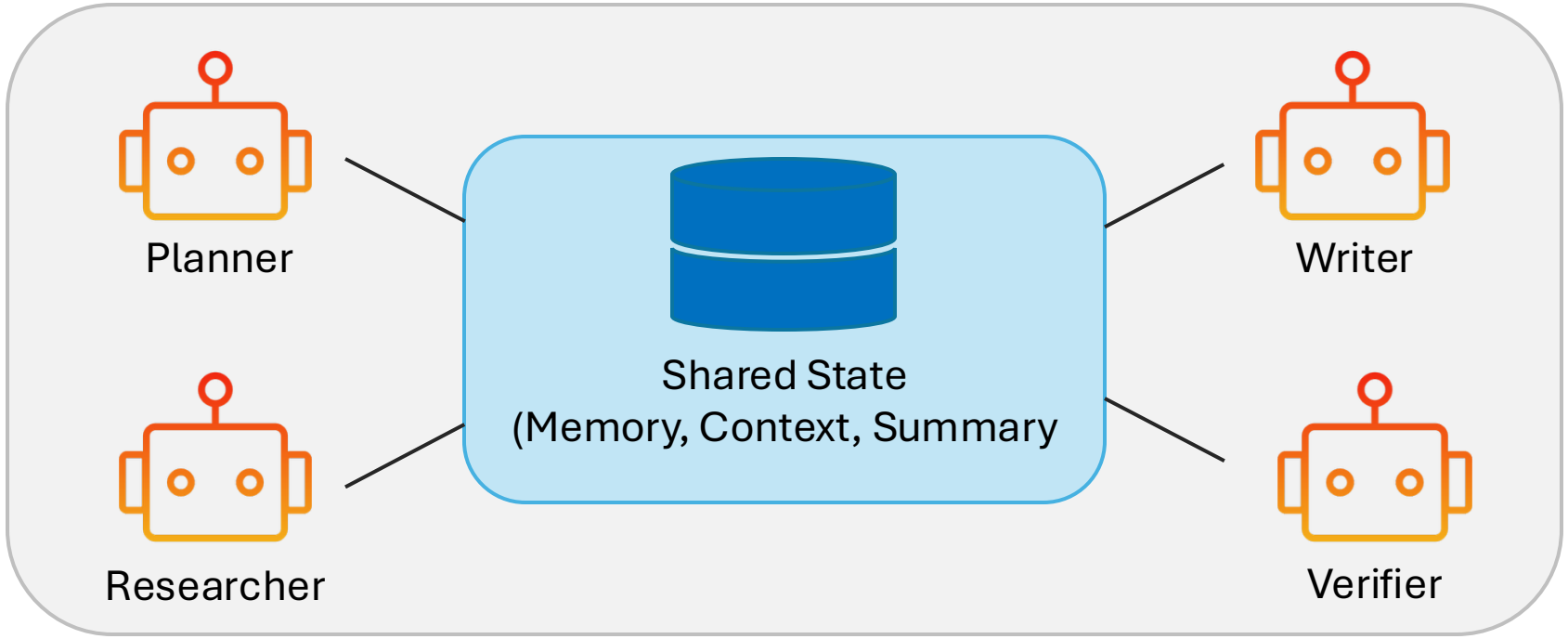
Preventing Memory Contamination in Multi- Agent Systems

1204 정지민

Natural Language Processing
& Artificial Intelligence

Preliminary

How Current Multi-Agent LLM Systems Work?



⚠ 모든 에이전트가 하나의 shared state에 접근하기 때문에 단일 오류가 전체 그룹에 순식간에 확장됨

Current Verification: 대부분 Output이 나온 후 마지막 단계에서 검증. 즉, state-level이 아니라 post-hoc correction으로 동작함

Preliminary

Why Multi-agent Hallucination is dangerous

Single LLM

- 오류는 한 번 생성되고 끝 (Static)
- Stateless → Cascade 효과 없음
- Self-correction 비교적 용이

Multi-Agent System

- Shared state 저장 → 다른 에이전트가 계속 사용
- **Cascade + Amplification + Temporal Propagation**
- 이미 퍼지면 복구 불가능 (Irreversible)

CASE STUDY: MEDICAL CONTAMINATION CHAIN

Multi-agent hallucination은 Time-based propagation 문제이며, 단일 LLM보다 훨씬 치명적임

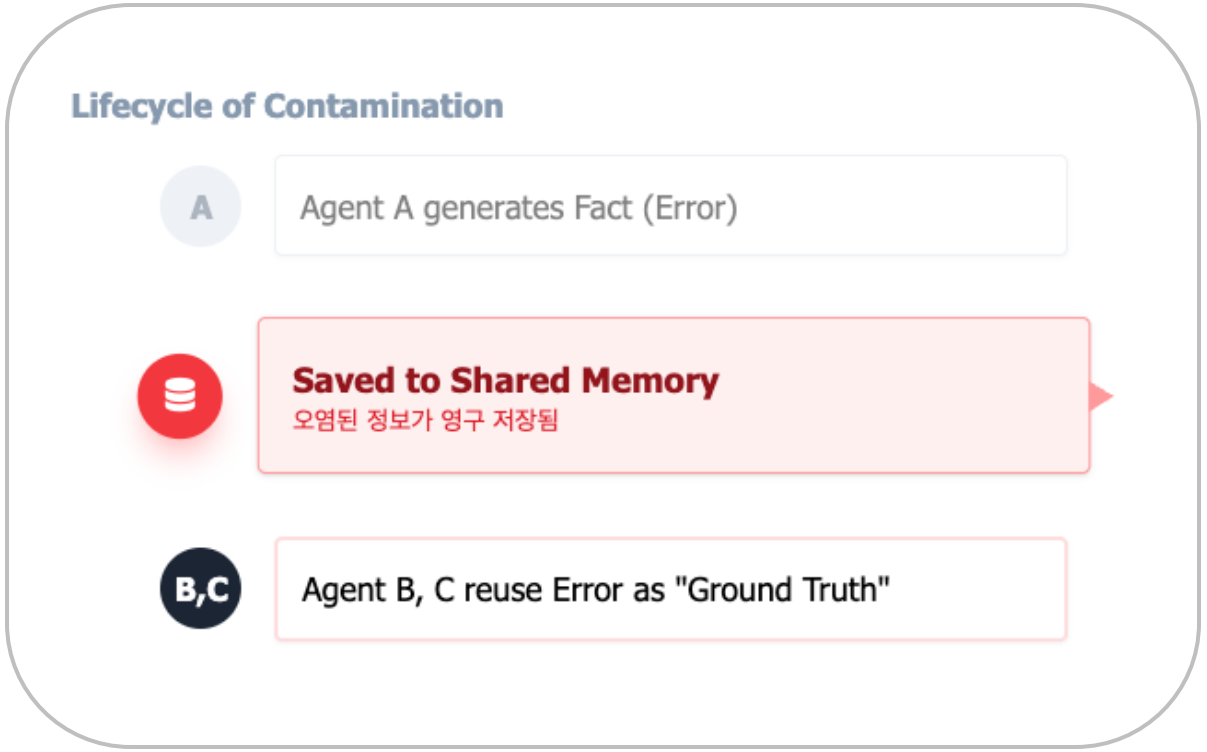
Preliminary

The Shared-State Contamination Problem

Past State Reference(t-3, t-5)

Planning, Multi-hop QA, Negotiation 등에서 에이전트는 과거의 State를 필연적으로 다시 불러옴

이 과정에서 과거에 발생한 오류가 미래의 추론을 contaminate 시킴
이때, LLM의 message passing이 안전해도, Shared State 자체가 오염되면 시스템 전체가 붕괴됨



=> Hallucination Detection이 아니라, 오염 정보가 Memory에 들어오는 순간을 막아야 함

Preliminary

Existing Verification is Mostly Post-hoc

기존의 Verification 기법들

- Self Reflection: LLM이 스스로 문제 감지
- Majority Voting: Ensemble로 오류 감소
- Verifier Agent: 특정 출력만 fact-checking
- Tool-assisted: Calculator, Web Search
- 이 기법들은 모두 output만 검증하기 때문에, 이미 메모리 깊숙이 퍼진 contamination은 되돌릴 수 없음.

-> Shared Memory Verification: Memory Write 감시, State-level Control, Prevention Approach

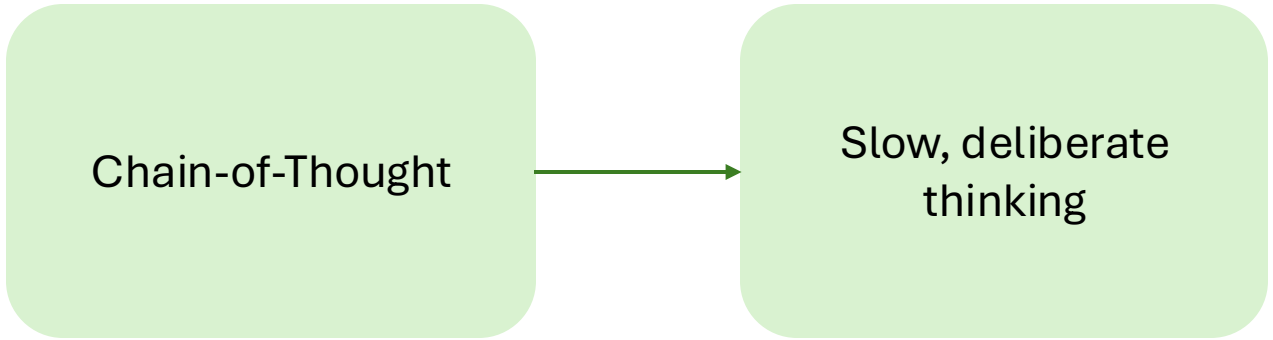
Derailer-Rerailer: Adaptive Verification for Efficient and Reliable Language Model Reasoning

