Probabilistic Soundness Guarantees in LLM Reasoning Chains

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

In reasoning chains generated by large language models (LLMs), initial errors often propagate and undermine the reliability of the final conclusion. Current LLM-based error detection methods often fail to detect propagated errors because earlier errors can corrupt judgments of downstream reasoning. To better detect such errors, we introduce Autoregressive Reasoning Entailment Stability (ARES), a probabilistic framework that evaluates each reasoning step based solely on previously-verified premises. We find that ARES can reliably detect propagated reasoning errors that other baselines fail to find with probabilistic guarantees.

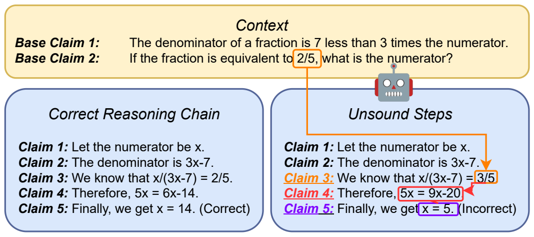
1 Introduction

Large Language Models (LLMs) often produce reasoning chains with errors that propagate, undermining the final outputs [Huang et al., 2025, Lyu et al., 2024]. An error can be *ungrounded* statements, *invalid* derivations, or *propagated* errors as illustrated in Figure 1. For example, deriving $x = 5$ from $5x = 9x - 20$ is logically valid [Lee and Hockenmaier, 2025], but can be a propagated error if the premise $5x = 9x - 20$ differs from the context [Tyagi et al., 2024].. These errors compromise the reliability LLMs in high-stakes domains [Agarwal et al., 2024, Chen and Mueller, 2023].

Current error detection methods typically aim to identify all errors at once. For example, LLM judges are prompted to evaluate the entire chain and assess each step for correctness [Tyagi et al., 2024, He et al., 2025]. Similarly, Process Reward Models (PRMs) are language models trained with step-level classification heads on this same objective [Lightman et al., 2023].

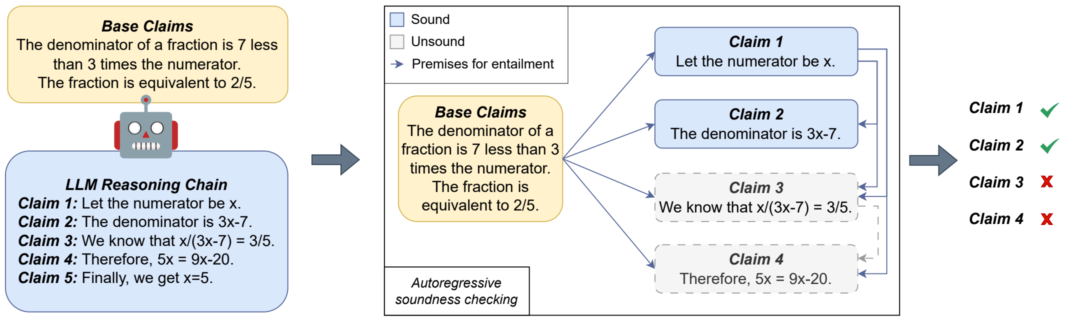
However, existing error detection methods often fall short. Specifically, they are often distracted by the presence of propagated errors [He et al., 2025, Turpin et al., 2023, Dhuliawala et al., 2023]. In the example from Figure 1, if steps 3, 4, and 5 are evaluated together, an LLM may incorrectly mark step 5 as sound by incorrectly relying on step 4, which is invalid. This highlights the need for robust methods that can assess the soundness of each step without being adversely distracted by prior errors.

To address this issue, we draw inspiration from human reasoning. Humans typically review claims sequentially, and disregard previously unsound statements when evaluating subsequent ones [Johnson-Laird, 2010, Mukherjee et al., 2025]. In contrast, LLMs struggle to ignore prior errors, which causes naive detection methods to fail at simultaneously identifying and localizing all errors in a reasoning chain [Wu et al., 2024, Song and Tavanapong, 2024]. To overcome this limitation, we develop Autoregressive Reasoning Entailment Stability (ARES), a probabilistic framework that evaluates the soundness of each reasoning step based on its expected entailment probability, conditioned only on previously-occurring, sound claims (Figure 2). We iteratively evaluate each claim as follows: *entailed* claims are retained as premises for subsequent steps, while *non-entailed* claims are discarded. For *uncertain* claims, retention is probabilistic based on the entailment model. This adaptation not



Context
Base Claim 1: The denominator of a fraction is 7 less than 3 times the numerator.
Base Claim 2: If the fraction is equivalent to 2/5, what is the numerator?
Correct Reasoning Chain
Claim 1: Let the numerator be x.
Claim 2: The denominator is 3x-7.
Claim 3: We know that x/(3x-7) = 2/5.
Claim 4: Therefore, 5x = 6x-14.
Claim 5: Finally, we get x = 14. (Correct)
Unsound Steps
Claim 1: Let the numerator be x.
Claim 2: The denominator is 3x-7.
Claim 3: We know that x/(3x-7) = 3/5
Claim 4: Therefore, 5x = 9x-20
Claim 5: Finally, we get x = 5. (Incorrect)

Figure 1: **Faulty LLM reasoning due to propagated errors from ungrounded and invalid steps.** An unsound step is a step that is either **ungrounded** (incorrect with respect to the context), **invalid** (logically incorrectly derived), or contains **propagated** errors. In this example, **Step 3 is ungrounded** because it contains information different from the base claim 2. **Step 4 is invalid** because it contains an incorrect mathematical computation. **Step 5 is a propagated error**, even though it is logically correct from Step 4. This figure is adapted from an example in Lee and Hockenmaier [2025].



Base Claims
The denominator of a fraction is 7 less than 3 times the numerator.
The fraction is equivalent to 2/5.
LLM Reasoning Chain
Claim 1: Let the numerator be x.
Claim 2: The denominator is 3x-7.
Claim 3: We know that x/(3x-7) = 3/5.
Claim 4: Therefore, 5x = 9x-20.
Claim 5: Finally, we get x=5.
Sound
Unsound
Premises for entailment
Base Claims
The denominator of a fraction is 7 less than 3 times the numerator. The fraction is equivalent to 2/5.
Claim 1
Let the numerator be x.
Claim 2
The denominator is 3x-7.
Claim 3
We know that x/(3x-7) = 3/5.
Claim 4
Therefore, 5x = 9x-20.
Autoregressive soundness checking
Claim 1 ✓
Claim 2 ✓
Claim 3 ✗
Claim 4 ✗

Figure 2: **(Autoregressive Soundness Checking)** When we verify an LLM generated reasoning chain, we can break the context and reasoning chain down to base claims and derived claims. An autoregressive soundness checker can then check each derived claim step-by-step, using only claims already identified to be sound as the premise.

only improves error detection but also enables us to give certified guarantees on the robustness of a reasoning chain.

Our contributions are highlighted as follows.

- We introduce Autoregressive Reasoning Entailment Stability (ARES), a novel probabilistic framework for evaluating claims in LLM reasoning chains. This framework uniquely assesses each step by conditioning only on previously verified sound claims, ensuring a robust and adaptable evaluation.
- We design a computationally and sample-efficient autoregressive algorithm for entailment estimation within this framework. Crucially, this algorithm provides sample-efficient certifications of entailment with rigorous statistical guarantees, a capability absent in prior methods.
- We demonstrate that ARES accurately certifies both sound and unsound reasoning steps, particularly excelling in long chains prone to error propagation. ARES significantly surpasses existing approaches and generalizes across diverse reasoning tasks.

2 Soundness in Reasoning Chains

We aim to identify and certify errors within LLM-generated chain-of-thought (CoT) reasoning. To this end, this section formalizes reasoning chains in terms of their constituent claims (Section 2.1), introduces the concept of probabilistic entailment between these claims (Section 2.2), and defines a notion of soundness that incorporates internal groundedness, validity, and the entailment of a final hypothesis (Section 2.3).

2.1 Claims and Sequences of Claims

A reasoning chain is conceptualized as a sequence of claims, where a claim is the assertion of a proposition. For instance, "The denominator is $3x - 7$" is a claim regarding a component of an algebraic expression, while "We know that $\frac{x}{3x-7} = \frac{2}{5}$" is a claim that synthesizes prior information about an equation. The granularity of claims is domain-dependent; it is permissible for a claim to range from an atomic statement or a single sentence (e.g., "We can simplify $\frac{x}{3x-7} = \frac{2}{5}$ to $5x = 6x - 14$.") to more extensive segments like entire theorems or proofs.

For a more formal discussion of our method, we let $\mathcal{C}$ denote the set of all possible claims, and $\mathcal{C}^\star$ represent the set of all possible sequences of claims. An example of such a sequence is as follows:

$$\big(\text{“Let the numerator be } x\text{”}, \text{“The denominator is } 3x-7\text{”}, \text{“We know that } \tfrac{x}{3x-7} = \tfrac{2}{5}\text{”}\big) \in \mathcal{C}^\star$$

which consists of the following individual claims:

$$\text{“Let the numerator be } x\text{”} \in \mathcal{C}, \text{“The denominator is } 3x-7\text{”} \in \mathcal{C}, \text{“We know that } \tfrac{x}{3x-7} = \tfrac{2}{5}\text{”} \in \mathcal{C}.$$

This distinction between individual claims and sequences of claims is important for discussing the inclusion and exclusion of items from a premise during logical entailment, which we define next.

