Online Learnability of Chain-of-Thought Verifiers: Soundness and Completeness Trade-offs

Maria-Florina Balcan [1] **Avrim Blum** [2] **Kiriaki Fragkia** [1] **Zhiyuan Li** [2] **Dravyansh Sharma** [2 3]

Abstract

Large Language Models (LLMs) with chain-of-thought generation have demonstrated great potential for solving complex reasoning and planning tasks, though their output is not fully reliable and requires careful verification. Using learned verifiers can help increase trust, enforce safety constraints, and ensure alignment with personal preferences, yet training them is challenging because interactions between generator and verifier may induce substantial distribution shift. Motivated by this challenge, we propose a framework for online learning chain-of-thought verifiers that, given a problem and a sequence of reasoning steps, check the correctness of the solution. Highlighting the asymmetric role of soundness errors (not flagging incorrect reasoning) and completeness errors (rejecting correct reasoning), we introduce novel extensions of the Littlestone dimension which tightly characterize mistake bounds in the realizable setting. We provide optimal algorithms for finding the Pareto-frontier (the smallest total number of mistakes given a budget of soundness mistakes) as well as for minimizing a linear combination of mistake costs. We further show how our learned verifiers can be used to boost the accuracy of a collection of weak generators: under the mild assumption that one of the generators can generate the next reasoning step correctly with some minimal probability, we show how to learn a strong generator with small error and abstention rates.

[1]Carnegie Mellon University [2]Toyota Technological Institute at Chicago [3]Northwestern University. Correspondence to: Maria-Florina Balcan, Kiriaki Fragkia <{ninamf, kiriakif}@cs.cmu.edu>, Avrim Blum, Zhiyuan Li, Dravyansh Sharma <{avrim,zhiyuanli,dravy}@ttic.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Large language models (LLMs) are rapidly being deployed as decision-making tools in domains ranging from scientific discovery to healthcare, law, and education. While LLMs have demonstrated impressive reasoning capabilities, their reliability degrades as reasoning chains grow longer: errors in intermediate steps can accumulate and propagate, leading to plausible but incorrect conclusions that are difficult to detect. Beyond correctness, as LLMs are increasingly deployed for autonomous decision-making—including agent planning, tool use, and interactive systems (Shinn et al., 2023)—verifying the reasoning in order to ensure that it does not produce unsafe or harmful intermediate actions becomes a critical requirement. Verification also plays a key role in aligning LLM reasoning with user preferences, ethical constraints, and domain-specific objectives, where correctness is not solely a property of the final answer but of the entire reasoning process (Ji et al., 2023). More broadly, the need for reliable verification extends beyond LLM reasoning: recent deployments of automated proof checkers at major venues such as STOC 2026 and ICML 2026 highlight the growing demand for tools that can assess the validity of complex arguments. These developments underscore that learning high-quality verifiers is a problem of both practical and scientific importance.

In practice, augmenting LLMs with learned verification models has already led to dramatically stronger reasoning systems, contributing to breakthroughs such as gold-level performance at the 2025 International Mathematical Olympiad (Shao et al., 2025) and solutions to open-ended mathematical research problems (Feng et al., 2026). These verifiers are used at inference time for best-of-N reranking and tree search (Snell et al., 2025) and during training as reward signals for reinforcement learning (Setlur et al., 2025; Cui et al., 2025). Despite this practical success, the development of learned verifiers has

been largely empirical. Fundamental questions remain unanswered: what are the tight learnability guarantees for chain-of-thought verifiers? What is the optimal tradeoff between accepting incorrect reasoning and rejecting correct reasoning? And can learned verifiers provably improve the accuracy of the underlying reasoning system?