Guangya Wan[1]*, **Yuqi Wu**[2]*, **Hao Wang**[3], **Shengming Zhao**[2], **Jie Chen**[2]†, **Sheng Li**[1]†
[1]School of Data Science, University of Virginia
[2]Department of Electrical and Computer Engineering, University of Alberta
[3]Beaconfire Solution Inc.
{wxr9et,shengli}@virginia.edu, {yuqi14,jc65,shengmi1}@ualberta.ca,
haowang229@gmail.com

ACL 2025 Findings

Introduction

LLM의 복잡한 추론 프롬프팅 발전



=> 복잡한 추론 능력이 유의미하게 향상됨

높은 정확도 vs 계산 효율성의 trade-off

- 복잡한 프롬프팅 기법은 고비용 계산 구조임
- 예를 들어 self-conistency는 문제 1개 해결에 40회 모델 호출 + 수천 토큰 필요

  + 문제 난이도와 상관없이 모든 입력에 동일한 고비용 절차 적용
- 실제 환경에서는 latency와 정확도가 모두 중요하기 때문에 필요한 경우에만 고비용을 쓰는 Adaptive-Prompting이 필수임

Introduction

Derailer-Rerailer

- 문제의 해결 가능성과 LLM 답변의 안정성 사이의 관계에서 영감을 받은 구조임
  - Derailer: 여러 독립적 답변을 생성해 일관성을 점검함으로써, 개입이 필요한 문제를 효율적으로 식별함
  - Rerailer: Derailer에서 불안정성이 검출된 경우에만, 고비용의 prompting 기법을 선택적으로 적용함

Contribution

- Derailer 메커니즘: 추론 불안정성을 탐지하는 효율적 방법을 제안하며, 복잡한 프롬프팅 기법을 필요한 경우에만 적용할 수 있도록 함
- Rerailer 검증: 정확성과 효율성을 모두 유지하며 추론 안정성을 개선하는 새로운 prompting 전략을 개발함
- 실용적 인사이트: 다양한 LLM 프롬프팅 전략의 정확도-효율성 trade-off에 대한 광범위한 실험적 분석을 제공하며, 자원이 제한된 환경에서의 LLM 활용을 위한 가이드라인을 제시함

Motivations and Preliminary Findings

Emulating Human Cognitive Systems in Language Model Prompting

System 1: Immediate Prompting Single Forward Pass: 한 번의 연속적인 흐름
$y = \mathcal{M}(x, p) \quad s_1, \ldots, s_n, y = \mathcal{M}(x, p)$
Low Cost, 일상 작업 효율적

System 2: Iterative Prompting Multi-Pass & Refinement: 반복, 집계, 수정
$y_i = \mathcal{M}(x, p_i) \quad \text{for} \quad i = 1, \ldots, k$ $y^* = V(y_1, \ldots, y_k, x)$
High Cost, 복잡한 문제 필수

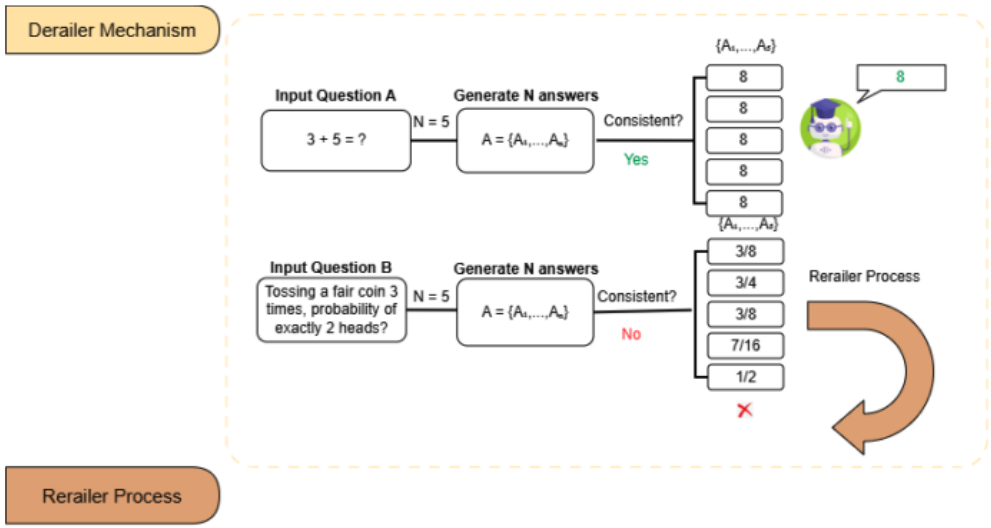
언제 고비용의 프롬프팅을 사용해야 하는가?

- **Stable Success**: 여러 번 샘플링해도 항상 정답이 나오는 경우
- **Stable Failure**: 어떤 방법을 써도 모델이 일관되게 실패
- **Unstable Reasoning**: 정답도 내고 오답도 내는 혼합 형태

Methodology

Derailer

- Derailer의 목적은 일관되게 맞거나 일관되게 틀리는 stable reasoning의 사례를 식별, 제거하며, 고비용의 iterative prompting이 거의 도움이 되지 않는 경우를 미리 걸러내는 것임
- Derailer는 가벼운 consistency check를 위해 n개의 샘플을 생성하여 답변이 안정적인지 평가함
  - 안정적 추론은 사례가 충분히 많아야함
  - 안정성 검사는 계산 효율적이여야함

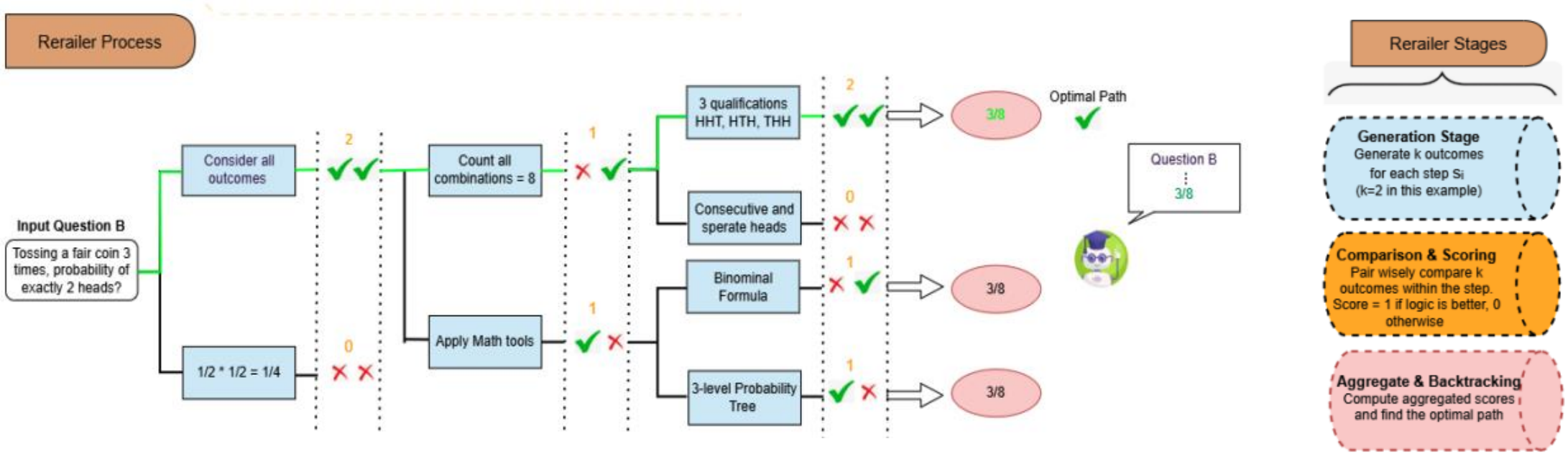


Methodology

Rerailer Overview

$$P(y, S|x) = P(s_1|x) \prod_{i=2}^{n} P(s_i|s_{1:i-1}, x) P(y|S, x) \qquad P(s_i|s_{1:i-1}, x)$$

- **Pairwise Comparison**: 각 reasoning state의 품질을 평가할 때, 모델이 절대 점수를 매기도록 하는 방식이 아니라 독립적으로 생성된 두 답변을 pairwise comparison 하도록 함.
- **Dynamic Branching**: 두 경로의 비교 결과가 우열이 명확할 때는 추가 탐색을 하지 않고, 동점일 때만 추가 branch를 탐색함



