마지막으로, Fig. 5(d)는 Fig. 5(c)의 결과, 즉 윈도우 내 단계 중 가장 이른 실패 기여 단계를 결정적 오류로 식별하여 출력한다.

프롬프트 설계를 위해 본 연구에서는 W_i에 포함된 실행 로그와 잠재적 오류 단계 및 에이전트를 바탕으로 LLM 입력 프롬프트를 구성한다. 구체적으로, 국소적 맥락을 LLM에 제공함으로써, LLM이 특정 단계의 출력이 이후 단계의 입력으로 전달되는 연쇄적 흐름을 파악하고, 오류가 최초로 형성된 단계와 이후 전파된 단계를 구분하여 결정적 오류 단계 및 에이전트를 재식별할 수 있도록 프롬프트를 구성한다. Table 1은 LLM 멀티에이전트 시스템에 주어진 문제(problem)와 정답(answer)의 예를, 그리고 Fig. 6은 이를 기반으로 구성된 프롬프트 예시를 보인다.

Table 1. Example Case

Attribute	Value
Problem (Query)	Where can I take martial arts classes within a five-minute walk from the New York Stock Exchange after work (7-9 pm)?
Answer (Ground Truth)	Renzo Gracie Jiu-Jitsu Wall Street

```
You are an AI assistant tasked with evaluating the correctness of each step
in an ongoing multi-agent conversation aimed at solving a real-world problem.
Select exactly ONE global step index that best represents the true critical mistake.
This should be the earliest step that directly leads to the failure.

Problem: Where can I take martial arts classes within a five-minute walk from the
New York Stock Exchange after work (7-9 pm)?
Answer: Renzo Gracie Jiu-Jitsu Wall Street
Window (global steps 3 to 9):
· · ·
Respond ONLY in the format:
Agent: <exact agent name>
Step: <global step integer>
Reason: <short reason>
```

Fig. 6. Example of Window-Based Prompt Construction

윈도우 집중 프롬프트가 구성된 이후, 이를 LLM에 입력하여 윈도우 집중 식별을 수행한다. LLM은 프롬프트의 지시문과 출력 형식 제약에 따라 윈도우 내에서 오류가 처음 형성되고 전파되기 시작한 실패 기여 단계를 결정적 오류 단계로 식별하고, 해당 에이전트와 판단 근거를 함께 출력한다. Fig. 7은 윈도우 집중 식별의 수행 결과로 생성되는 최종 출력 형식의 예시를 나타낸다.

```
Decisive Error Agent: WebSurfer
Decisive Error Step: 3
Reason: The WebSurfer failed to extract the correct information
(addresses and class schedules) from the initial search results and
instead clicked on irrelevant links, leading to a loop of unproductive
actions.
```

Fig. 7. Example of Final Output

이처럼 윈도우 집중 식별은 확장된 윈도우 내에서 국소적 맥락을 활용하여 실패 기여 단계를 재식별하는 과정으로, 이를 통해 연쇄적 상호작용 구조를 갖는 LLM 멀티에이전트 시스템에서 보다 정밀한 오류 원인 귀속을 수행할 수 있다.

IV. Experiment

1. Experiment Overview

본 장에서는 제안 방법론을 실제 데이터에 적용한 실험 과정 및 성능 분석 결과를 소개한다. 실험에는 전통적인

오류 원인 귀속 연구[20]에서 구축한 데이터셋(dataset) 중, Captain Agent[37] 기반 멀티에이전트 시스템 실행 로그로부터 구축된 'Algorithm Generated' 데이터를 사용하였다. 본 데이터는 결정적 오류 단계 및 에이전트 레이블(label)로 이루어진 텍스트 분류 데이터이며, 126개의 사례로 구성되어 있다. 각 사례는 LLM 멀티에이전트 시스템의 에이전트 간 상호작용 로그 전체와 함께, 결정적 오류 단계 및 에이전트 레이블을 포함한다. 하나의 사례는 최소 5단계 이상, 최대 10단계 이하의 상호작용 단계를 포함하며, 각 단계에는 에이전트명과 해당 에이전트의 출력 결과가 포함되어 있다.