2.2 Probabilistic Entailment of Claims

To capture the notion of logical entailment between claims expressed in natural language, we introduce probabilistic entailment models. This approach is motivated by the inherent fuzziness and ambiguity often present in natural language reasoning [Zadeh, 2008, Yu et al., 2024]. Formally, a probabilistic entailment model $\mathcal{E} : \mathcal{C}^\star \times \mathcal{C} \to [0, 1]$ accepts a sequence of claims as a premise, $P \in \mathcal{C}^\star$, and a single claim as a hypothesis, $H \in \mathcal{C}$. It then returns a scalar value representing the probability that the premise P entails the hypothesis H. For instance, consider the premise and hypothesis pair

$$P = \big(\text{“Sarah put on her running shoes.”}, \text{“She stretched by the sidewalk.”}, \text{“The sun was setting.”}\big)$$
$$H = \text{“Sarah is going for an evening run.”}$$

A probabilistic entailment model might output $\mathcal{E}(P, H) = 0.85$. This score reflects the linguistic and social ambiguity in inferring the certainty of an "evening run" from the actions of "donning running shoes and stretching". Such a fuzzy, probabilistic approach generalizes classical Boolean logic, where the output is strictly 1 for entailment and 0 for non-entailment. [1]

2.3 Reasoning Chains and Soundness

To analyze the step-by-step reasoning of LLMs, particularly in CoT processes, we conceptualize the output as a *reasoning chain*. This chain initiates with a set of provided statements or contextual information, designated as *base claims*. Following these, the LLM autoregressively produces a sequence of subsequent statements, which we term *derived claims*. This entire sequence is formally represented as:

$$(C_1, \ldots, C_n, C_{n+1}, \ldots, C_{n+m}) \in \mathcal{C}^\star \tag{1}$$

where $C_1, \ldots, C_n$ are the base claims, and $C_{n+1}, \ldots, C_{n+m}$ are the derived claims.

This partition is methodologically crucial. Base claims $(C_1, \ldots, C_n)$ serve as the foundational premises for a given reasoning task; their factual accuracy is considered out of scope for the present analysis and is assumed to be handled by external mechanisms. We focus on assessing whether each derived claim (C_{n+i} for $i = 1, \ldots, m$) is soundly inferred from the set of preceding statements. We begin by defining a deterministic (i.e., "hard") version of soundness, where the entailment model $\mathcal{E}$ is assumed to be binary-valued.

Definition 2.1 (Hard Soundness)**.** Consider a reasoning chain $(C_1, \ldots, C_{n+m})$ with base claims $C_1, \ldots, C_n$ and derived claims $C_{n+1}, \ldots, C_{n+m}$. Then, this reasoning chain is *hard-sound* with respect to the deterministic entailment model $\mathcal{E}$ if

$$\mathcal{E}((C_1, \ldots, C_{n+i-1}), C_{n+i}) = 1, \tag{2}$$

for all derived claims indexed by $i = 1, \ldots, m$.

[1]We distinguish between a non-entailed claim (not logically following premises) and a provably false claim (factually incorrect). For instance, "Sarah lives in Philadelphia" is not entailed but not demonstrably false.

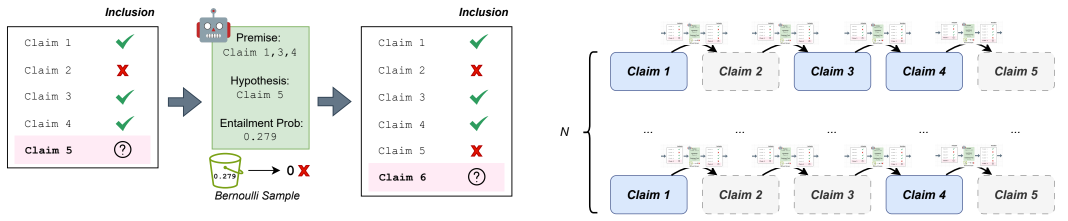


Figure 3: **(Estimating ARES)** (Left) The entailment rate of each derived claim is autoregressively computed. We first randomly initialize a premise (denoted by α) according to the base priors $p_1, \dots, p_n$. Then, for each derived claim, we compute its entailment rate with respect to the premise set. Finally, we add this derived claim to the premise set with probability equal to its entailment rate. (Right) This is run in parallel across N instances.

The concept of hard soundness provides a precise, albeit strict, benchmark for evaluating the logical integrity of a reasoning chain because it requires every derived claim to be perfectly entailed by its predecessors. However, LLM-generated reasoning chains often deviate from this ideal. Therefore, while hard soundness serves as an important theoretical standard of correctness, it cannot give nuanced measures of error, particularly in long reasoning chains. This necessitates more flexible methods for measuring claim soundness even in the presence of errors, which we address next.

3 Soundness Checks via Autoregressive Reasoning Entailment Stability

We now consider the practical certification of LLM-generated reasoning chains. These chains are formed autoregressively: starting from an initial sequence of base claims $C_1, \dots, C_n$, the LLM iteratively generates the derived claims $C_{n+1}, \dots, C_{n+m}$ where each

$$C_{n+k} = \mathsf{LLM}(C_1, \dots, C_{n+k-1}),$$

for reasoning steps $k = 1, \dots, m$. We aim to quantify the reliability of this process using a sequence of *entailment stability scores*: $\tau_1, \dots, \tau_m \in [0, 1]$, where each τ_k denotes how reliably the k-th derived claim (C_{n+k}) is entailed with respect to its preceding claims $(C_1, \dots, C_{n+k-1})$. The connection between entailment and error detection is straightforward: a claim C_{n+k} is likely to be erroneous if τ_k is low.

While the notion of reliability is general, a rigorous definition of each τ_k is challenging due to probabilistic uncertainty and unreliable preceding claims in the reasoning chain. Critically, deterministic hard soundness (Definition 2.1) cannot account for premises with uncertainty. Autoregressive Reasoning Entailment Stability addresses this: Section 3.1 motivates probabilistic entailment using insights from human psychology, LLM empirics, and mathematical logic. Subsequently, Section 3.2 formalizes our approach, defines Autoregressive Reasoning Entailment Stability, and details its efficient Monte Carlo estimation.

3.1 Entailment with Probabilistic Premises

The key challenge lies in accurately assessing entailment when premises are probabilistically uncertain. To address this, our main insight is to calculate an overall likelihood by averaging across various probable combinations of that uncertain information.

Our approach is motivated by several observations. In **human cognition**, people naturally discount or ignore dubious statements when reasoning [Johnson-Laird, 2010]. Similarly, lengthy contexts are often filtered to remove irrelevant and erroneous claims to improve **LLM performance** on reasoning tasks [Mukherjee et al., 2025]. These observations collectively motivate our development of a probabilistic entailment framework based on premise subsets.

To measure the reliability of a hypothesis H with respect to a premise P containing k claims with uncertain soundness, we consider all 2^k configurations of inclusion and exclusion for P's claims. Each configuration is represented by a binary vector $\alpha \in \{0, 1\}^k$, where $\alpha_i = 1$ indicates inclusion of claim C_i and $\alpha_i = 0$ indicates exclusion. This leads to the following natural measure of *stability* for H with respect to P and $\mathcal{E}$:

$$\tau(\mathcal{E}, P, H) = \sum_{\alpha \in \{0,1\}^k} \mathcal{E}(P(\alpha), H) \cdot \Pr[\alpha], \tag{3}$$

where $\Pr[\alpha]$ is the probability of this specific configuration of premise claim inclusions, and depends on the base and derived claims, as well as the entailment model $\mathcal{E}$, which we discuss next.

3.2 Autoregressive Reasoning Entailment Stability with Efficient Sampling

In the previous section, we established a method for calculating the entailment of a single hypothesis based on a set of premises that might be uncertain (Equation (3)). Now, we will extend this concept to evaluate an entire LLM-generated reasoning chain, which consists of multiple, sequential steps. Our goal is to compute a sequence of *entailment stability scores*, denoted as $\tau_1, \dots, \tau_m$, where each score τ_k quantifies the reliability of the k-th derived claim, C_{n+k}.

The core challenge remains the same: how do we reliably judge a claim when the preceding claims it relies on are themselves not entirely trustworthy? Our approach, **Autoregressive Reasoning Entailment Stability (ARES)**, solves this by autoregressively assessing each claim while accounting for the soundness of previous claims. In particular, when we evaluate the k-th derived claim, we consider all possible combinations of soundness for the preceding $n+k-1$ claims. The stability score, τ_k, is then the expected entailment of the current claim, averaged across all sound combinations.