In this work, we aim to close this clear theoretical gap. We propose a new framework that treats verification as an inherently interactive and sequential problem. We model the verifier as an online learner that observes a sequence of (problem statement, reasoning trace) pairs and is tasked to determine whether each step in the reasoning trace is correct. Crucially, the verifier must operate under minimal assumptions on the sequence of reasoning traces, allowing for fully adaptive and potentially adversarial interactions. This is a remarkable shift in perspective from prior work (Balcan et al., 2025) which initiated the study of chain-of-thought verification through the lens of statistical learning. We argue that this framework is fundamentally limited in capturing the behavior of modern reasoning systems, where reasoning traces are generated adaptively—often conditioned on prior feedback from the verifier—and may deviate significantly from any fixed training distribution. Our approach enables us to reason about verification in settings where the data distribution is shaped by the behavior of the underlying reasoning system, and where robustness to distribution shift is not an auxiliary consideration but a central requirement.

Our work introduces several conceptual advances toward a principled theory of learning verifiers for natural language reasoning. First, to overcome the challenges caused by the *interactive nature of verification*, which is typically coupled with adaptive reasoning systems, we adopt an online learning model. Second, our analysis involves *asymmetric error costs*, explicitly distinguishing between *soundness errors* (accepting incorrect reasoning) and *completeness errors* (rejecting correct reasoning). This distinction captures a fundamental asymmetry in high-stakes applications, where undetected errors can have severe consequences, while overly conservative verification can often be mitigated through iterative refinement. Finally, we show concretely how learned verifiers can influence and improve downstream reasoning processes. Conceptually, we train the verifier agent with the help of the frozen pre-trained reasoning agent (LLM) and the end user (human); then at the inference time we show a system where the pre-trained LLM agent works together with the trained verifier to achieve highly accurate reasoning for the user for which the verifier has been trained. Prior work (Rohatgi et al., 2026) has shown the usefulness of process verifiers, but without actually showing whether or how a verifier may be learned. In contrast, we can show provable guarantees that translate to exponential improvement in the ability of a reasoner to generate correct proofs.

Building on this framework, we make several technical contributions. Our initial focus is to study the *online learnability of chain-of-thought verifiers* while distinguishing between soundness and completeness verification errors. In an abstract sense, there is a direct tension between these two types of errors: if we want to ensure the soundness of a reasoning trace, we must also be overly conservative, rejecting any reasoning step we are even slightly uncertain of. We formalize this tradeoff through a natural model that interpolates between two extremes: learning a perfectly sound verifier and permitting both types of errors. We refer to this model as *verification with a soundness budget*. In this setting, the learner's goal is to minimize the number of completeness mistakes while adhering to a given budget for soundness mistakes. If the budget is zero, then the learner must guarantee that any accepted reasoning trace is indeed correct. In this model, we ask the following fundamental learnability questions:

(a) Is there an online chain-of-thought verification algorithm that minimizes the number of completeness mistakes, while adhering to a soundness mistake budget?
(b) Is there a natural notion of dimension that characterizes learnability in this setting?[1]

We fully resolve these questions in this work. We characterize the optimal tradeoff between soundness and completeness mistakes by introducing a new combinatorial complexity measure, the *soundness-completeness Littlestone dimension*, and show that it precisely governs the optimal online learnability of chain-of-thought verifiers. We provide matching upper and lower bounds, establishing that our guarantees are tight and instance-independent. We provide a clean proof of this result via an equivalence between two natural verification models: *chain-of-thought verification*, where the learner must provide the location of the first faulty reasoning step, and *prefix verification*, where the learner must only check the correctness of the last step in the prefix of a reasoning trace. This two-way reduction allows us to analyze the simpler prefix verification setting while ensuring that the tight bounds we provide directly apply to chain-of-thought verification as well.

[1] We also pose and answer similar questions for another natural model in which the learner aims to minimize the weighted combination of the number of soundness and the number of completeness mistakes.

The main contribution of this work is not only to establish tight learnability results for chain-of-thought verifiers, but also to understand their consequences for generation. In particular, we ask the following deeper question:

How can tight performance guarantees of a verifier of chain-of-thought reasoning translate to improved reliability of an LLM generator?