실험에서는 윈도우 증강 범위를 3으로 설정하였으며, 윈도우 집중 식별을 수행하는 LLM으로 GPT-4o를 사용하였다. 제안 방법론과 비교 방법론의 성능 비교 실험 전체 프로세스는 Fig. 8과 같다. 본 실험 환경은 Python 3.12를 통해 구축하였으며, 구체적인 H/W 및 S/W 환경은 Table 2와 같다.

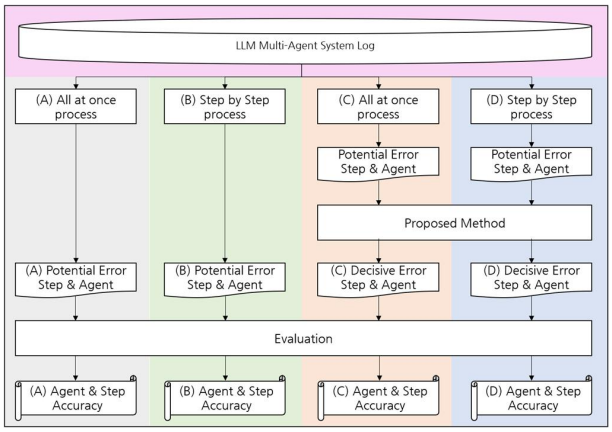

Fig. 8. Overall Process of Performance Evaluation

Table 2. System Environments

H/W	CPU	24 Core
	RAM	32 GB
	OS	Windows 11
S/W	Python	3.12.8
	OpenAI SDK	1.59.3
	LLM	GPT-4o

Fig. 8의 (A)와 (B)는 전통적인 오류 원인 귀속[20] 방법론인 전역 분석과 순차 분석을 적용하여 잠재적 오류 단계 및 에이전트를 도출하는 과정이며, (C)와 (D)는 각각 (A)와 (B)에서 도출된 잠재적 오류를 기준으로 제안 방법론을 적용하여 결정적 오류 단계 및 에이전트를 식별하는 과정이다. 실험에서는 (A), (B), (C), 그리고 (D)에서 예측된 단계 및 에이전트 결과를 정답 레이블과 비교하여, 단계 정확도(step accuracy)와 에이전트 정확도(agent accuracy)를 산출하였다[20]. 단계 정확도는 예측된 결정적 오류 단계가 정답 단계와 일치한 비율을 의미하며, 에이전트 정확도는 예측된 결정적 오류 에이전트가 정답 에이전트와 일치한 비율을 의미한다.

2. Experimental Results and Analysis

본 절에서는 Fig. 8에 제시된 실험 설계를 기반으로 각 방법론의 오류 원인 귀속 성능을 비교한다. Table 3은 각 분석 과정(process)에 대해, 단계 정확도와 에이전트 정확도를 산출한 결과를 나타낸다.

Table 3. Accuracy of Each Process

Process	Step Accuracy	Agent Accuracy
(A)	0.1746	0.4444
(B)	0.2460	0.3571
(C)	0.4444	0.5873
(D)	**0.4524**	**0.6270**

먼저 전통적인 오류 원인 귀속 방법론인 (A)와 (B)의 오류 원인 귀속 성능의 경우, 순차 분석 (B)가 전역 분석 (A)에 비해 단계 정확도는 다소 높은 결과를 보였으나, 에이전트 정확도는 상대적으로 낮게 나타났다. 이는 실행 로그를 단계적으로 분석할 경우 결정적 오류 단계 식별에는 도움이 될 수 있으나, 결정적 오류 에이전트를 식별하는 데에는 한계가 있음을 의미한다.

제안 방법론을 적용한 (C)와 (D)의 경우 두 지표 모두 기존 방법론 대비 전반적으로 향상된 성능을 보였다. 특히 (D)는 단계 정확도 0.4524, 에이전트 정확도 0.6270으로 가장 높은 성능을 보였으며, 이는 (B)의 단계 정확도 0.2460, 에이전트 정확도 0.3571과 비교할 때 각각 약 83.9%와 75.6% 향상된 결과이다.

3. Performance Variation With Window Sizes

본 절에서는 윈도우 크기(window size)에 따른 오류 원인 귀속 성능 변화를 분석한다. Fig. 9는 Fig. 8의 (C)와 (D)에서, 윈도우 크기를 1부터 5까지 변화시켰을 때의 단계 정확도와 에이전트 정확도의 변화를 나타낸다.