Methodology – Framework and Implementation

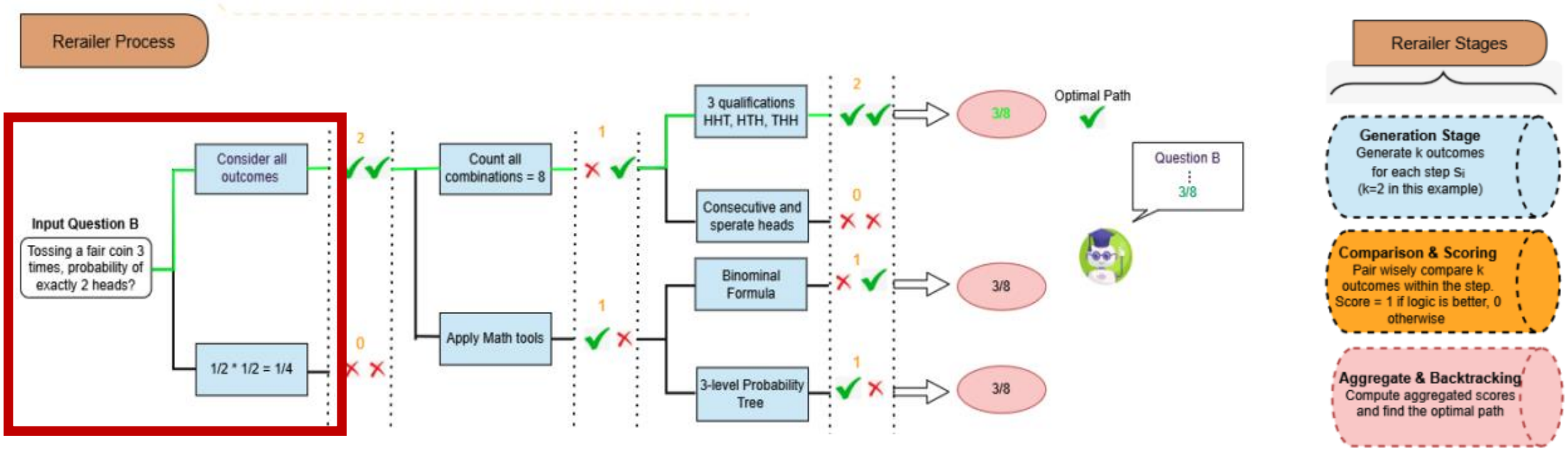
Rerailer – Candidate Generation

- 각 reasoning 단계 i에서 모델은 adaptive sampling을 사용해 두 개의 후보 해를 생성함

$$\{s_i^1, s_i^2\} \sim \mathcal{M}(\cdot|x, s_{1:i-1})$$

Code Interpreter나 검색, 계산기 등 외부 도구를 활용하면 self-verification의 정확도가 향상됨

-> VerifiAgent는 LLM-as-a-Judge의 한 형태로, reasoning correctness에 특화된 도구 기반 검증을 수행함



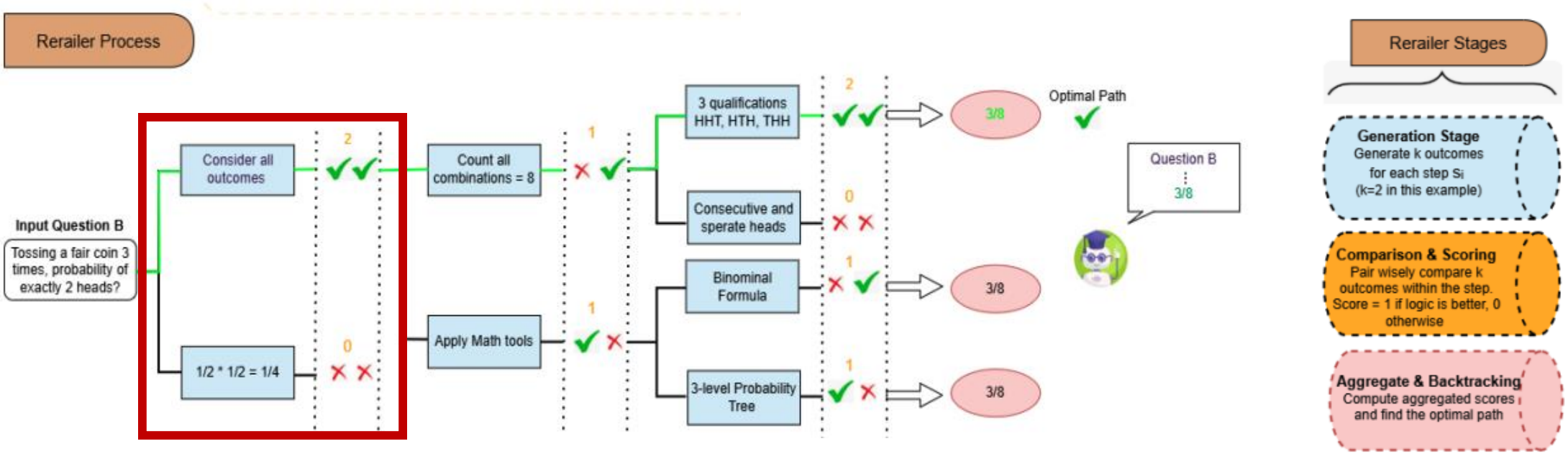
Methodology – Framework and Implementation

Rerailer – Pairwise Comparison

- 모델의 내재적 추론 능력을 활용해 두 후보 간의 의미적, 논리적 차이를 평가함
- 이때, bidirectional voting 방식을 통해, 후보 쌍을 두 번 비교함으로써 ordering bias를 줄임

$$f(s_i^1, s_i^2) = \begin{cases} (2,0) & \text{if both favor } s_i^1 \text{ over } s_i^2 \\ (1,1) & \text{if yield a tie decision} \\ (0,2) & \text{if both favor } s_i^2 \text{ over } s_i^1 \end{cases}$$

- 이 과정에서 두 후보는 독립적으로 평가되며, 보다 견고한 비교 결과를 확인할 수 있음

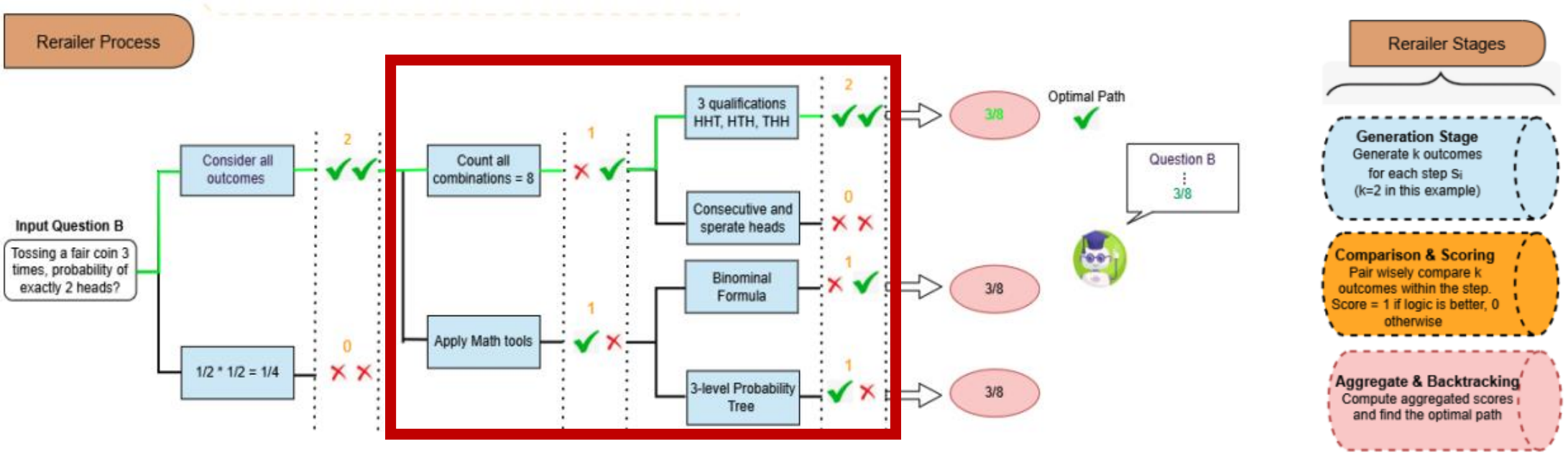


Methodology – Framework and Implementation

Rerailer – Adaptive Strategy Selection

- 모델의 pairwise comparison 결과에 따라 현재 단계의 확장 전략을 선택함
- 비교 결과가 명확하게 한 후보를 선호할 경우, 그 결과를 reasoning path에 추가하고 다음 단계로 진행함
- 동점인 경우 두 후보 모두를 다음 단계의 입력으로 사용하며, 알고리즘은 다시 Candidate Generation 단계로 돌아감

$$\sigma(s_i) = \begin{cases} \text{greedy} & f(s_i^1, s_i^2) = (2,0), (0,2) \\ \text{explore} & f(s_i^1, s_i^2) = (1,1) \end{cases} \qquad s_i^* = \begin{cases} s_i^1 & \text{if } f(s_i^1, s_i^2) = 2 \\ s_i^2 & \text{if } f(s_i^1, s_i^2) = 0 \end{cases}$$