To formalize this, we represent a particular combination of inclusion or exclusion of previous claims using a binary vector $\alpha \in \{0,1\}^{n+k-1}$, where let $\alpha_i = 1$ denote the inclusion of claim C_i and let $\alpha_i = 0$ denote its exclusion. The probability of this combination $\Pr[\alpha]$ is calculated recursively as follows:

- **Base Case ($k = 1$):** For the first derived claim, C_{n+1}, the premises are the initial base claims $C_1, \dots, C_n$. We assume that each base claim C_i is associated with a prior probability of soundness p_i that is given. Therefore, let:
$$\Pr[\alpha_{1:n}] = \prod_{i=1}^{n} p_i^{\alpha_i} (1 - p_i)^{\alpha_i} \tag{4}$$
- **Inductive Case ($k > 1$):** For any derived claim after the first, the probability of a specific premise combination $\alpha_{1:n+k}$ depends on two factors: the probability of the preceding combination ($\Pr[\alpha_{1:n+k-1}]$) and the entailment probability of the new claim given that preceding combination. In other words, a claim is added to our set of "sound" premises for the next step based on how strongly the currently accepted set entails it:
$$\Pr[\alpha_{1:n+k}] = \Pr[\alpha_{1:n+k-1}] \ \cdot \ \mathcal{E}(C(\alpha_{1:n+k-1}), C_{n+k}) \tag{5}$$
where $C(\alpha_{1:n+k-1})$ denotes the subset of claims indexed by $\alpha_{1:n+k-1} \in \{0,1\}^{n+k-1}$.

Using the above definition for $\Pr[\alpha]$, we may quantify how likely each combination of previous claims may affect the current entailment. In particular, we naturally define the *entailment stability score* τ_k for the k-th derived claim as a marginalization over all combinations of its predecessors:

$$\tau_k = \sum_{\alpha \in \{0,1\}^{n+k-1}} \mathcal{E}(C(\alpha), C_{n+k}) \cdot \Pr[\alpha] \tag{6}$$

However, directly computing τ_k is highly inefficient, as it requires summing over 2^{n+k-1} possible premise combinations. Instead, we estimated it by sampling the premise combinations:

$$\hat{\tau}_k = \frac{1}{N} \sum_{i=1}^{N} \mathcal{E}(C(\alpha^{(i)}), C_{n+k}), \tag{7}$$

where let $\alpha^{(1)}, \dots, \alpha^{(N)} \sim \{0,1\}^{n+k-1}$ be i.i.d. sampled according to Algorithm 1 and Figure 3. Additionally, note that $\hat{\tau}_k$ converges rapidly to τ as the number of samples N grows, allowing us to obtain a rigorous statistical guarantee on our stability scores as a function of the number of samples.

Theorem 3.1. *Let $N \geq \frac{\log(2m/\delta)}{2\varepsilon^2}$ for any $\varepsilon > 0$ and $\delta > 0$. Given an entailment model $\mathcal{E}$ and a reasoning chain with m derived claims, use N i.i.d. samples to estimate each τ_k. Then, with probability at least $1 - \delta$, we have $|\hat{\tau}_k - \tau_k| \leq \varepsilon$ for all k.*

Proof. See Appendix A. □

Dataset / Method	GPT-4o-mini			Qwen3-4B		
	Recall	Precision	F1	Recall	Precision	F1
PRMBench						
ARES	**0.680 ± 0.024**	**0.627 ± 0.021**	**0.640 ± 0.023**	**0.688 ± 0.020**	0.623 ± 0.011	0.636 ± 0.011
Entail-Prev	0.639 ± 0.032	0.602 ± 0.016	0.596 ± 0.024	**0.698 ± 0.016**	0.626 ± 0.015	0.641 ± 0.017
Entail-Base	0.524 ± 0.022	0.511 ± 0.011	0.484 ± 0.016	0.631 ± 0.016	0.558 ± 0.007	0.530 ± 0.011
ROSCOE-LI-Self	**0.672 ± 0.012**	0.575 ± 0.007	0.489 ± 0.022	0.458 ± 0.011	0.478 ± 0.006	0.446 ± 0.006
ROSCOE-LI-Source	**0.676 ± 0.014**	0.584 ± 0.008	0.570 ± 0.011	0.497 ± 0.003	0.496 ± 0.004	0.495 ± 0.004
ReCEval-Intra	0.563 ± 0.012	0.581 ± 0.014	0.568 ± 0.013	0.550 ± 0.007	0.573 ± 0.013	0.554 ± 0.007
ReCEval-Inter	0.664 ± 0.012	0.573 ± 0.007	0.465 ± 0.022	0.449 ± 0.004	0.476 ± 0.003	0.433 ± 0.004
LLM-Judge	0.647 ± 0.011	**0.645 ± 0.019**	**0.643 ± 0.013**	**0.695 ± 0.017**	**0.662 ± 0.016**	**0.675 ± 0.016**
DeltaBench						
ARES	**0.702 ± 0.024**	**0.728 ± 0.022**	**0.708 ± 0.026**	0.513 ± 0.013	0.512 ± 0.013	0.498 ± 0.010
Entail-Prev	**0.698 ± 0.032**	**0.709 ± 0.029**	**0.699 ± 0.031**	0.523 ± 0.011	0.522 ± 0.010	0.506 ± 0.009
Entail-Base	0.614 ± 0.010	0.596 ± 0.004	0.594 ± 0.005	**0.580 ± 0.008**	0.586 ± 0.008	**0.579 ± 0.009**
ROSCOE-LI-Self	0.579 ± 0.006	0.664 ± 0.027	0.571 ± 0.013	0.555 ± 0.007	**0.638 ± 0.039**	0.522 ± 0.003
ROSCOE-LI-Source	0.471 ± 0.006	0.456 ± 0.009	0.453 ± 0.005	0.484 ± 0.013	0.472 ± 0.021	0.457 ± 0.017
ReCEval-Intra	0.500 ± 0.000	0.357 ± 0.012	0.416 ± 0.009	0.530 ± 0.006	0.529 ± 0.005	0.528 ± 0.005
ReCEval-Inter	0.503 ± 0.007	0.508 ± 0.012	0.483 ± 0.010	0.507 ± 0.006	0.508 ± 0.006	0.505 ± 0.007
LLM-Judge	0.498 ± 0.002	0.371 ± 0.026	0.381 ± 0.027	0.548 ± 0.010	0.563 ± 0.016	0.494 ± 0.009
ClaimTrees						
ARES	**0.914 ± 0.012**	**0.921 ± 0.013**	**0.903 ± 0.020**	**0.731 ± 0.006**	0.755 ± 0.009	**0.723 ± 0.006**
Entail-Prev	0.587 ± 0.012	0.704 ± 0.025	0.491 ± 0.020	0.580 ± 0.013	0.760 ± 0.006	0.480 ± 0.022
Entail-Base	0.645 ± 0.018	0.647 ± 0.019	0.619 ± 0.021	0.586 ± 0.019	0.630 ± 0.018	0.521 ± 0.026
ROSCOE-LI-Self	0.528 ± 0.005	0.569 ± 0.016	0.430 ± 0.011	0.568 ± 0.009	0.732 ± 0.005	0.473 ± 0.017
ROSCOE-LI-Source	0.540 ± 0.012	0.543 ± 0.013	0.511 ± 0.016	0.491 ± 0.004	0.484 ± 0.006	0.448 ± 0.008
ReCEval-Intra	0.500 ± 0.000	0.254 ± 0.006	0.336 ± 0.005	0.500 ± 0.000	0.252 ± 0.003	0.335 ± 0.003
ReCEval-Inter	0.546 ± 0.013	0.548 ± 0.013	0.513 ± 0.016	0.495 ± 0.003	0.489 ± 0.005	0.451 ± 0.007
LLM-Judge	0.687 ± 0.018	0.780 ± 0.016	0.628 ± 0.027	0.602 ± 0.026	**0.769 ± 0.013**	0.502 ± 0.034
CaptainCookRecipes						
ARES	**0.636 ± 0.010**	0.657 ± 0.011	**0.633 ± 0.010**	0.532 ± 0.012	0.532 ± 0.012	0.517 ± 0.009
Entail-Prev	0.468 ± 0.004	0.462 ± 0.004	0.428 ± 0.010	0.511 ± 0.005	0.529 ± 0.014	0.384 ± 0.008
Entail-Base	0.591 ± 0.007	0.598 ± 0.008	0.589 ± 0.007	0.500 ± 0.000	0.290 ± 0.005	0.367 ± 0.005
ROSCOE-LI-Self	0.555 ± 0.005	**0.703 ± 0.018**	0.483 ± 0.011	**0.619 ± 0.007**	**0.711 ± 0.012**	**0.601 ± 0.010**
ROSCOE-LI-Source	0.500 ± 0.000	0.283 ± 0.009	0.361 ± 0.007	0.500 ± 0.000	0.290 ± 0.006	0.367 ± 0.004
ReCEval-Intra	0.515 ± 0.008	0.540 ± 0.022	0.396 ± 0.010	0.500 ± 0.000	0.290 ± 0.006	0.367 ± 0.004
ReCEval-Inter	0.500 ± 0.000	0.283 ± 0.009	0.361 ± 0.007	0.500 ± 0.000	0.290 ± 0.005	0.367 ± 0.004
LLM-Judge	0.560 ± 0.023	0.569 ± 0.024	0.530 ± 0.028	0.500 ± 0.000	0.289 ± 0.005	0.366 ± 0.004

Table 1: **(Benchmark Results)** ARES is top-performing in majority of settings (5/8), with no other single method being a consistent challenger. For each dataset+model group, **Bold** is the best and underline is the second best.

Error Detection. Recall the connection between entailment stability and error detection: the lower a claim's entailment stability, the greater its error. Consider a simple thresholding mechanism: if some $\hat{\tau}_k$ falls below a prescribed error threshold, then we mark the derived claim C_{n+k} as erroneous. In the following, we demonstrate the empirical effectiveness of this procedure.