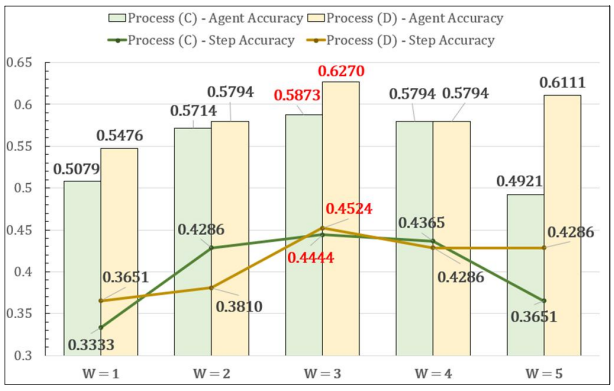

Fig. 9. Performance Comparison across Window Sizes for Processes (C) and (D)

Fig. 9에서 (C)의 경우, 윈도우 크기가 3일 때 에이전트 정확도 0.5873으로 가장 높은 성능을 기록하였으며, 이후 윈도우 크기가 증가함에 따라 성능이 감소하는 경향이 나타났다. (D)에서도 유사한 경향이 관찰되었으며, 윈도우 크기가 3일 때 에이전트 정확도 0.6270으로 가장 높은 성능을 보였다. 윈도우 크기가 증가함에 따라 성능이 감소하는 경향은 과도한 문맥 확장이 오류 원인 귀속에 불필요한 정보를 포함시킨 것에 기인한 것으로 해석된다.

전반적으로 (D)가 (C)보다 우수한 성능을 보였으며, 이는 순차 분석을 기반으로 한 제안 방법론이 오류 원인 귀속에 보다 효과적으로 작용할 수 있음을 시사한다. 단계 정확도 또한 에이전트 정확도와 유사한 변화 양상을 보였으며, 윈도우 크기 3에서 (C)와 (D)의 단계 정확도는 각각 0.4444와 0.4524로 가장 높은 성능을 기록하였다.

본 실험 결과는 윈도우 크기에 따른 정보 활용 범위의 차이를 나타낸다. 즉, 윈도우 크기가 작을 경우 충분한 오류 전파 정보를 포착하지 못하고, 윈도우 크기가 클 경우 불필요한 정보로 인해 오류 원인 귀속 성능을 저해할 수 있어 적절한 윈도우 크기를 설정하는 것이 오류 원인 귀속 성능을 극대화하는 데 중요한 요소임을 확인하였다.

4. Generalization and Robustness Analysis

본 절에서는 제안 방법론의 일반화 가능성을 검증하기 위해, 본 실험에서 사용된 LLM 모델 및 데이터를 변경하여 추가 실험을 수행한 결과를 소개한다.

모델 변경에 따른 제안 방법론의 일반화(generalization) 성능을 검증하기 위해, 경량 모델인 GPT-4o-mini 모델로 교체하여 'Algorithm-Generated' 데이터에 대해 Fig. 8과 동일한 실험을 수행하였다. 제안 방법론의 윈도우 크기는 3으로 설정하였으며, Table 4는 각 프로세스에 대한 단계 정확도와 에이전트 정확도를 나타낸다.

Table 4. Accuracy of Each Process (GPT-4o-mini)

Process	Step Accuracy	Agent Accuracy
(A)	0.0873	0.3889
(B)	0.1746	0.2857
(C)	0.1825	**0.4762**
(D)	**0.2222**	0.3968

실험 결과, 제안 방법론을 적용한 (C)와 (D)가 전통적인 오류 원인 귀속 방법론인 (A)와 (B) 대비 단계 정확도와 에이전트 정확도 모두에서 향상된 성능을 보였다. 이를 통해, 제안 방법론의 성능 향상이 특정 LLM에 의존하지 않으며, 경량 모델에서도 유효함을 확인하였다.