In our work, we analyze the interaction between verifiers and reasoning systems, showing that even when the underlying generator produces correct reasoning only with small probability, a learned verifier can be used to *exponentially* amplify its reliability—yielding quantifiable and substantially stronger correctness guarantees across a broader class of problems. Our result builds on and extends Littlestone's mistake-bound-to-PAC argument (Littlestone, 1989) to our setting, where samples are drawn from the stochastic process induced by the dynamic interaction between the generator and verifier. Our algorithm uses a verifier to combine proof strategies from multiple generators, enabling the generation of solutions for potentially completely new problems. We further show how the error and abstention rates of the verifier-assisted generator can be bounded in terms of the soundness and completeness mistake bounds of the verifier. In particular, the rate of incorrect proof generation is governed by the verifier's soundness, justifying stronger emphasis on limiting soundness mistakes.

Related Work. We summarize below the research most relevant to our work and relegate further background and discussion of related research to Appendix C.1.

Balcan et al. (2025) give learning-theoretic foundations for chain-of-thought verification from a statistical (PAC) perspective, showing that learning a verifier with bounded overall error requires sample complexity linear in the VC dimension, but learning a *perfectly sound* verifier requires $\Omega(|H|)$ samples—an exponential gap that motivates the soundness-completeness tradeoff studied in our work. A parallel line of recent theory studies when and how post-training can improve reasoning. The *sharpening* framework (Huang et al., 2025) and the *coverage principle* (Chen et al., 2026) show that sample-and-evaluate approaches (such as best-of-N) require the base model to place non-negligible probability mass on entirely correct sequences, and that this *sequence-level* coverage governs the success of post-training and test-time scaling. Tsilivis et al. (2025) provide a complementary account of why reinforcement learning after next-token prediction can unlock longer chain-of-thought behaviors, though it still relies on correct trajectories not being exponentially rare. These results operate at the *sequence level*: they analyze selection or reweighting of complete reasoning traces. In contrast, our work and Rohatgi et al. (2026) take a *step-level* perspective. Rohatgi et al. (2026) show how to use a (given) process verifier to guide backtracking via a random walk on partial reasoning traces. Our work addresses the more fundamental question of how to *learn* such a step-level verifier, with formal guarantees that hold for arbitrary sequences of reasoning traces—whether generated by a fixed prover, an evolving prover undergoing RL training, or an adversary.

Structure of the Paper. In the main body, we provide an overview of our main results along with descriptions of our proposed algorithmic procedures and proof techniques. In Appendix A, we present our main results regarding learning chain-of-thought verifiers in full detail. Specifically, Appendix A.2 along with Appendix A.3 complete the proof of Theorem 3.5, whereas Appendices A.2, A.4 complete the proof of Theorem 3.7. We devote Appendix B to proving Theorem B.1. Finally, we close with a discussion and open questions in Section 5.

2. Formal Definition of CoT Verification

We begin with a formal introduction to our model.

Problem statements, reasoning traces, and verifiers. Let X denote a set of problem statements written in natural language. For example, a problem statement could be a mathematical problem such as "Find $y \geq 1$, such that $\sqrt{y-1} = 4$", a list of contributions claimed to be solved in a research paper, or a prompt such as "Please give me a list of instructions for booking my conference trip". Let Σ denote a set of reasoning steps, defined broadly as a sequence of tokens, such as "We first square both sides of the equation: $\left(\sqrt{y-1}\right)^2 = 4^2$" or "Book a United flight from Chicago to Sydney from 12/5 to 12/12". A reasoning trace is a finite sequence of reasoning steps $\tau \in \Sigma^*$. We are interested in using a learned verification model that verifies these steps against some desirable properties such as mathematical correctness, safety, or preference alignment. For example, when designing a plan for a conference trip, the itinerary should match the user's stated preferences over airlines and hotels.