4 Evaluating ARES for Estimating Probabilistic Soundness

ARES performs error detection by estimating the entailment stability of each derived claim and applying a thresholding mechanism. We next run experiments to validate the performance of ARES against multiple baselines on diverse benchmarks.

Experiment Setup. We compare ARES with baselines including LLM-Judge, Entail-Prev, Entail-Base, and pairwise comparison methods from ROSCOE [Golovneva et al., 2023] and ReCEval [Prasad et al., 2023]. Our experiments used proprietary (GPT-4o-mini [OpenAI, 2024]) and open-source (Qwen3-4B [Yang et al., 2025], Qwen2.5-Math-PRM-7B [Zhang et al., 2025]) models. We tested on established benchmarks (PRMBench [Song et al., 2025], DeltaBench [He et al., 2025]) and two new synthetic datasets, ClaimTrees and CaptainCookRecipes, designed to isolate error propagation. Performance was measured using Macro-F1 score with a 5-fold cross-validation setup (see Appendix C for full details).

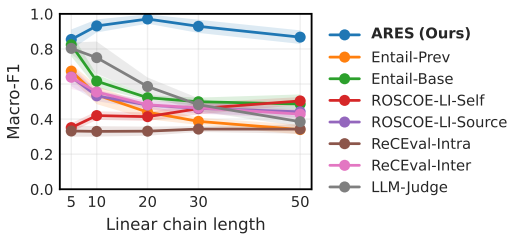


Figure 4: **(ClaimTrees) GPT-4o-mini.** ARES can robustly identify error propagations in long reasoning chains, whereas other methods fail.

Method	Step Avg	Final Step
ARES	**0.730**	**0.660**
Entail-Prev	**0.790**	0.240
Entail-Base	0.540	0.300
ROSCOE-LI-Self	0.540	0.210
ROSCOE-LI-Source	0.630	0.310
ReCEval-Intra	0.480	0.060
ReCEval-Inter	0.480	0.190
LLM-Judge	0.570	0.250

Figure 5: **(PRMBench Best-of-N)** ARES is the best at choosing the best sequence for downstream task performance. **Bold**: best within bootstrap standard error.

Method	**PRMBench**	**DeltaBench**	**ClaimTrees-10**	**CaptainCookRecipes**
ARES-ε0.1	**0.640**	**0.708**	**0.931**	<u>0.633</u>
ARES-ε0.2	<u>0.599</u>	<u>0.697</u>	<u>0.926</u>	0.631
ARES-ε0.3	0.582	0.694	0.919	0.621
ARES-ε0.4	0.595	0.687	0.922	**0.640**

Table 2: **(GPT-4o-mini) Performance Convergence with Samples** ARES is able to achieve high accuracy even when using a smaller number of samples. When ε =0.1, 0.2, 0.3, 0.4, a sequence of length $m = 10$ needs 265, 67, 30, 17 samples per step respectively. We can see that there is no significant performance change when we increase the ϵ to 0.4 and thus decrease the number of samples 15x.

4.1 RQ1: Does ARES work better than baseline methods on Benchmarks?

On natural reasoning chains in PRMBench and DeltaBench, ARES consistently achieves the best Macro-F1 scores (Table 1). Baselines struggled with specific challenges; for instance, DeltaBench's long reasoning chains appeared to confuse LLM-Judge's holistic judgments, while Entail-Base underperformed on PRMBench. Additional experiments show ARES also improves the performance of PRM backbones (Appendix C.9).

4.2 RQ2: In what setting does ARES identify more errors than baselines?

To pinpoint where ARES excels, we created two synthetic datasets designed to test error propagation. By intentionally removing a key base claim in each—such as a logical rule or an ingredient—we created unsound derivations to precisely track how errors propagate. The performance gap is most pronounced on these datasets; as shown in Figure 4, ARES maintains a high Macro-F1 score (over 89%) on chains up to 50 steps long, while baseline performance collapses. This confirms that ARES uniquely satisfies the key desiderata for detecting propagated errors (Appendix B).

4.3 RQ3: Is ARES computationally efficient?

Our analysis shows that ARES's performance remains stable even with significantly fewer samples, indicating its efficiency and potential for further computational savings (Table 2). On synthetic benchmarks, performance is consistent for ε from 0.1 to 0.4, while more variance is seen on PRMBench and DeltaBench.

4.4 RQ4: Is ARES useful for selecting Best-of-N generations?

In a best-of-n selection task on PRMBench, ARES was significantly better at identifying the correct reasoning chain when using the final step's score—a strict metric on which simpler approaches like Entail-Prev collapse (Figure 5). This highlights its reliability and robustness as a predictor for practical applications.

4.5 Ablations

We performed several ablations on ClaimTrees to analyze ARES's strengths, testing its robustness against irrelevant claims and benign, non-propagating errors.

Irrelevant Claims and Benign Errors. ARES maintains high performance on both deep and wide reasoning trees, effectively ignoring irrelevant claims that degrade the performance of other methods (Table A8). In cases with benign errors that do not affect subsequent reasoning steps, all methods perform equally well (Table A9).

Choice of p and Entailment Model Granularity. Our ablations show that using a probabilistic entailment model with $p = 1$ (including all base claims) consistently yields the best and most computationally efficient performance for ARES. A binary entailment model, in contrast, sometimes benefits from a slightly lower $p = 0.95$ (Table A7).

4.6 Discussion of Errors

Our error analysis reveals specific baseline failure modes. Methods like Entail-Base and LLM-Judge struggle with long, complex reasoning chains, while Entail-Prev fails to detect the propagated errors present in our synthetic data. Pairwise methods are limited to simple errors requiring few premises. Ultimately, ARES's effectiveness is bounded by the capability of its underlying entailment model.

5 Related Work

Reasoning Chain Verifiers. Approaches to verifying reasoning chains include LLM Judges [Tyagi et al., 2024, He et al., 2024, 2025] and Process Reward Models (PRMs) [Lightman et al., 2023]. While recent verifiers incorporate logic, they have limitations: ROSCOE [Golovneva et al., 2023] and ReCEval [Prasad et al., 2023] use pairwise contradiction, which is less effective with complex premises, and PARC [Mukherjee et al., 2025] provides only a binary soundness classification. Our work differs by introducing a probabilistic framework for a more nuanced assessment of each claim.

Evaluating Reasoning Error Detectors. While many benchmarks exist for evaluating CoT error detectors—including GridPuzzle [Tyagi et al., 2024], REVEAL [Jacovi et al., 2024], PRMBench [Song et al., 2025], ProcessBench [Zheng et al., 2024], and DeltaBench [He et al., 2025]—they lack a consistent definition of error. We establish a clear standard by adopting a unified definition of soundness, incorporating concepts of validity and groundedness from Lee and Hockenmaier [2025] and propagated error from Mukherjee et al. [2025]. We define a step as unsound if it is not logically entailed by correct preceding claims. Since existing benchmarks do not uniformly apply this standard, we created synthetic datasets for a more robust evaluation.

Probabilistic Guarantees. Our work applies statistical guarantees for reliability—a practice common in high-stakes domains [Fayyad et al., 2024, McShane et al., 2023, Lindemann et al., 2023] and explainable AI [Jin et al., 2025]—to natural language reasoning. We provide soundness guarantees over the entire directed acyclic graph (DAG) of a reasoning chain. This holistic approach contrasts with frameworks like BIRD [Feng et al., 2025], which calibrate individual components (e.g., the entailment model), whereas we provide formal guarantees for the multi-step process itself. As a model-agnostic framework, our method is complementary and can leverage improved, calibrated models to enhance its own certification reliability.

6 Conclusion

Current methods cannot reliably detect LLM reasoning errors that propagate. To overcome these limitations, we introduce Autoregressive Reasoning Entailment Stability (ARES), a model-agnostic framework for the probabilistic certification of LLM reasoning. Theoretically, ARES offers a novel probabilistic approach to inductively assess reasoning soundness by considering only previously validated claims, mirroring human-like error checking that discards incorrect intermediate steps. Experimentally, ARES demonstrates superior performance in robustly identifying errors in lengthy and complex reasoning chains, outperforming existing methods that degrade under error propagation.

References

Chirag Agarwal, Sree Harsha Tanneru, and Himabindu Lakkaraju. Faithfulness vs. plausibility: On the (un) reliability of explanations from large language models. *arXiv preprint arXiv:2402.04614*, 2024.

Jiuhai Chen and Jonas Mueller. Quantifying uncertainty in answers from any language model and enhancing their trustworthiness. *arXiv preprint arXiv:2308.16175*, 2023.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. *arXiv preprint arXiv:2309.11495*, 2023.

Jamil Fayyad, Shadi Alijani, and Homayoun Najjaran. Empirical validation of conformal prediction for trustworthy skin lesions classification, 2024. URL `https://arxiv.org/abs/2312.07460`.

Yu Feng, Ben Zhou, Weidong Lin, and Dan Roth. BIRD: A trustworthy bayesian inference framework for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=fAAaT826Vv`.

Olga Golovneva, Moya Peng Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. ROSCOE: A suite of metrics for scoring step-by-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=xYlJRpzZtsY`.