또한, 데이터의 변경에 따른 제안 방법론의 견고함(robustness)을 검증하기 위해, 본 실험에서 사용한 'Algorithm-Generated' 데이터셋과 상이한 특성을 갖는 'Hand-Crafted' 데이터 58건에 대해 Fig. 8과 동일한 실험 설계로 추가 실험을 수행하였다. 'Hand-Crafted' 데이터는 전통적인 오류 원인 귀속 연구[20]에서 구축한 데이터셋 중 하나이다. 해당 데이터셋은 Magnetic-One[38] 멀티에이전트 프레임워크 기반 시스템 실행 로그로 구성되어 있으며, 에이전트 구성 및 구조를 사람이 수작업으로 구축하였다. 하나의 사례는 최소 5단계 이상, 최대 130단계 이하의 긴 상호작용 단계로 구성되어 있다. LLM 모델은 GPT-4o-mini를 사용하였고, 윈도우 크기는 5로 설정하였으며 실험 결과는 Table 5와 같다.

Table 5. Accuracy of Each Process (Hand-Crafted)

Process	Step Accuracy	Agent Accuracy
(A)	0.0172	0.5000
(B)	0.1207	0.4762
(C)	**0.1897**	0.5172
(D)	0.1724	**0.5517**

실험 결과, 전통적인 오류 원인 귀속 방법론인 (A)와 (B) 대비 제안 방법론 (C)와 (D)가 단계 정확도와 에이전트 정확도 모두에서 향상된 성능을 보였다. 이를 통해, 제안 방법론이 긴 상호작용 단계를 포함하는 멀티에이전트 시스템 환경에서도 일관된 오류 원인 귀속 성능 향상을 제공할 수 있음을 확인하였다.

V. Conclusions

최근 LLM 멀티에이전트 시스템은 복잡한 문제 해결 능력을 바탕으로 다양한 분야에서 활용되며 그 성능을 입증하고 있다. 그러나 시스템의 순차적 상호작용 구조로 인해, 특정 단계에서 발생한 오류가 이후 단계로 전파되어 최종 작업 실패로 이어지는 한계를 갖는다. 이에 본 연구에서는 기존 오류 원인 귀속 방법론이 인접 단계 간 상호작용을 충분히 반영하지 못하는 한계를 개선하기 위해, 윈도우 증강 방법론을 제안하였다.

실험 결과, 제안 방법론은 전통적인 오류 원인 귀속 방법론 대비 단계 정확도와 에이전트 정확도 모두에서 향상된 성능을 보였다. 특히 순차 분석 기반 접근과 결합된 경우 가장 우수한 결과를 보였으며, 제안한 윈도우 증강 방법론이 오류 원인 귀속 성능을 향상시킬 수 있음을 확인하였다. 또한 윈도우 크기에 따른 성능 변화를 분석한 결과, 적절한 윈도우 크기를 설정하는 것이 오류 원인 귀속 성능을 최적화하는 데 중요한 요소임을 확인하였다. 경량 모델 및 이질적인 데이터셋의 추가 실험에서도 일관된 오류 원인 귀속 성능 향상이 확인되어, 제안 방법론의 일반화 가능성을 검증하였다.

본 연구는 전통적인 오류 원인 귀속 방법론의 결과를 탐색의 출발점으로 활용한다는 점에서 기존 연구와 연계된다. 또한, 전통적인 오류 원인 귀속 방법론이 전체 로그를 전역적으로 탐색하거나 각 단계를 독립적으로 분석하는 데 반해, 본 연구는 잠재적 오류 단계를 중심으로 윈도우 증강을 통해 국소적 맥락을 구성하여 재식별을 수행한다는 점에서 차별성을 갖는다.

다만, 본 연구는 일부 데이터셋 및 실험 환경을 기반으로 검증되었으며, 다양한 멀티에이전트 구조에 대한 일반화는 제한적으로 검증되었다. 향후 연구에서는 다양한 LLM 멀티에이전트 프레임워크 및 실제 응용 환경에 제안 방법론을 적용하여 일반화 성능을 보다 폭넓게 검증할 필요가 있다. 또한 오류 원인 귀속 결과를 기반으로 실제 시스템의 오류를 수정하여 성능을 개선하는 통합적 프레임워크로의 확장 연구가 필요하며, 이를 통해 보다 효과적인 오류 원인 귀속 및 시스템 성능 향상이 가능할 것으로 기대한다.