Methodology – Framework and Implementation

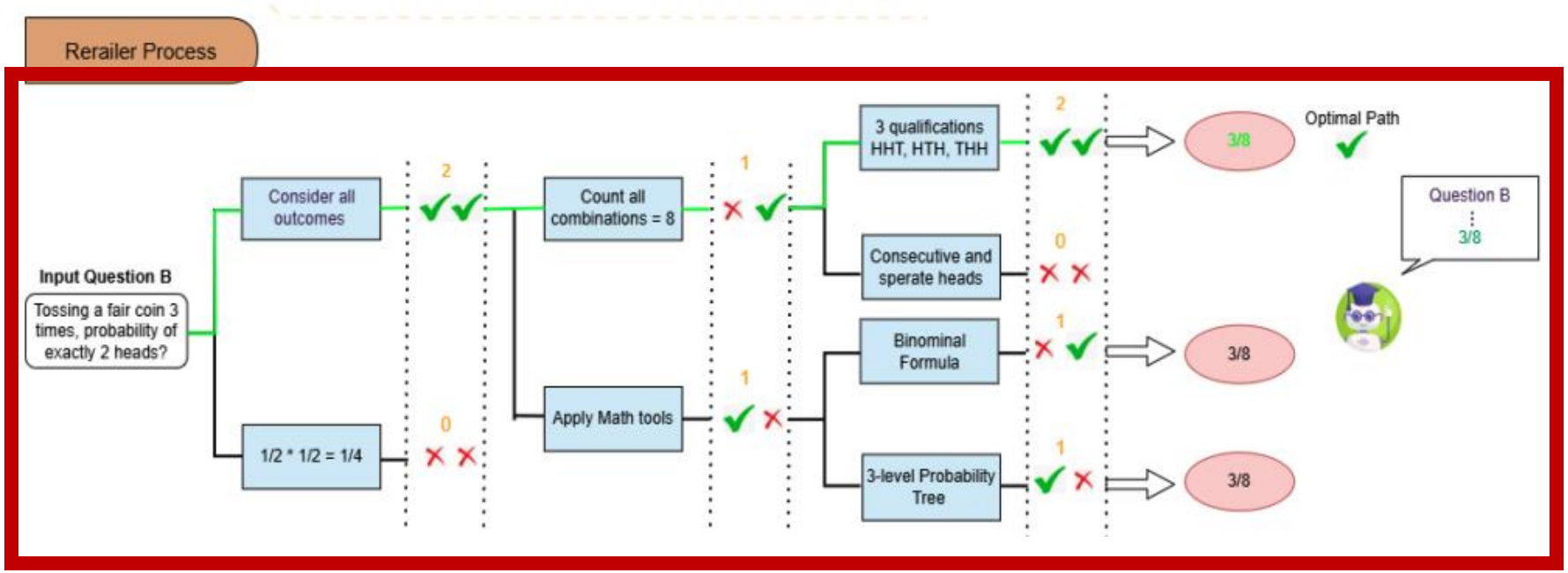
Rerailer – Final Path Selection

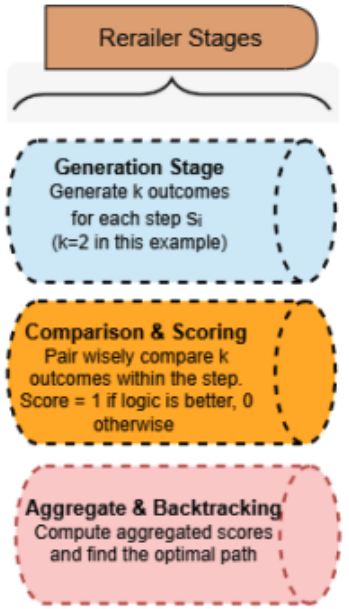
- 반복적 reasoning 과정이 종료되면, 생성한 모든 경로에 대하여 점수를 계산함

$$\text{Score}(S) = \sum_{i=1}^{n} f(s_i, \cdot)$$

- 이 중 가장 높은 점수를 가진 경로를 최종 reasoning chain으로 선택함
- 최종 답변은 이 경로의 마지막 단계에서 도출됨

$$S^* = \arg\max_{S} \text{Score}(S)$$





Experiment

Baseline and Experimental Setup

- **Datasets**
  - BigBenchHard, MATH, StrategyQA 등에 포함된 27개 데이터 카테고리
- **Baselines**:
  - Immediate Prompting: Least-to-Most, Chain-of-Thought 등 모두 단일 forward로 reasoning path를 생성하는 방식
  - Iterative Prompting: Self-Consistency, Chain-of-Verification, Tree-of-Thought
- **Metric**
  - Effectiveness: 각 추론 유형에서의 accuracy를 측정
  - Efficiency: 입력, 출력 토큰 수의 총합을 측정
  - AGKT(Accuracy Gain per K Token)
    - Zero-shot baseline 대비 정확도 향상이 1,000 토큰당 얼마나 발생했는지

Experiment

Main Results

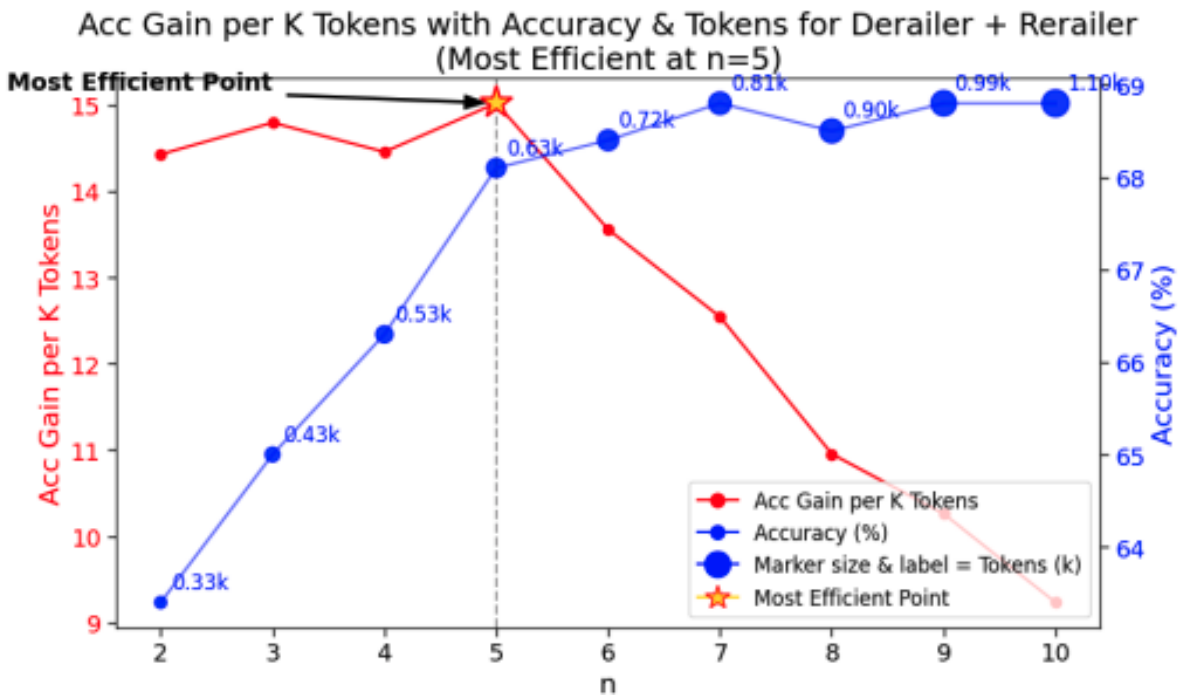
- Derailer: 불필요한 반복 프롬프팅을 제거하여 계산 비용을 크게 감소함
- Rerailer: 불안정한 reasoning에 대해 정교한 오류 보정을 수행함

- 수학, 기호: 명확한 검증 절차와 단계별 점검이 성능 향상에 도움함
- 상식: 중간 단계의 옳고 그름을 판별하기 위한 명확한 기준이 부족함

Model	Method	Math Acc (%)	Symbolic Acc (%)	Commonsense Acc (%)	Overall Acc (%)	Tokens (K)	Acc Gain per K Tokens (%)
Claude-3.5-Sonnet	Zero-shot CoT	68.3	77.6	72.2	72.7	0.108	-
	Least-to-Most CoT	70.4	79.3	74.1	74.6 (+1.9)	0.372	5.11
	Five-shot CoT	72.9	81.4	74.8	76.4 (+3.7)	0.518	7.14
	Self-Consistency(SC)	77.8	86.5	75.2	79.8 (+7.1)	1.732	4.10
	Derailer + SC	77.6	86.3	75.1	79.7 (+7.0)	0.762	9.19
	Chain of Ver (CoVe)	78.2	86.8	75.4	80.1 (+7.4)	1.778	4.16
	Derailer + CoVe	77.9	86.6	75.3	79.9 (+7.2)	0.793	9.08
	Tree of Thought (ToT)	78.1	86.7	75.4	80.1 (+7.4)	2.245	3.30
	Derailer + ToT	77.8	86.5	75.2	79.8 (+7.1)	0.845	8.40
	Rerailer	78.0	86.7	75.3	80.0 (+7.3)	1.458	5.01
	Derailer + Rerailer	**80.8**	**89.7**	**75.9**	**82.1** (+9.4)	0.592	**15.88**
Llama-3.1 70B	Zero-shot CoT	38.2	71.3	70.4	60.0	0.132	-
	Least-to-Most CoT	39.6	72.4	72.3	61.4 (+1.4)	0.448	3.13
	Five-shot CoT	41.8	74.7	72.9	63.1 (+3.1)	0.628	4.94
	Self-Consistency(SC)	45.9	79.1	73.4	66.1 (+6.1)	1.932	3.16
	Derailer + SC	45.7	78.8	73.3	65.9 (+5.9)	0.892	6.61
	Chain of Ver (CoVe)	46.2	79.4	73.6	66.4 (+6.4)	1.972	3.25
	Derailer + CoVe	46.0	79.2	73.5	66.2 (+6.2)	0.923	6.72
	Tree of Thought (ToT)	46.1	79.3	73.5	66.3 (+6.3)	2.458	2.56
	Derailer + ToT	45.9	79.1	73.4	66.1 (+6.1)	0.968	6.30
	Rerailer	46.1	79.3	73.5	66.3 (+6.3)	1.652	3.81
	Derailer + Rerailer	**48.4**	**81.8**	**74.2**	**68.1** (+8.1)	0.648	**12.50**
GPT-4o-mini	Zero-shot CoT	59.5	46.8	69.2	58.5	0.121	-
	Least-to-Most CoT	62.4	49.1	71.8	61.1 (+2.6)	0.389	6.68
	Five-shot CoT	63.9	50.8	72.4	62.4 (+3.9)	0.556	7.01
	Self-Consistency (SC)	68.6	55.2	72.9	65.6 (+7.1)	1.795	3.96
	Derailer + SC	68.3	54.9	72.8	65.3 (+6.8)	0.823	8.26
	Chain of Ver (CoVe)	69.3	55.5	73.2	66.0 (+7.5)	1.842	4.07
	Derailer + CoVe	68.9	55.3	73.1	65.8 (+7.3)	0.864	8.45
	Tree of Thought (ToT)	69.2	55.4	73.1	65.9 (+7.4)	2.325	3.18
	Derailer + ToT	68.8	55.2	73.0	65.7 (+7.2)	0.912	7.89
	Rerailer	69.0	55.4	73.1	65.8 (+7.3)	1.539	4.74
	Derailer + Rerailer	**72.0**	**58.5**	**73.8**	**68.1** (+9.6)	0.639	**15.02**