Hangfeng He, Hongming Zhang, and Dan Roth. Socreval: Large language models with the socratic method for reference-free reasoning evaluation, 2024. URL `https://arxiv.org/abs/2310.00074`.

Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Zhongyuan Peng, Zhaoxiang Zhang, Zhicheng Zheng, Wenbo Su, and Bo Zheng. Can large language models detect errors in long chain-of-thought reasoning?, 2025. URL `https://arxiv.org/abs/2502.19361`.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

Alon Jacovi, Yonatan Bitton, Bernd Bohnet, Jonathan Herzig, Or Honovich, Michael Tseng, Michael Collins, Roee Aharoni, and Mor Geva. A chain-of-thought is as strong as its weakest link: A benchmark for verifiers of reasoning chains. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4615–4634, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.254. URL `https://aclanthology.org/2024.acl-long.254/`.

Helen Jin, Anton Xue, Weiqiu You, Surbhi Goel, and Eric Wong. Probabilistic stability guarantees for feature attributions. *arXiv preprint arXiv:2504.13787*, 2025.

Phil Johnson-Laird. Deductive reasoning. *Wiley Interdisciplinary Reviews: Cognitive Science*, 1(1): 8–17, 2010.

Jinu Lee and Julia Hockenmaier. Evaluating step-by-step reasoning traces: A survey, 2025. URL `https://arxiv.org/abs/2502.12289`.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL `https://arxiv.org/abs/2305.20050`.

Lars Lindemann, Matthew Cleaveland, Gihyun Shim, and George J. Pappas. Safe planning in dynamic environments using conformal prediction, 2023. URL `https://arxiv.org/abs/2210.10254`.

Qing Lyu, Marianna Apidianaki, and Chris Callison-Burch. Towards faithful model explanation in NLP: A survey. *Computational Linguistics*, 50(2):657–723, June 2024. doi: 10.1162/coli_a_00511. URL `https://aclanthology.org/2024.cl-2.6/`.

Staffan Arvidsson McShane, Ulf Norinder, Jonathan Alvarsson, Ernst Ahlberg, Lars Carlsson, and Ola Spjuth. Cpsign-conformal prediction for cheminformatics modeling. *bioRxiv*, pages 2023–11, 2023.

Sagnik Mukherjee, Abhinav Chinta, Takyoung Kim, Tarun Anoop Sharma, and Dilek Hakkani-Tür. Premise-augmented reasoning chains improve error identification in math reasoning with llms, 2025. URL `https://arxiv.org/abs/2502.02362`.

OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`, July 2024. Accessed: 2025-05-19.

Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Eric Ragan, Nicholas Ruozzi, Yu Xiang, and Vibhav Gogate. Captaincook4d: A dataset for understanding errors in procedural activities. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 135626–135679. Curran Associates, Inc., 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/f4a04396c2ed1342a5d8d05e94cb6101-Paper-Datasets_and_Benchmarks_Track.pdf`.

Archiki Prasad, Swarnadeep Saha, Xiang Zhou, and Mohit Bansal. ReCEval: Evaluating reasoning chains via correctness and informativeness. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10066–10086, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.622. URL `https://aclanthology.org/2023.emnlp-main.622/`.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained and challenging benchmark for process-level reward models, 2025. URL `https://arxiv.org/abs/2501.03124`.

Seok Hwan Song and Wallapak Tavanapong. How much do prompting methods help llms on quantitative reasoning with irrelevant information? In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 2128–2137, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Nemika Tyagi, Mihir Parmar, Mohith Kulkarni, Aswin Rrv, Nisarg Patel, Mutsumi Nakamura, Arindam Mitra, and Chitta Baral. Step-by-step reasoning to solve grid puzzles: Where do LLMs falter? In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19898–19915, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1111. URL `https://aclanthology.org/2024.emnlp-main.1111/`.

Siye Wu, Jian Xie, Jiangjie Chen, Tinghui Zhu, Kai Zhang, and Yanghua Xiao. How easily do irrelevant inputs skew the responses of large language models? *arXiv preprint arXiv:2404.03302*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Fei Yu, Hongbo Zhang, Prayag Tiwari, and Benyou Wang. Natural language reasoning, a survey. *ACM Computing Surveys*, 56(12):1–39, 2024.

Lotfi A Zadeh. Fuzzy logic. *Scholarpedia*, 3(3):1766, 2008.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning, 2025. URL `https://arxiv.org/abs/2501.07301`.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning, 2024. URL `https://arxiv.org/abs/2412.06559`.

A Proofs

Theorem 3.1. *Let* $N \geq \frac{\log(2m/\delta)}{2\varepsilon^2}$ *for any* $\varepsilon > 0$ *and* $\delta > 0$. *Given an entailment model* $\mathcal{E}$ *and a reasoning chain with* m *derived claims, use* N *i.i.d. samples to estimate each* τ_k. *Then, with probability at least* $1 - \delta$, *we have* $|\hat{\tau}_k - \tau_k| \leq \varepsilon$ *for all k.*

Proof. Let $\mathcal{A}_i$ denote the event that $|\hat{\tau}_i - \tau_i| < \varepsilon$ for each $i \in \{n+1, \ldots, n+m\}$. We want to prove that

$$\Pr\left(\bigcap_{i=n+1}^{n+m} \mathcal{A}_i\right) = 1 - \Pr\left(\bigcup_{i=n+1}^{n+m} \bar{\mathcal{A}}_i\right) \geq 1 - \delta. \tag{8}$$

According to Boole's inequality and Hoeffding's inequality,

$$\begin{aligned}
\Pr\left(\bigcup_{i=n+1}^{n+m} \bar{\mathcal{A}}_i\right) &\leq \sum_{i=n+1}^{n+m} \Pr\left(\bar{\mathcal{A}}_i\right) && \text{(Boole's)} \\
&= \sum_{i=n+1}^{n+m} \Pr\left(|\hat{\tau}_i - \tau_i| \geq \varepsilon\right) && (9) \\
&\leq \sum_{i=n+1}^{n+m} 2\exp(-2N\varepsilon^2) && \text{(Hoeffding's)} \\
&= 2m\exp(-2N\varepsilon^2) && (10) \\
&\leq \delta \quad \text{when } N \geq \frac{\log(2m/\delta)}{2\varepsilon^2}, && (11)
\end{aligned}$$

with the estimation error of each stability rate bounded by $\delta_i = \frac{\delta}{m}$. □

B Method

There are three important desiderata for error detection methods:

1. **Robust:** Previous errors do not adversely affect current step.
2. **Causal:** Downstream steps do not affect current step.
3. **Sufficient:** All relevant claims included as premise for detection.

Appendix B shows that only ARES satisfies all desiderata while none of the baseline methods does. Algorithm details is shown in Algorithm 1.

C Experiments

C.1 Entailment Model

We instantiate the entailment model by prompting LLMs to judge the entailment of a hypothesis given a premise, where there can be multiple claims in the premise. The LLM's output is either YES/NO in the binary case, or a 7-point Likert scale converted to a real value between 0 and 1.

Method	**Robust**	**Causal**	**Sufficient**
ARES (ours)	✓	✓	✓
Entail-Prev	✗	✓	✓
Entail-Base	✓	✓	✗
ROSCOE-LI-Self	✗	✓	✗
ROSCOE-LI-Source	✗	✓	✗
ReCEval-Intra	✓	✓	✗
ReCEval-Inter	✗	✓	✗
LLM-Judge	✗	✗	✓

Table A3: (Desiderata for methods) **Robust:** Previous errors do not adversely affect current step. **Causal:** Downstream steps do not affect current step. **Sufficient:** All relevant claims included as premise for detection.

Algorithm 1 Estimating ARES

```
Require: Reasoning chain $(C_1, \dots, C_{n+m})$, tolerance $(\varepsilon, \delta)$, base priors $p_1, \dots, p_n$, and entailment model $\mathcal{E}$.
 1: $N \leftarrow \frac{\log(2m/\delta)}{2\varepsilon^2}$
 2: for $i = 1, \dots, N$ do
 3:     $\alpha_1^{(i)} \sim \text{Bernoulli}(p_1), \dots, \alpha_n^{(i)} \sim \text{Bernoulli}(p_n)$
 4:     for $k = 1, \dots, m$ do
 5:         $p_{n+k}^{(i)} \leftarrow \mathcal{E}(C(\alpha_{1:n+k-1}^{(i)}), C_{n+k})$
 6:         $\alpha_{n+k}^{(i)} \sim \text{Bernoulli}(p_{n+k}^{(i)})$
 7:     end for
 8: end for
 9: for $k = 1, \dots, m$ do
10:     $\hat{\tau}_k = \frac{1}{N} \sum_{i=1}^{N} p_{n+k}^{(i)}$
11: end for
```

C.2 Hyperparameters for ARES

In our experiments, we used $\delta = 0.1$ and $\varepsilon = 0.1$ for ARES, which determines the number of samples to take. We use $p = 0.95$ for the inclusion rate for base claims to allow buffer for information overload.

C.3 Experiment Details

We use a subset of examples for each experiment. Experiment results are computed using 5-fold cross-validation. For each split, the thresholds are picked for the best Macro-F1 on the validation split, and the final numbers are on the test split, averaged over the 5 folds.