REFERENCES

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, D. Amodei., “Scaling laws for neural language models,” arXiv preprint arXiv:2001.08361, January 2020. DOI: 10.48550/arXiv.2001.08361

[2] T. B. Brown et al., “Language models are few shot learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877-1901, December 2020.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. A. Lachaux, T. Lacroix, B. Roziere, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, “Llama: Open and efficient foundation language models,” arXiv preprint arXiv:2302.13971, February 2023. DOI: 10.48550/arXiv.2302.13971

[4] A. Chowdhery et al., “Palm: Scaling language modeling with pathways,” Journal of Machine Learning Research, vol. 24, no. 240, pp. 1-113, 2023.

[5] D. Guo et al., “DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning,” arXiv preprint arXiv:2501.12948, January 2025. DOI: 10.48550/arXiv.2501.12948

[6] L. Zheng et al., “Judging LLM as a Judge with MT Bench and Chatbot Arena,” in Advances in Neural Information Processing Systems, vol. 36, pp. 46595-46623, December 2023.

[7] H. S. Zheng et al., “NATURAL PLAN: Benchmarking LLMs on Natural Language Planning,” arXiv preprint arXiv:2406.04520, June 2024. DOI: 10.48550/arXiv.2406.04520

[8] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, G. Durrett, “Musr: Testing the limits of chain-of-thought with multistep soft reasoning,” arXiv preprint arXiv:2310.16049, October 2023. DOI: 10.48550/arXiv.2310.16049

[9] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, “Lost in the middle: How language models use long contexts,” Transactions of the Association for Computational Linguistics, vol. 12, pp. 157-173, February 2024. DOI: 10.1162/tacl_a_00638

[10] L. Wang et al., “A survey on large language model based autonomous agents,” Frontiers of Computer Science, vol. 18, no. 6, p. 186345, March 2024. DOI: 10.1007/s11704-024-40231-1

[11] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, S. Shi, Z. Tu, “Encouraging divergent thinking in large language models through multi-agent debate,” in Proceedings of the 2024 Conf. on Empirical Methods in Natural Language Processing, pp. 17889-17904, November 2024. DOI: 10.18653/v1/2024.emnlp-main.992

[12] J. Chen, A. Prasad, S. Saha, E. S. Eskin, M. Bansal, “Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning,” Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, November 2025. DOI: 10.18653/v1/2025.emnlp-main.1660

[13] Q. Wu et al., “Autogen: Enabling next-gen LLM applications via multi-agent conversations,” in Proceedings of the First Conf.

on Language Modeling, July 2024.

[14] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, "CAMEL: Communicative agents for 'mind' exploration of large language model society," Advances in Neural Information Processing Systems, vol. 36, pp. 51991-52008, 2023.

[15] Qian et al., "Chatdev: Communicative agents for software development," Proceedings of the 62nd annual meeting of the association for computational linguistics, vol. 1, pp. 15174-15186, August 2024. DOI: 10.18653/v1/2024.acl-long.810

[16] S. Hong et al., "MetaGPT: Meta programming for a multi-agent collaborative framework," in Proceedings of the Twelfth International Conf. on Learning Representations, January 2024.

[17] K. Zuo, Y. Jiang, F. Mo, P. Lio, "Kg4diagnosis: A hierarchical multi-agent LLM framework with knowledge graph enhancement for medical diagnosis," in Proceedings of AAAI Bridge Program on AI for Medicine and Healthcare, pp. 195-204, 2025.

[18] Y. Xiao, E. Sun, D. Luo, W. Wang, "TradingAgents: Multi-agents LLM financial trading framework," arXiv preprint arXiv:2412.20138, December 2024. DOI: 10.48550/arXiv.2412.20138

[19] M. Sanwal and I. Deva, "An autonomous multi-agent LLM framework for agile software development," International Journal of Trend in Scientific Research and Development, vol. 8, no. 5, pp. 892-898, November 2024.