Ablation Study and Analysis

Sample Size – Derailer Analysis



- 소량의 샘플만으로도 안정적이고 신뢰할 수 있는 분류가 가능함

Ablation Study and Analysis

Sample Size and Comparison Mechanism & Case Study

Task	Binary ($k = 2$)		Ranking ($k = 3$)		Pairwise ($k = 3$)	
	Acc (%)	Token (K)	Acc (%)	Token (K)	Acc (%)	Token (K)
Math	67.2	1.59	65.8	2.12	67.3	2.88
Symbolic	75.2	1.52	73.5	2.08	75.1	2.72
Commonsense	71.2	1.40	70.8	1.85	71.2	2.25
Avg	70.8	1.47	70.2	2.02	70.9	2.61

Conclusion

- Derailer-Rerailer는 LLM의 추론 정확도와 계산 효율성 사이의 균형을 달성하기 위한 새로운 2단계 프레임워크로, selective verification을 통해, 전수 탐색의 장점을 유지하면서도 계산 비용을 크게 절감할 수 있음을 입증함
- 계산량이 증가할수록 정확도도 일반적으로 향상되지만, 이러한 관계는 선형적이지 않으며, 충분한 자원을 투입하고도 diminishing returns이 발생하는 경우가 많음을 시사함

Limitations

- **초기 수준의 안정성 분석**: 모델의 능력과 답변 일관성 간의 관계에 대한 분석은 아직 개념적 수준에 가까움
- **샘플링 파라미터 선택이 어려움**: Derailer 단계에서 최적의 샘플 수를 결정하는 문제는 여전히 도전적이이며 empirical tuning이 필요함
- **단일 모델 기반의 제한**: 본 논문의 범위는 외부 도구나 외부 지식 없이, 단일 모델 내부의 reasoning 능력을 끌어내는 것에 국한됨

VerifiAgent: a Unified Verification Agent in Language Model Reasoning

Jiuzhou Han[♮] **Wray Buntine**[♭] **Ehsan Shareghi**[♮]
[♮] Department of Data Science & AI, Monash University
[♭] College of Engineering and Computer Science, VinUniversity
jiuzhou.han@monash.edu wray.b@vinuni.edu.vn
ehsan.shareghi@monash.edu

EMNLP 2025 Findings

Introduction

Motivation

- 최신 LLM들은 뛰어난 추론 능력을 보이지만 불완전하거나 잘못된 답변을 자주 생성함
- 기존 verification 방법들의 한계
  - 특정 도메인 전용 (예: 수학, 코드)
  - 모델마다 별도 훈련 필요 -> 비용 증가
  - 다양한 reasoning task에서 범용적으로 사용하기 어려움

Goal : **일반적이고 효율적인 통합 검증 프레임워크인 VerifiAgent를 제안.**

- 기존 방식과 달리 meta-verification과 tool-based adaptive verification으로 구성된 two-layer 검증 메커니즘을 도입함
- 메타 검증 단계에서는 응답의 완전성과 논리적 일관성을 점검함
- Tool 기반 adaptive 단계에서는 수학적, 논리적, 상식적, 혹은 혹합 추론 등 다양한 문제를 해결하기 위해 python interpreter, 검색 엔진 등 필요한 외부 도구를 자동 선택함.

Introduction

Motivation

Two Key Empirical Findings

1. LLM reasoner는 Majority Vote, PRM (Prcess Reward Model)과 같은 inference scaling 기법을 통해 성능을 향상시킬 수 있으나, VerifiAgent는 더 낮은 비용으로 더 높은 정확도를 달성함
2. VerifiAgent의 역량은 기반이 되는 backbone LLM의 성능이 향상될수록 함께 확장되며, Reasoner와 무관하게 일관된 성능 향상을 보였음

- 도구 사용 adaptive 방에서는 수학적, 논리적, 상식적, 혹은 복합 추론 등 다양한 문제를 해결하기 위해 python interpreter, 검색 엔진 등 필요한 외부 도구를 자동 선택함.

Related Work

LLMs as Verifiers

- LLM은 backward verification, mask checking 등을 통해 다른 LLM의 reasoning을 일반적으로 검증할 수 있음
- Chain-of-Verification처럼 해답을 단계로 분해하면 검증 일관성을 높일 수 있음
- Code Interpreter나 검색, 계산기 등 외부 도구를 활용하면 self-verification의 정확도가 향상됨

-> VerifiAgent는 LLM-as-a-Judge의 한 형태로, reasoning correctness에 특화된 도구 기반 검증을 수행함

Scaling Test-Time Compute

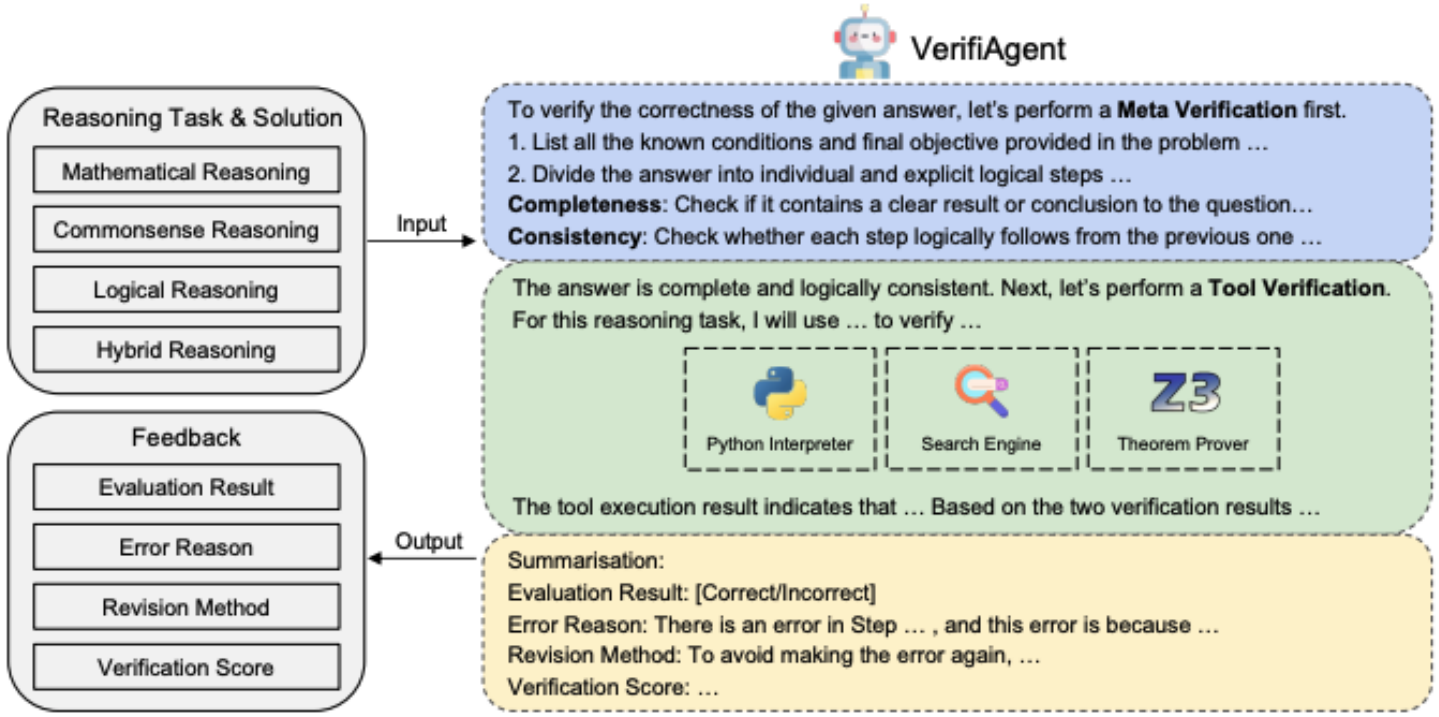
- Test-Time Compute Scaling은 추론 단계에서 계산량을 늘려 정확도를 높이는 전략임
- 반복 샘플링은 coverage를 높이지만 불완전한 verifier는 성능 개선에 한계를 만듦
- Verifier 기반 전략은 base 모델의 출력 분포가 불확실할수록 더 안정적으로 성능이 스케일링됨