C.4 Controllable Datasets

ClaimTrees. One is ClaimTrees, a synthetic dataset in which the reasoning chain reasons starts from a state A, and reason all the way to another state, say T. All the reasoning rules are provided in the premise, except one, so that from that point on we know that all the claims are unsound: An example of a chain of reasoning is shown in Figure A7. In this example, rule B -> C does not actually exist, and thus the reasoning steps starting from the second derived step are unsound claims. We can construct reasoning chains with arbitrary length and errors occurring at different places.

CaptainCookRecipes. CaptainCookRecipes is derived from the recipe graphs in Captain-Cook4D [Peddi et al., 2024], where certain actions must follow other actions. We then construct base claims using edges in the graph as rules, similar to how we construct the ones in ClaimTrees. In addition, we add ingredients to the base claims and randomly drop an ingredient. Then, all the claims that require the ingredient and claims that follow them become unsound. We extract the ingredients from the claims using GPT-4o-mini.

An example of results for CaptainCookRecipes is shown in Table A4. With propagated errors present, only ARES is able to capture all errors.

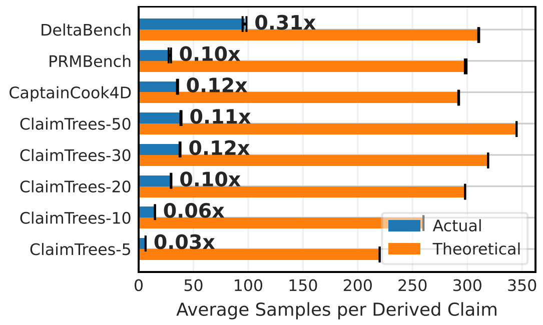


Figure A6: **(Per-Claim Samples)** ARES in practice only uses 0.03x to 0.31x of theoretical number of samples on average for each derived claim.

Long Chain Example.

Base Claims:
Rule: AZ -> DG (meaning that if I have AZ, I can derive DG)
Rule: SG -> H3 (meaning that if I have SG, I can derive H3)
I have AZ
Rule: DG -> SG (meaning that if I have DG, I can derive SG)
Reasoning Steps:
I have AZ, I use rule (AZ -> DG) to derive DG, now I have DG
I have DG, I use rule (DG -> SG) to derive SG, now I have SG
I have SG, I use rule (SG -> H3) to derive H3, now I have H3
I have H3, I use rule (H3 -> VD) to derive VD, now I have VD

Figure A7: Long Chain Example for ClaimTrees

C.5 Computing Resources

We used an NVIDIA A100 GPU with 80GB of memory for the Qwen3-4B model. For GPT-4o-mini, we used approximately 600 USD in total for prototyping and experiments.

C.6 Additional Computational Efficiency Analysis

ARES's computational efficiency stems from a two-tiered optimization. First, ARES uses a sampling-based strategy for soundness checking, which is inherently more efficient than an exhaustive approach. Second, we add another layer of efficiency by eliminating redundant LLM calls for the same premise-hypothesis pairs. This dual approach dramatically reduces computational overhead, as shown by the gap between theoretical and actual samples in Figure A6. The result is a highly efficient process: on shorter chains (ClaimTrees-5), we require only 0.03x the theoretical samples. Even on DeltaBench, which needs more sampling due to model uncertainty, the method remains effective at 0.31x the theoretical maximum.

C.7 Probabilistic Entailment Model Output

To obtain probabilistic entailment model output, we instruct LLM to output one of the following: Very Likely, Likely, Somewhat Likely, Neutral, Somewhat Unlikely, Unlikely, Very Unlikely and convert them to 1, 0.8, 0.6, 0.5, 0.4, 0.2, 0.0, respectively.

C.8 Best-of-N Results

For best-of-N result with standard deviations, see Table A5.

C.9 ARES Also Improves PRMs

Process Reward Models (PRMs) can sometimes rival LLMs, and can also provide a non-binary soundness score. We run additional experiments using a SOTA PRMs, Qwen2.5-Math-PRM-7B, as the base entailment model. The results show that ARES can help significantly improve upon PRM on reasoning chains with propagated errors.

The results in Table A6 show that, while the specialized PRM is a strong baseline on its in-domain dataset (PRMBench), applying ARES significantly improves performance on the abstract ClaimTrees dataset which has many propagated errors. On out-of-domain (non-math) CaptainCook4D, ARES achieves on par performance with PRM. This demonstrates ARES's value as a flexible, general-purpose framework that adds robustness, especially on tasks with propagated errors.

C.10 Discussion of Errors

Our inspection of the data and error detection outputs reveals some insights. Entail-Base fails on PRMBench because judging entailment in long math derivations is challenging. Both LLM-Judge and Entail-Base fail in DeltaBench, with Entail-Base struggling to judge entailment in very long reasoning chains. In naturally occurring datasets, error propagation is limited and not always annotated, so Entail-Prev performs close to ARES. However, synthetic data shows Entail-Prev fails with propagated errors. LLM-Judge sometimes fails to follow instructions, outputting incorrect numbers of scores relative to claims being judged. Pairwise methods in ROSCOE and ReCEval cannot detect complex errors that need multiple claims as premise. ARES can only improve upon entailment models that can already do correct entailment.