[20] S. Zhang et al., "Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems," Proceedings of Machine Learning Research, vol 267, pp. 76583-76599, May 2025. DOI: 10.48550/arXiv.2505.00212

[21] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, X. Huang, "Pre-trained models for natural language processing: A survey," Science China Technological Sciences, vol. 63, no. 10, pp. 1872-1897, September 2020. DOI: 10.1007/s11431-020-1647-3

[22] H. Naveed, A. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian, "A comprehensive overview of large language models," ACM Transactions on Intelligent Systems and Technology, vol. 16, no. 5, pp. 1-72, August 2025. DOI: 10.1145/3744746

[23] W. X. Zhao et al., "A survey of large language models," arXiv preprint arXiv:2303.18223, November 2023. DOI: 10.48550/arXiv.2303.18223

[24] Q. Dong et al., "A survey on in-context learning," in Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, November 2024. DOI: 10.18653/v1/2024.emnlp-main.64

[25] T. Kojima, S. Gu, M. Reid, Y. Matsuo, Y. Lwasawa, "Large language models are zero-shot reasoners," Advances in Neural Information Processing Systems, vol. 35, pp. 22199-22213, 2022.

[26] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," ACM Computing Surveys, vol. 55, no. 9, pp. 1-35, January 2023. DOI: 10.1145/3560815

[27] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," Advances in Neural Information Processing Systems, vol. 35, pp. 24824-24837, 2022.

[28] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. Chawla, O. Wiest, X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," arXiv preprint arXiv:2402.01680, April 2024. DOI: 10.48550/arXiv.2402.01680

[29] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," Autonomous Robots, vol. 8, no. 3, pp. 345-383, June 2000. DOI: 10.1023/A:1008942012299

[30] Q. Wang, T. Wang, Z. Tang, Q. Li, N. Chen, J. Liang, B. He, "MegaAgent: A large-scale autonomous LLM-based multi-agent system without predefined SOPs," Findings of the Association for Computational Linguistics: ACL 2025, pp. 4998-5036, July 2025. DOI: 10.18653/v1/2025.findings-acl.259

[31] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, J. Zou, "Mixture-of-Agents enhances large language model capabilities," arXiv preprint arXiv:2406.04692, June 2024. DOI: 10.48550/arXiv.2406.04692

[32] Z. Li, Q. Ji, X. Ling, Q. Liu, "A comprehensive review of multi-agent reinforcement learning in video games," IEEE Transactions on Games, July 2025. DOI: 10.1109/TG.2025.3588809

[33] M. Wooldridge, An introduction to multiagent systems, John Wiley & Sons, 2009.

[34] B. Lin et al., "AgentAsk: Multi-agent systems need to ask," arXiv preprint arXiv:2510.07593, October 2025. DOI: 10.48550/arXiv:2510.07593

[35] K. Zhu et al., "Where LLM agents fail and how they can learn from failures," arXiv preprint arXiv:2509.25370, September 2025. DOI: 10.48550/arXiv.2509.25370

[36] M. Cemri et al., "Why do multi-agent LLM systems fail?," arXiv preprint arXiv:2503.13657, October 2025. DOI: 10.48550/arXiv.2503.13657

[37] L. Song, J. Liu, J. Zhang, S. Zhang, A. Luo, S. Wang, Q. Wu, C. Wang, "Adaptive in-conversation team building for language model agents," arXiv preprint arXiv:2405.19425, March 2025. DOI: 10.48550/arXiv.2405.19425

[38] A. Fourney et al., "Magentic-one: A generalist multi-agent system for solving complex tasks." arXiv preprint arXiv:2411.04468, November 2024. DOI: 10.48550/arXiv.2411.04468

Authors

Seungkeon Lee received the B.A. degree in Computer and Software Engineering from Wonkwang University in 2025 and currently enrolled in Graduate School of Business IT, Kookmin University.

He is interested in LLM Multi-Agents System, Recommender System, and Prompt Engineering.

Namgyu Kim received the B.S. in Computer Engineering from Seoul National University in 1998, M.S. and Ph.D. degrees in Management Engineering from KAIST, Korea, in 2000 and 2007, respectively.

Dr. Kim joined the faculty of the School of Management Information Systems at Kookmin University, Seoul, Korea, in 2007. He served as the Dean of the Graduate School of Business IT at Kookmin University and is currently a professor at the Business IT. He is interested in deep learning, text mining, and data modeling.