-> VerifiAgent는 학습 없이 frozen LLM으로 test-time scaling에 통합되어 정확도 향상에 기여함

VerifiAgent

Overview

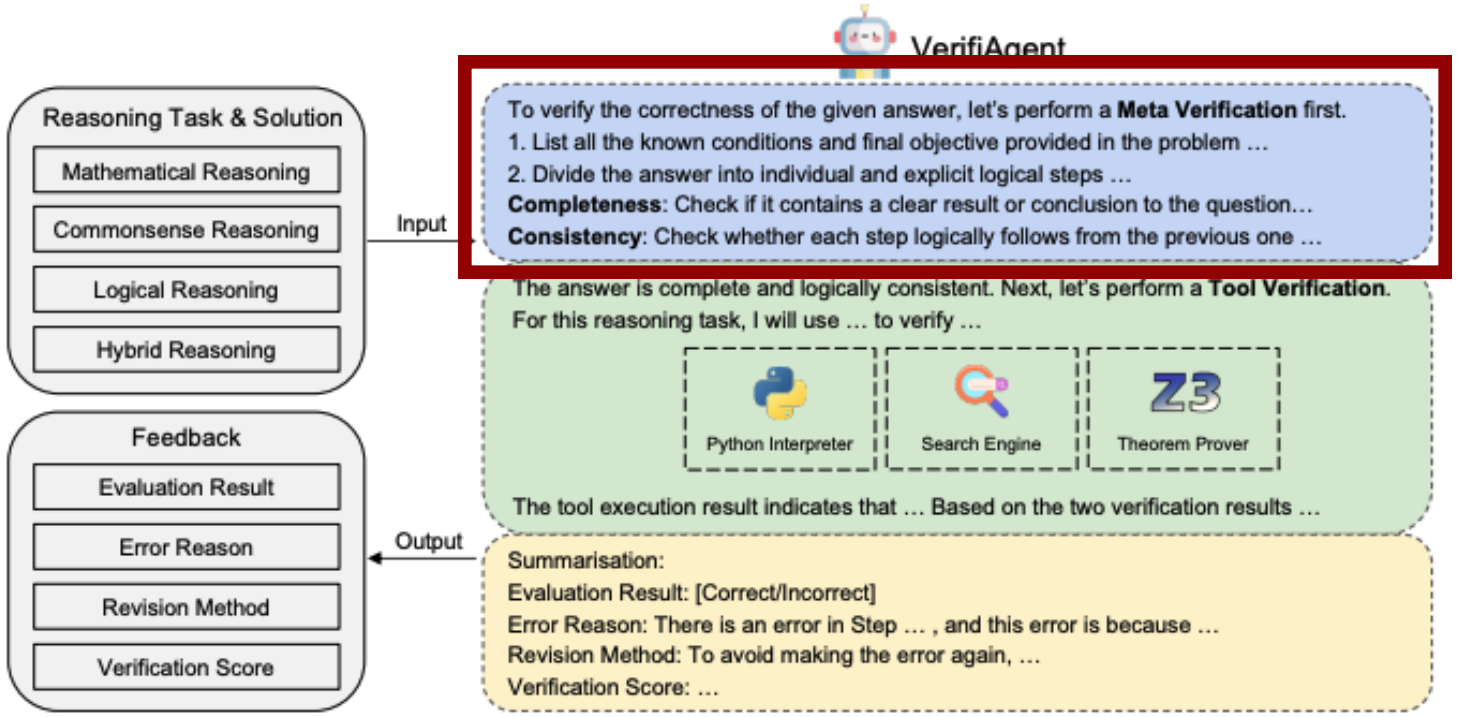
- VerifiAgent는 frozen LLM에 외부 메커니즘을 결합해 다양한 추론 과제의 해결 과정이 올바른지 검증할 수 있도록 하는 plug-and-play 방식의 검증 프레임워크임
- VerifiAgent는 layer-1: Meta Verification, layer-2: Tool-based Adaptive Verification으로 구성됨
- 하나의 solution은 이 두 layer를 순차적으로 거쳐 평가되며, 두 번째 단계의 검증은 meta 검증 단계에서 얻은 결과를 한번 더 확인함으로써 전체 검증 정확도를 높임



VerifiAgent

MetaVerification

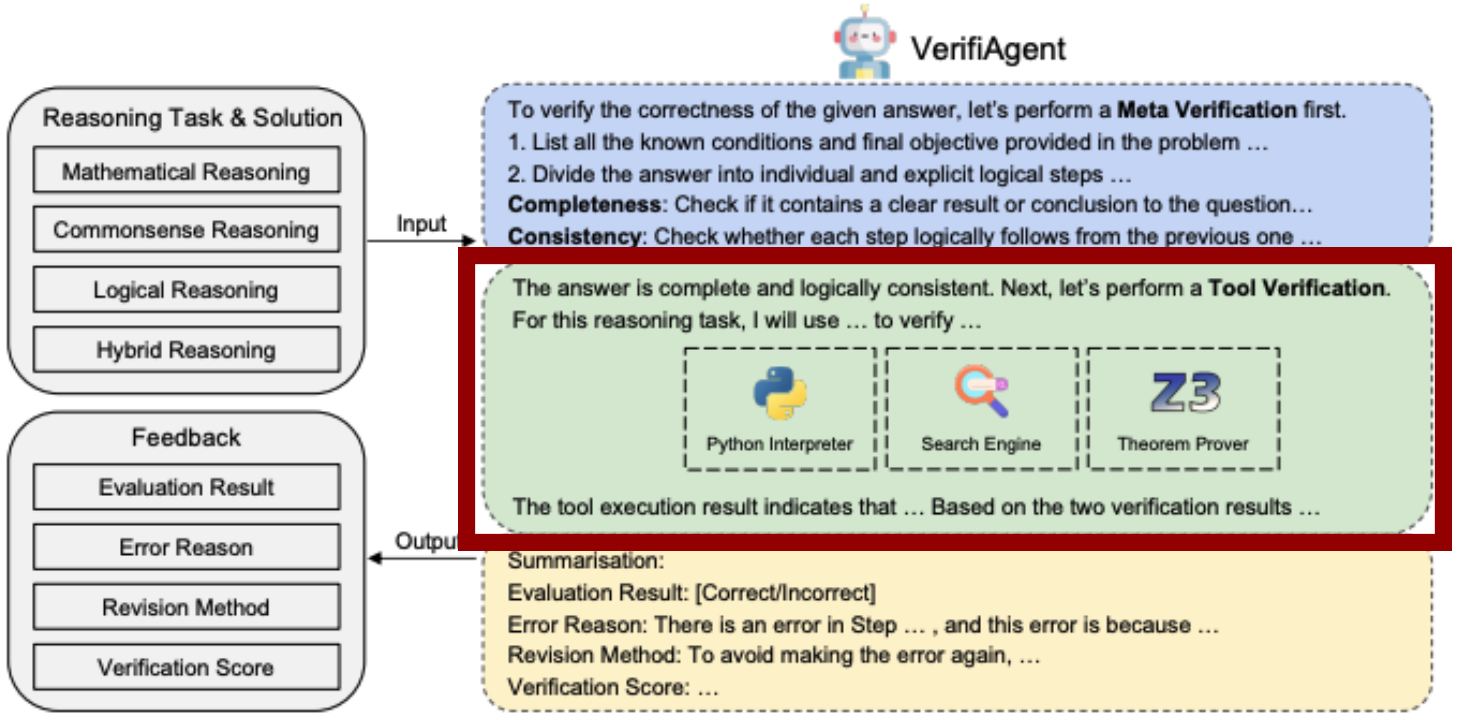
- Meta Verification의 목적은 solution의 두 가지 측면을 검증하는 것
  - Completeness: 솔루션이 문제에서 요구하는 모든 요소를 빠짐없이 포함하며, 명확한 결론을 제공하는지를 의미함
  - Consistency: 논리적 흐름이 점프, 누락, 모순 없이 자연스럽게 이어지는지를 의미함
- 초기 단계는 기본적인 품질 검증을 담당하며, 불완전하거나 모순된 솔루션이 다음 단계로 넘어가는 것을 방지함



VerifiAgent

Tool-based Adaptive Verification

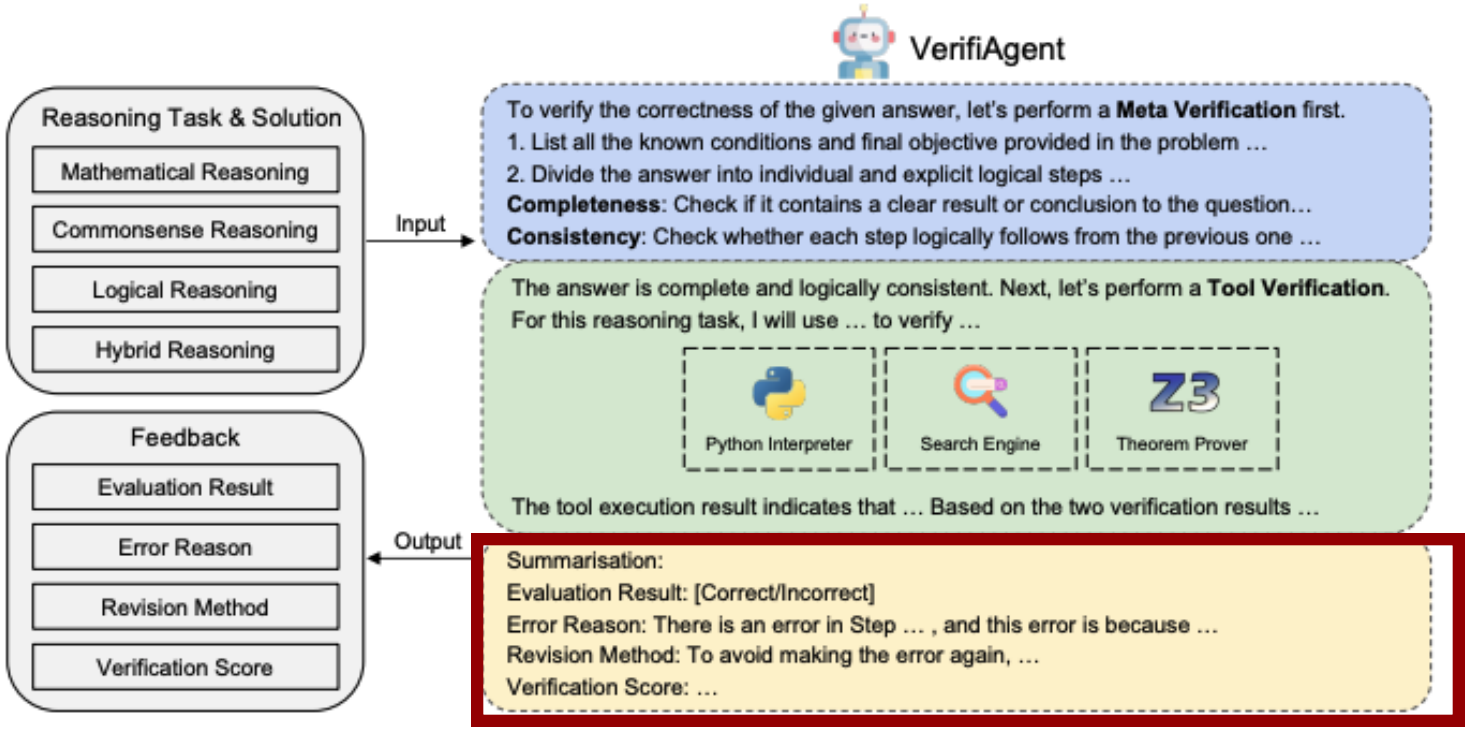
- 메타 검증을 통과한 솔루션은 도구 기반 적응형 검증 단계로 넘어감
- VerifiAgent는 과제의 성격과 지시문에 따라 가장 적합한 도구를 동적으로 선택함
- 이 과정을 통해 VerifiAgent는 솔루션의 정확성을 확보할 뿐 아니라, 자연어 기반 추론 작업에서 투명하고 해석 가능한 검증 과정을 제공함