Claim	**ARES (Ours)**	Entail -Prev	Entail -Base	ReCEval -Inter	ReCEval -Intra	ROSCOE -LI-Source	ROSCOE -LI-Self	LLM -Judge	*Ground Truth*
sent1: Only after the necessary preceding steps (put-put tomatoes on a serving plate), And if we have all the ingredients, we can then Pour-Pour the egg mixture into the pan.	–	–	–	–	–	–	–	–	–
sent2: Only after the necessary preceding steps (Take-Take a tomato), And if we have all the ingredients, we can then Cut-Cut tomato into two pieces.	–	–	–	–	–	–	–	–	–
sent3: Only after the necessary preceding steps (Stop-Stop stirring when it's nearly cooked to allow it to set into an omelette), And if we have all the ingredients, we can then Transfer-Transfer omelette to the plate and serve with the tomatoes.	–	–	–	–	–	–	–	–	–
sent4: Only after the necessary preceding steps (Chop-Chop 2 tbsp cilantro), And if we have all the ingredients, we can then add-add the chopped cilantro to the bowl.	–	–	–	–	–	–	–	–	–
sent5: Only after the necessary preceding steps (START), And if we have all the ingredients, we can then add-1/2 tsp ground black pepper to the bowl.	–	–	–	–	–	–	–	–	–
sent6: We have ground black pepper.	–	–	–	–	–	–	–	–	–
sent7: We have oil.	–	–	–	–	–	–	–	–	–
sent8: Only after the necessary preceding steps (Scoop-Scoop the tomatoes from the pan), And if we have all the ingredients, we can then put-put tomatoes on a serving plate.	–	–	–	–	–	–	–	–	–
sent9: Only after the necessary preceding steps (Pour-Pour the egg mixture into the pan), And if we have all the ingredients, we can then stir-stir gently with a wooden spoon so the egg that sets on the base of the pan moves to enable the uncooked egg to flow into the space.	–	–	–	–	–	–	–	–	–
sent10: Only after the necessary preceding steps (Transfer-Transfer omelette to the plate and serve with the tomatoes), And if we have all the ingredients, we can then END.	–	–	–	–	–	–	–	–	–
sent11: Only after the necessary preceding steps (add-add the chopped cilantro to the bowl, and crack-crack one egg in a bowl, and add-1/2 tsp ground black pepper to the bowl), And if we have all the ingredients, we can then Beat-Beat the contents of the bowl.	–	–	–	–	–	–	–	–	–
sent12: Only after the necessary preceding steps (Heat-Heat 1 tbsp oil in a non-stick frying pan), And if we have all the ingredients, we can then cook-cook the tomatoes cut-side down until they start to soften and colour.	–	–	–	–	–	–	–	–	–
sent13: Only after the necessary preceding steps (START), And if we have all the ingredients, we can then crack-crack one egg in a bowl.	–	–	–	–	–	–	–	–	–
sent14: Only after the necessary preceding steps (cook-cook the tomatoes cut-side down until they start to soften and colour), And if we have all the ingredients, we can then Scoop-Scoop the tomatoes from the pan.	–	–	–	–	–	–	–	–	–
sent15: Only after the necessary preceding steps (START), And if we have all the ingredients, we can then Take-Take a tomato.	–	–	–	–	–	–	–	–	–
sent16: Only after the necessary preceding steps (Beat-Beat the contents of the bowl, and Cut-Cut tomato into two pieces), And if we have all the ingredients, we can then Heat-Heat 1 tbsp oil in a non-stick frying pan.	–	–	–	–	–	–	–	–	–
sent17: We have egg.	–	–	–	–	–	–	–	–	–
sent18: Only after the necessary preceding steps (START), And if we have all the ingredients, we can then Chop-Chop 2 tbsp cilantro.	–	–	–	–	–	–	–	–	–
sent19: Only after the necessary preceding steps (stir-stir gently with a wooden spoon so the egg that sets on the base of the pan moves to enable the uncooked egg to flow into the space), And if we have all the ingredients, we can then Stop-Stop stirring when it's nearly cooked to allow it to set into an omelette.	–	–	–	–	–	–	–	–	–
sent20: We have tomato.	–	–	–	–	–	–	–	–	–
sent21: We now START.	–	–	–	–	–	–	–	–	–
int1: Because we have completed all previous steps (START), and have all necessary ingredients (cilantro), we can now do the step Chop-Chop 2 tbsp cilantro. And now we have completed this step Chop-Chop 2 tbsp cilantro.	**0.35**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	**0.00**×	1.00✓	1.00✓	×
int2: Because we have completed all previous steps (START), and have all necessary ingredients (egg), we can now do the step crack-crack one egg in a bowl. And now we have completed this step crack-crack one egg in a bowl.	**0.85**✓	**1.00**✓	**1.00**✓	0.00×	**1.00**✓	0.00×	0.00×	**1.00**✓	✓
int3: Because we have completed all previous steps (START), and have all necessary ingredients (tomato), we can now do the step Take-Take a tomato. And now we have completed this step Take-Take a tomato.	**0.98**✓	**1.00**✓	**1.00**✓	0.00×	**1.00**✓	0.00×	0.00×	**1.00**✓	✓
int4: Because we have completed all previous steps (START), and have all necessary ingredients (ground black pepper), we can now do the step add-1/2 tsp ground black pepper to the bowl. And now we have completed this step add-1/2 tsp ground black pepper to the bowl.	**0.80**✓	**1.00**✓	**1.00**✓	0.00×	**1.00**✓	0.00×	**1.00**✓	**1.00**✓	✓
int5: Because we have completed all previous steps (Chop-Chop 2 tbsp cilantro), and have all necessary ingredients (cilantro), we can now do the step add-add the chopped cilantro to the bowl. And now we have completed this step add-add the chopped cilantro to the bowl.	**0.00**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int6: Because we have completed all previous steps (Take-Take a tomato), and have all necessary ingredients (tomato), we can now do the step Cut-Cut tomato into two pieces. And now we have completed this step Cut-Cut tomato into two pieces.	**0.96**✓	**1.00**✓	**1.00**✓	0.00×	**1.00**✓	0.00×	0.00×	**1.00**✓	✓
int7: Because we have completed all previous steps (add-add the chopped cilantro to the bowl, and crack-crack one egg in a bowl, and add-1/2 tsp ground black pepper to the bowl), we can now do the step Beat-Beat the contents of the bowl. And now we have completed this step Beat-Beat the contents of the bowl.	**0.01**×	**0.00**×	1.00✓	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int8: Because we have completed all previous steps (Beat-Beat the contents of the bowl, and Cut-Cut tomato into two pieces), and have all necessary ingredients (oil), we can now do the step Heat-Heat 1 tbsp oil in a non-stick frying pan. And now we have completed this step Heat-Heat 1 tbsp oil in a non-stick frying pan.	**0.00**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int9: Because we have completed all previous steps (Heat-Heat 1 tbsp oil in a non-stick frying pan), and have all necessary ingredients (tomatoes), we can now do the step cook-cook the tomatoes cut-side down until they start to soften and colour. And now we have completed this step cook-cook the tomatoes cut-side down until they start to soften and colour.	**0.01**×	1.00✓	1.00✓	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int10: Because we have completed all previous steps (cook-cook the tomatoes cut-side down until they start to soften and colour), we can now do the step Scoop-Scoop the tomatoes from the pan. And now we have completed this step Scoop-Scoop the tomatoes from the pan.	**0.21**×	1.00✓	1.00✓	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int11: Because we have completed all previous steps (Scoop-Scoop the tomatoes from the pan), we can now do the step put-put tomatoes on a serving plate. And now we have completed this step put-put tomatoes on a serving plate.	**0.18**×	1.00✓	1.00✓	**0.00**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	×
int12: Because we have completed all previous steps (put-put tomatoes on a serving plate), we can now do the step Pour-Pour the egg mixture into the pan. And now we have completed this step Pour-Pour the egg mixture into the pan.	**0.18**×	1.00✓	**0.00**×	**0.00**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	×
int13: Because we have completed all previous steps (Pour-Pour the egg mixture into the pan), we can now do the step stir-stir gently with a wooden spoon so the egg that sets on the base of the pan moves to enable the uncooked egg to flow into the space. And now we have completed this step stir-stir gently with a wooden spoon so the egg that sets on the base of the pan moves to enable the uncooked egg to flow into the space.	**0.19**×	1.00✓	**0.00**×	**0.00**×	**0.00**×	**0.00**×	**0.00**×	1.00✓	×
int14: Because we have completed all previous steps (stir-stir gently with a wooden spoon so the egg that sets on the base of the pan moves to enable the uncooked egg to flow into the space), we can now do the step Stop-Stop stirring when it's nearly cooked to allow it to set into an omelette. And now we have completed this step Stop-Stop stirring when it's nearly cooked to allow it to set into an omelette.	**0.19**×	1.00✓	**0.00**×	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int15: Because we have completed all previous steps (Stop-Stop stirring when it's nearly cooked to allow it to set into an omelette), we can now do the step Transfer-Transfer omelette to the plate and serve with the tomatoes. And now we have completed this step Transfer-Transfer omelette to the plate and serve with the tomatoes.	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×
int16: Because we have completed all previous steps (Transfer-Transfer omelette to the plate and serve with the tomatoes), we can now do the step END. And now we have completed this step END.	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	**0.00**×	**0.00**×	1.00✓	×

Table A4: **(CaptainCookRecipes Example)** Only ARES is able to correctly judge all steps for soundness. Checks ✓ indicate that a method classifies the step as sound after thresholding, and crosses × indicate that the method judges that step to be erroneous. **Bold**: Correctly judged soundness.

Method	Using Step Average (acc±std)	Using Final Step (acc±std)
ARES	**0.730±0.045**	**0.660±0.049**
Entail-Prev	**0.790±0.043**	0.240±0.042
Entail-Base	0.540±0.049	0.300±0.046
ROSCOE-LI-Self	0.540±0.051	0.210±0.041
ROSCOE-LI-Source	0.630±0.049	0.310±0.043
ReCEval-Intra	0.480±0.050	0.060±0.024
ReCEval-Inter	0.480±0.048	0.190±0.038
LLM-Judge	0.570±0.050	0.250±0.044

Table A5: **(PRMBench Best-of-N)** ARES is a strong and robust predictor of downstream task performance. **Bold** is the best and underline is the second best.

Dataset / Method	Qwen2.5-Math-PRM-7B		
	Recall	Precision	F1
PRMBench			
ARES	**0.751 ± 0.017**	**0.733 ± 0.020**	**0.736 ± 0.014**
Entail-Prev	**0.751 ± 0.016**	**0.733 ± 0.020**	**0.736 ± 0.013**
Entail-Base	0.643 ± 0.022	0.632 ± 0.024	0.624 ± 0.018
ROSCOE-LI-Self	0.651 ± 0.013	0.598 ± 0.013	0.592 ± 0.006
ROSCOE-LI-Source	0.670 ± 0.020	0.621 ± 0.019	0.623 ± 0.013
ReCEval-Inter	0.644 ± 0.014	0.597 ± 0.013	0.596 ± 0.009
PRM	**0.763 ± 0.020**	**0.743 ± 0.017**	**0.749 ± 0.016**
ClaimTrees-10			
ARES	**0.739 ± 0.013**	**0.743 ± 0.012**	**0.733 ± 0.010**
Entail-Prev	0.722 ± 0.016	0.725 ± 0.017	0.715 ± 0.011
Entail-Base	0.611 ± 0.013	0.616 ± 0.013	0.597 ± 0.017
ROSCOE-LI-Self	0.655 ± 0.005	0.662 ± 0.005	0.644 ± 0.008
ROSCOE-LI-Source	0.604 ± 0.020	0.612 ± 0.020	0.591 ± 0.024
ReCEval-Inter	0.629 ± 0.020	0.628 ± 0.019	0.624 ± 0.020
PRM	0.607 ± 0.012	0.622 ± 0.013	0.594 ± 0.017
CaptainCook4D			
ARES	**0.551 ± 0.012**	**0.556 ± 0.014**	**0.543 ± 0.012**
Entail-Prev	**0.553 ± 0.011**	**0.560 ± 0.014**	**0.546 ± 0.010**
Entail-Base	0.531 ± 0.016	0.533 ± 0.017	0.519 ± 0.014
ROSCOE-LI-Self	0.546 ± 0.008	**0.563 ± 0.016**	0.529 ± 0.008
ROSCOE-LI-Source	0.469 ± 0.015	0.464 ± 0.018	0.457 ± 0.017
ReCEval-Inter	0.469 ± 0.015	0.465 ± 0.018	0.461 ± 0.017
PRM	**0.560 ± 0.013**	**0.569 ± 0.017**	**0.552 ± 0.013**

Table A6: **(Benchmark Results on Qwen2.5-Math-PRM-7B)** ARES performs the best across various datasets and backbone entailment models. For each dataset+model group, **Bold** is the best and underline is the second best.