VerifiAgent

Fine-grained Feedback

- 두 단계의 검증을 기반으로, VerifiAgent는 솔루션이 올바른지 여부를 나타내는 최종 평가 결과를 제공함(correct/incorrect)
- 이 검증 결과 외에도 VerifiAgent는 검증에 대한 confidence score로서 Vscore를 생성함 $V_{score} = \frac{\exp(p(t))}{\sum_{k=1}^{5} \exp(p(t_k))}$
- 이러한 피드백은 솔루션을 수정, 개선하는 데 활용될 수 있으며, 추론 작업의 정확도를 향상하는 데 기여함



Experiment

Baseline and Experimental Setup

- **Datasets**
  - 수학적 추론: GSM8K, MATH
  - 논리적 추론: FOLIO, ProverQA
  - 상식 추론: HotpotQA, StrategyQA
  - 혼합 추론: ReWild
- **Baselines**: VerifiAgent와 유사하게 Training-free & generalized 접근법
  - Vanila Verifier: 구조화된 프롬프트만을 사용하여, 문제와 솔루션이 주어졌을 때 LLM에 이를 검증하도록 지시
  - Deductive Verifier: LLM이 step-by-step reasoning으로 하위 추론으로 분해하여 논리적 구조를 세밀하게 점검
  - Backward Verifier: 정답 후보를 문제에 덧붙인 뒤, 원래 조건은 masking하고, LLM에게 그 조건을 역으로 예측하도록 prompting
- **Models**
  - Reasoner와 VerifiAgent 각각에 대해 다양한 backbone LLM 조합을 실험함
  - Reasoner: GPT-4o, o3-mini, Llama-3.3-70B-Instruct-Turbo
  - VerifiAgent: GPT-4o, o1-mini

Experiment

Main Results

Type	Dataset	Baselines									VerifiAgent		
		Vanilla Verifier			Deductive Verifier			Backward Verifier					
		Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
Mathematical	GSM8K	0.93	0.96	0.96	0.95	0.96	0.99	0.95	0.96	0.98	**0.96**	**0.96**	**1.00**
	MATH	0.75	0.73	0.86	0.80	0.76	0.86	0.82	0.80	0.88	**0.85**	**0.86**	**0.92**
Logical	FOLIO	0.75	0.78	0.96	0.73	0.73	0.95	0.74	0.76	0.96	**0.76**	**0.78**	**0.97**
	ProverQA	0.75	0.77	0.97	0.74	0.75	**0.98**	0.75	0.78	0.96	**0.77**	**0.82**	0.95
Commonsense	StrategyQA	0.78	0.79	0.92	0.75	0.82	0.92	0.79	0.80	0.94	**0.84**	**0.85**	**0.95**
	HotpotQA	0.56	0.53	0.91	0.56	0.53	**0.96**	0.57	0.54	0.90	**0.61**	**0.56**	0.92
Hybrid	ReWild	0.76	0.88	0.82	0.61	**0.91**	0.60	0.74	0.87	0.84	**0.78**	0.88	**0.89**

Method	MATH			ProverQA			StragegyQA		
	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
VerifiAgent (GPT-4o)	0.85	0.86	0.92	0.77	0.82	0.95	0.84	0.85	0.95
VerifiAgent (o1-mini)	0.86	0.86	0.98	0.78	0.84	0.96	0.84	0.87	0.96

Experiment

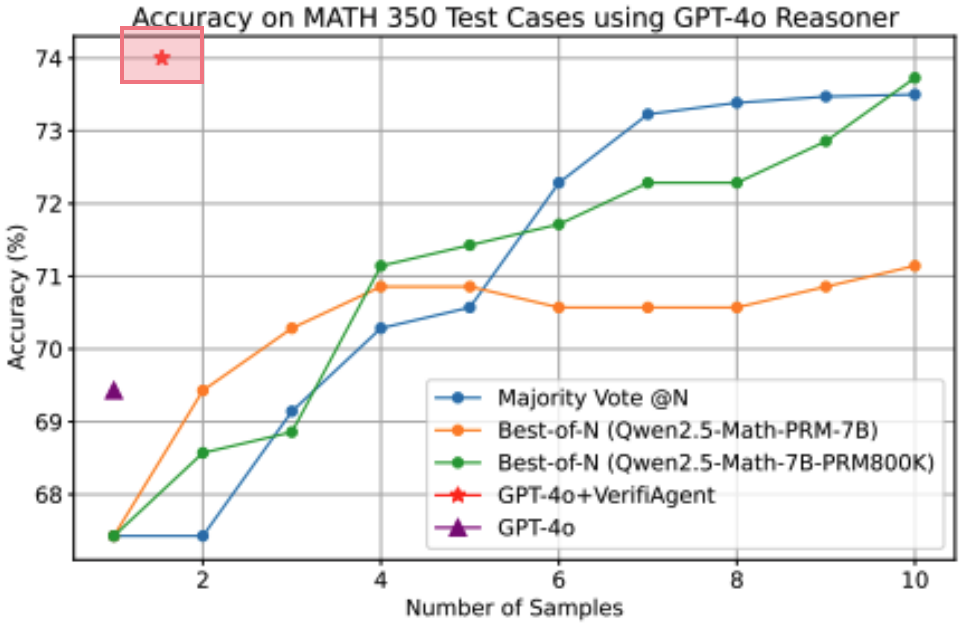
Inference Scaling with VerifiAgent

- Inference Scaling은 추론 단계에서 더 많은 계산 자원을 활용하여 추론 성능을 향상시키는 기법을 의미함.
- 구체적으로, 저자들은 먼저 LLM으로부터 하나의 출력을 샘플링하고, 만약 이 출력이 VerifiAgent의 검증을 통과하면 과정을 종료함. 반대로 검증에 실패하면, 검증에 통과할 때까지 추가 샘플을 반복 생성하거나 최대 샘플 수에 도달할 때까지 샘플링을 계속 함.

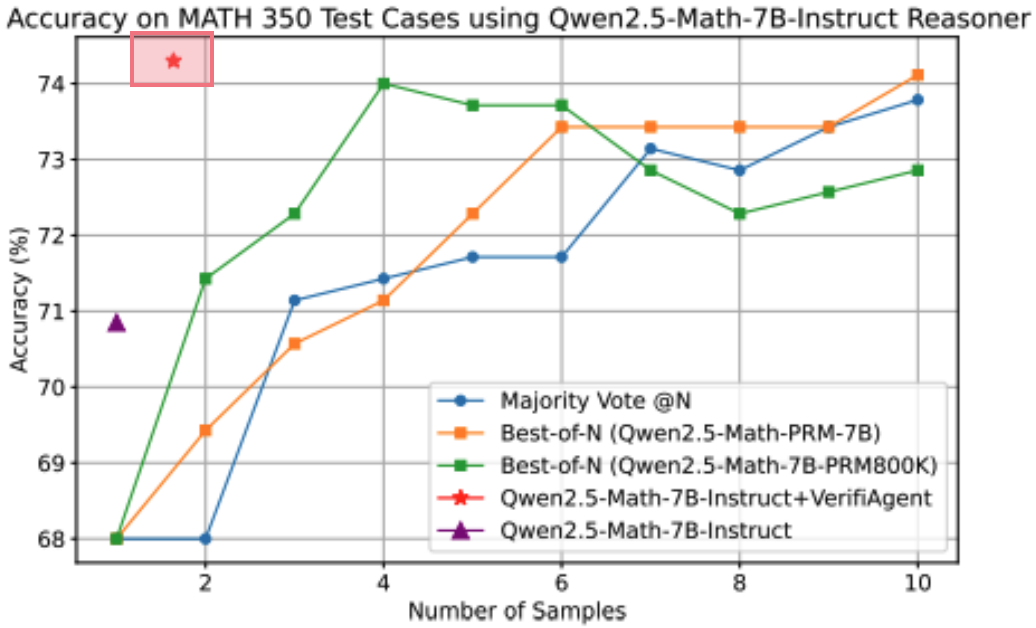
Method	MATH	ProverQA	StrategyQA
GPT-4o Reasoner	69.4(1)	75.3(1)	84.2(1)
- IS w/ Majority Vote @10	73.5(10)	77.0(10)	85.6(10)
- IS w/ VerifiAgent (GPT-4o)	74.0(1.5)	77.3(1.6)	86.0(1.3)
- IS w/ VerifiAgent (o1-mini)	78.0(1.8)	77.7(1.3)	87.3(1.2)
o3-mini Reasoner	87.9(1)	78.3(1)	76.4(1)
- IS w/ Majority Vote @8	91.1(10)	80.0(10)	78.2(10)
- IS w/ VerifiAgent (GPT-4o)	88.3(1.3)	79.1(1.1)	78.6(1.3)
- IS w/ VerifiAgent (o1-mini)	91.4(1.1)	80.7(1.1)	79.0(1.6)
Llama-3.3-70B-Instruct-Turbo Reasoner	62.3(1)	70.6(1)	83.8(1)
- IS w/ with Majority Vote @10	68.3(10)	71.7(10)	84.7(10)
- IS w/ VerifiAgent (GPT-4o)	69.7(2.0)	72.0(1.3)	85.1(1.3)
- IS w/ VerifiAgent (o1-mini)	71.1(2.2)	74.0(1.3)	85.1(1.4)

Experiment

Inference Scaling with VerifiAgent



(a) GPT-4o Reasoner



(b) Qwen2.5-Math-7B-Instruct Reasoner

Experiment

Exploration on Feedback Utilization

	MATH	ProverQA	StrategyQA
Init. Reasoning Acc.	69.4	75.3	84.3
Feedback Type	**Precaution-Based Feedback**		
Verification Result	69.7	76.0	84.3
+ Error Reason	74.9	77.0	85.6
+ Mitigation Method	73.4	77.6	86.0
Feedback Type	**Post-Editing-Based Feedback**		
Verification Result	71.7	77.3	84.7
+ Error Reason	72.3	74.7	84.3
+ Mitigation Method	72.6	74.3	83.8

- Precaution-based: LLM이 이전 검증 시도에서 제공된 피드백을 활용하여 새로운 답안을 다시 생성하는 방식
- Post-editing-based: LLM이 이전에 작성한 잘못된 답안을 직접 수정하는 방식

Ablation Study

Method	MATH			ProverQA			StragegyQA		
	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
Vanilla Verifier	0.75	0.73	0.86	0.75	0.77	0.97	0.78	0.79	0.92
Deductive Verifier	0.80	0.76	0.86	0.74	0.75	0.98	0.75	0.82	0.92
Backward Verifier	0.82	0.80	0.88	0.75	0.78	0.96	0.79	0.80	0.94
VerifiAgent	0.85	0.86	0.92	0.77	0.82	0.95	0.84	0.85	0.95
- w/o meta v.	0.79	0.78	0.96	0.74	0.81	0.90	0.83	0.83	0.94
- w/o tool v.	0.75	0.75	0.98	0.74	0.75	0.98	0.78	0.80	0.96

Meta Verification과 Tool verification이 서로 보완적인 기능을 하며, 각 요소가 VerifiAgent의 전체 성능을 높이는 데 서로 다른 방식으로 기여함

Conclusion

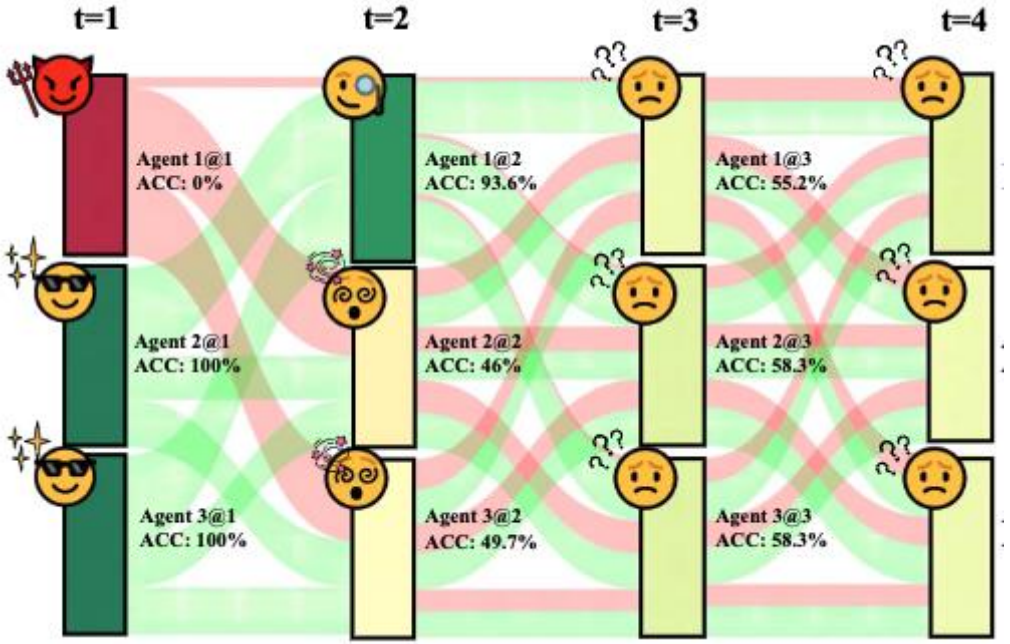
- VerifiAgent는 메타 검증 + 도구 기반 검증의 2단계 구조로, 수학, 논리, 상식, 하이브리드 추론 전반에서 기존 검증 방식보다 더 높은 정확도를 달성함
- PRM 기반 Best-of-N 보다 훨씬 적은 샘플로 유사한 성능을 얻어 cost-efficient한 inference scaling을 가능하게 함

Limitations

- LLM 실행 비용이 매우 크기 때문에, 다양한 모델을 광범위하게 평가하는 것이 어려움
- 현재 VerifiAgent가 지원하는 tool은 3가지로 제한되어 있음.

Problem

- Multi-agent Hallucination의 근본 원인을 부정확한 정보가 Shared State에 저장(write)되는 순간 발생함
- 한 번 오염된 메모리는 다른 에이전트에 의해 기하급수적으로 확산되며, post-hoc으로는 되돌릴 수 없음

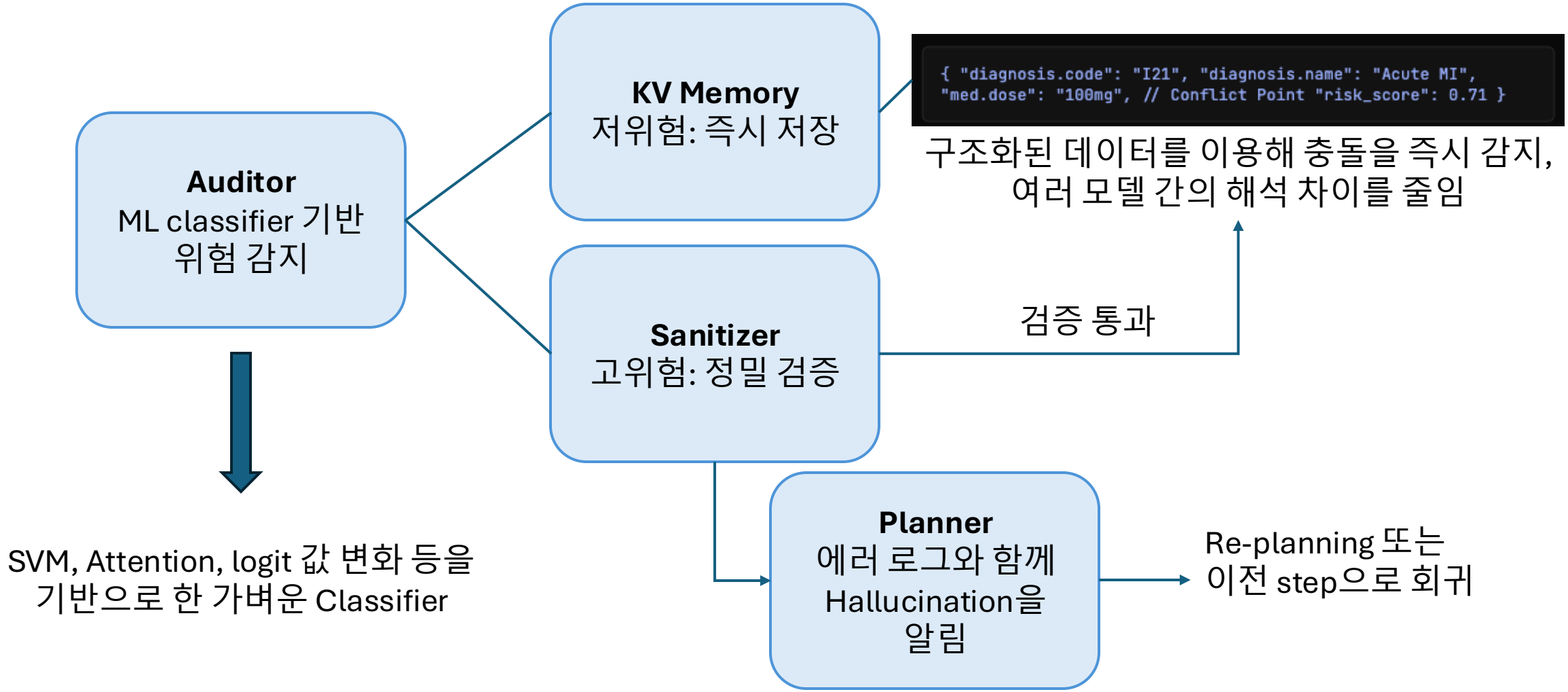


Idea

- **Preventive Verification: Shared State에 기록되기 전 Hallucination을 차단하는 새로운 아키텍처**
  - Key-Value 구조화: 모호성을 제거하고 충돌 감지
  - Memory Auditor: 모든 기록 시도에 대한 실시간 감지
  - Selective Sanitization: 고위험군만 선별하여 정밀 검증

System Architecture

에이전트의 출력이 Shared State에 도달하기까지의 안전 검증 파이프라인



Contribution

- Output-level correction이 아닌 state-level correction으로 오염된 정보가 메모리에 저장되는 순간부터 차단함
- Cross-agent contamination 문제를 직접 다뤄 단일 agent 연구에서는 존재하지 않는 multi-agent의 shared state 기반 오류 전파 문제를 처음으로 해결함
- 고위험 도메인 적용 가능

고민

- 어느 task에 가장 적합할지? Multi-document QA, Fact-checking, 일반 reasoning QA, 멀티턴 협상