Dataset / Method	Recall	Precision	F1
ClaimTrees-5			
ARES-1	0.881	0.900	0.873
ARES-0.95	0.861	0.889	0.854
ARES-bin-1	0.898	0.913	0.891
ARES-bin-0.95	**0.909**	**0.919**	**0.902**
Entail-Prev	0.704	0.813	0.673
Entail-Base	0.830	0.832	0.824
ROSCOE-LI-Self	0.499	0.500	0.351
ROSCOE-LI-Source	0.647	0.650	0.640
ReCEval-Intra	0.500	0.250	0.332
ReCEval-Inter	0.645	0.648	0.638
LLM-Judge	0.811	0.864	0.803
ClaimTrees-10			
ARES-1	0.937	0.943	0.936
ARES-0.95	0.931	0.936	0.931
ARES-bin-1	**0.960**	**0.965**	**0.962**
ARES-bin-0.95	0.947	0.951	0.948
Entail-Prev	0.608	0.783	0.538
Entail-Base	0.626	0.636	0.616
ROSCOE-LI-Self	0.524	0.589	0.420
ROSCOE-LI-Source	0.544	0.548	0.533
ReCEval-Intra	0.500	0.247	0.330
ReCEval-Inter	0.566	0.573	0.555
LLM-Judge	0.767	0.839	0.750
ClaimTrees-20			
ARES-1	**0.979**	**0.979**	**0.978**
ARES-0.95	0.971	0.971	0.971
ARES-bin-1	0.964	0.966	0.963
ARES-bin-0.95	0.968	0.970	0.968
Entail-Prev	0.551	0.760	0.440
Entail-Base	0.533	0.537	0.522
ROSCOE-LI-Self	0.521	0.580	0.414
ROSCOE-LI-Source	0.508	0.509	0.480
ReCEval-Intra	0.500	0.248	0.331
ReCEval-Inter	0.513	0.516	0.482
LLM-Judge	0.640	0.788	0.586
ClaimTrees-30			
ARES-1	**0.973**	0.972	**0.971**
ARES-0.95	0.931	0.934	0.929
ARES-bin-1	0.967	**0.973**	0.969
ARES-bin-0.95	0.957	0.960	0.956
Entail-Prev	0.530	0.731	0.387
Entail-Base	0.531	0.539	0.499
ROSCOE-LI-Self	0.543	0.595	0.460
ROSCOE-LI-Source	0.498	0.498	0.461
ReCEval-Intra	0.500	0.262	0.343
ReCEval-Inter	0.506	0.509	0.464
LLM-Judge	0.581	0.757	0.482
ClaimTrees-50			
ARES-1	**0.895**	0.899	**0.890**
ARES-0.95	0.871	0.871	0.867
ARES-bin-1	0.887	**0.904**	0.886
ARES-bin-0.95	0.892	0.892	0.888
Entail-Prev	0.512	0.601	0.340
Entail-Base	0.507	0.508	0.486
ROSCOE-LI-Self	0.555	0.581	0.504
ROSCOE-LI-Source	0.505	0.509	0.442
ReCEval-Intra	0.500	0.262	0.343
ReCEval-Inter	0.498	0.496	0.428
LLM-Judge	0.529	0.714	0.385

Table A7: **GPT-4o-mini (ClaimTrees)** ARES consistently identifies errors in long reasoning chains while other methods gradually fail.

Dataset / Method	Recall	Precision	F1
ClaimTrees-s3d3			
ARES-1	**0.921± 0.102**	**0.980± 0.018**	**0.941± 0.074**
ARES-0.95	0.904± 0.110	0.975± 0.027	0.927± 0.081
Entail-Prev	0.821± 0.046	0.951± 0.032	0.863± 0.039
Entail-Base	0.859± 0.122	0.866± 0.142	0.837± 0.134
ROSCOE-LI-Self	0.500± 0.000	0.115± 0.060	0.181± 0.078
ROSCOE-LI-Source	0.623± 0.101	0.593± 0.087	0.497± 0.161
ReCEval-Intra	0.500± 0.000	0.115± 0.060	0.181± 0.078
ReCEval-Inter	0.585± 0.081	0.562± 0.061	0.449± 0.115
LLM-Judge	0.833± 0.051	0.957± 0.022	0.875± 0.035
ClaimTrees-s3d5			
ARES-0.95	**0.867± 0.171**	**0.971± 0.037**	**0.887± 0.146**
Entail-Prev	0.718± 0.090	0.936± 0.045	0.761± 0.097
Entail-Base	0.659± 0.061	0.618± 0.076	0.610± 0.091
ROSCOE-LI-Self	0.497± 0.044	0.500± 0.242	0.460± 0.074
ROSCOE-LI-Source	0.513± 0.117	0.514± 0.077	0.340± 0.081
ReCEval-Intra	0.500± 0.000	0.100± 0.054	0.161± 0.074
ReCEval-Inter	0.550± 0.070	0.539± 0.050	0.356± 0.083
LLM-Judge	0.774± 0.178	0.942± 0.057	0.796± 0.169
ClaimTrees-s5d3			
ARES-1	**0.875± 0.217**	0.889± 0.232	**0.880± 0.223**
ARES-0.95	0.867± 0.217	**0.889± 0.232**	0.875± 0.222
Entail-Prev	0.767± 0.181	0.873± 0.223	0.799± 0.191
Entail-Base	0.824± 0.205	0.700± 0.149	0.729± 0.167
ROSCOE-LI-Self	0.500± 0.000	0.055± 0.033	0.097± 0.052
ROSCOE-LI-Source	0.650± 0.054	0.560± 0.031	0.380± 0.073
ReCEval-Intra	0.500± 0.000	0.055± 0.033	0.097± 0.052
ReCEval-Inter	0.594± 0.095	0.539± 0.043	0.357± 0.063
LLM-Judge	0.742± 0.192	0.868± 0.222	0.770± 0.201
ClaimTrees-s5d5			
ARES-1	**0.900± 0.163**	**0.990± 0.017**	**0.920± 0.139**
ARES-0.95	**0.900± 0.163**	**0.990± 0.017**	**0.920± 0.139**
Entail-Prev	0.723± 0.096	0.969± 0.018	0.783± 0.095
Entail-Base	0.692± 0.141	0.597± 0.067	0.610± 0.083
ROSCOE-LI-Self	0.481± 0.020	0.446± 0.018	0.462± 0.010
ROSCOE-LI-Source	0.578± 0.063	0.533± 0.027	0.321± 0.055
ReCEval-Intra	0.500± 0.000	0.053± 0.019	0.094± 0.031
ReCEval-Inter	0.584± 0.097	0.534± 0.059	0.310± 0.084
LLM-Judge	0.847± 0.140	0.951± 0.082	0.881± 0.111

Table A8: **GPT-4o-mini (ClaimTrees)** ARES differs from other methods in deeper trees instead of wider trees. s3d5 means trees with 3 sources and depth of 5.

Dataset / Method	Recall	Precision	F1
ClaimTrees-v5i1			
ARES-1	0.985± 0.014	0.950± 0.046	0.965± 0.032
ARES-0.95	0.990± 0.022	0.998± 0.005	0.994± 0.015
Entail-Prev	0.992± 0.011	0.974± 0.038	0.982± 0.026
Entail-Base	0.900± 0.027	0.788± 0.030	0.813± 0.038
ROSCOE-LI-Self	0.975± 0.009	0.918± 0.025	0.942± 0.019
ROSCOE-LI-Source	0.690± 0.062	0.626± 0.038	0.545± 0.058
ReCEval-Intra	0.500± 0.000	0.100± 0.000	0.167± 0.000
ReCEval-Inter	0.755± 0.047	0.671± 0.021	0.590± 0.066
LLM-Judge	**1.000± 0.000**	**1.000± 0.000**	**1.000± 0.000**
ClaimTrees-v5i2			
ARES-1	**1.000± 0.000**	**1.000± 0.000**	**1.000± 0.000**
ARES-0.95	0.995± 0.011	0.998± 0.005	0.996± 0.008
Entail-Prev	0.990± 0.010	0.981± 0.019	0.985± 0.015
Entail-Base	0.863± 0.009	0.823± 0.007	0.815± 0.013
ROSCOE-LI-Self	0.965± 0.030	0.951± 0.036	0.956± 0.033
ROSCOE-LI-Source	0.635± 0.054	0.642± 0.058	0.555± 0.057
ReCEval-Intra	0.500± 0.000	0.167± 0.000	0.250± 0.000
ReCEval-Inter	0.695± 0.029	0.721± 0.020	0.594± 0.038
LLM-Judge	0.978± 0.016	0.960± 0.028	0.967± 0.024
ClaimTrees-v5i5			
ARES-1	0.988± 0.028	0.991± 0.020	0.989± 0.026
ARES-0.95	**0.998± 0.004**	**0.998± 0.005**	**0.998± 0.005**
Entail-Prev	0.988± 0.013	0.990± 0.010	0.989± 0.011
Entail-Base	0.930± 0.019	0.950± 0.012	0.936± 0.018
ROSCOE-LI-Self	0.938± 0.012	0.955± 0.008	0.943± 0.012
ROSCOE-LI-Source	0.661± 0.006	0.736± 0.033	0.649± 0.011
ReCEval-Intra	0.500± 0.000	0.278± 0.000	0.357± 0.000
ReCEval-Inter	0.665± 0.024	0.826± 0.008	0.642± 0.034
LLM-Judge	0.982± 0.017	0.983± 0.017	0.982± 0.017

Table A9: **GPT-4o-mini (ClaimTrees)** ARES does not differ much from other methods in inserted errors that do not affect downstream reasoning. v5i2 means 5 valid claims and 2 inserted claims.