A verifier is a function $h : X \times \Sigma^* \to \{\text{YES}, \text{NO}\}$ that, given a problem statement x and a finite sequence of reasoning steps $\tau = (\tau_1, \tau_2, \ldots, \tau_\ell)$, outputs YES if τ_ℓ is a valid inference from x and the preceding steps $\tau_{1:\ell-1}$, assuming the prefix $\tau_{1:\ell-1}$ is correct. Otherwise, h outputs NO. In the case where $\tau_{1:\ell-1}$

contains a faulty reasoning step, we allow h to output anything on (x, τ_ℓ). Given a problem statement and a reasoning trace $(x, \tau = (\tau_1, \tau_2, \ldots, \tau_L))$, we evaluate the correctness of the reasoning trace by "running" a verifier $h \in H$ on every prefix: $h(x, (\tau_1))$, $h(x, (\tau_1, \tau_2)), \ldots h(x, (\tau_1, \tau_2, \ldots \tau_L))$. We say that h *accepts* (x, τ) if $h(x, \tau_{1:\ell}) = \text{YES}$ for all $\ell \in [L]$. Else, we say that h *rejects* (x, τ) and outputs $\min\{\, \ell \in [L] : h(x, \tau_{1:\ell}) = \text{NO} \,\}$ as the location of the first incorrect step.

This framework relies on the following natural assumption ("inductive bias" in machine learning): the correctness of each reasoning step can be checked *locally* given the context of the problem statement and preceding steps. Concretely, we assume that there exists a target (unknown) verifier h^* in a class of verifiers H that always agrees with an oracle $\mathcal{O} : X \times \Sigma^* \mapsto \{\text{YES}, \text{NO}\}$ (e.g. a human expert) about the validity of each reasoning step.

Remark 2.1. We follow the formalization of Balcan et al. (2025), where the goal is to emulate the behavior of the target verifier $h^* \in H$. As stated, this formalization is agnostic to certifying that each problem statement makes progress in answering the given problem statement, which may be important in some applications (e.g. verifying mathematical proofs) more than others (e.g. verifying that no action in a plan is unsafe). Later we discuss how this issue can be addressed under additional assumptions on the human oracle (Remark 4.3).

Online chain-of-thought verification. We study an online chain-of-thought verification setting, in which a learner is asked to verify an arbitrary sequence of problem statements and reasoning traces over T timesteps. At each timestep $t \in [T]$ the learner observes a pair $(x^{(t)}, \tau^{(t)})$ that consists of a problem statement $x^{(t)} \in X$ and a reasoning trace $\tau^{(t)} \in \Sigma^L$[2]. The learner must decide whether $\tau^{(t)}$ is a correct reasoning trace for $x^{(t)}$, and if not, identify the location of the first incorrect step. Formally, at timestep t, the learner outputs a label $\hat{y}^{(t)}$ from a totally ordered label set $\mathcal{Y} = \{1, \ldots, L\} \cup \{\infty\}$, where $1 < 2 < \cdots < L < \infty$. Returning $i \in [L]$ indicates that the location of the first faulty step is i, whereas returning "∞" indicates that all steps in the reasoning trace are correct. After the learner predicts, the oracle reveals the true label $y^{(t)} = \min\big(\{\, i \in [L] : \mathcal{O}(x^{(t)}, \tau^{(t)}_{1:i}) = \text{NO} \,\} \cup \{\infty\}\big)$.

We can think of the learner playing according to an online verification algorithm, $V_H : \big((X \times \Sigma^L) \times \mathcal{Y}\big)^* \times (X \times \Sigma^L) \to \mathcal{Y}$ which, given the history so far and a new example, uses a class of verifiers, H, to output a prediction. We say that an online verification algorithm makes a mistake in round t if $\hat{y}^{(t)} \neq y^{(t)}$. Throughout, we distinguish between the following two types of mistakes:

1. **Soundness mistake**, which occurs when the learner predicts the location of the first faulty step "too late". Formally, the learner makes a soundness mistake in round t if $\hat{y}^{(t)} > y^{(t)}$.
2. **Completeness mistake**, which occurs when the learner predicts the location of the first faulty step "too early". Formally, the learner makes a completeness mistake in round t if $\hat{y}^{(t)} < y^{(t)}$.

Online Prefix Verification. Parts of our technical analysis (e.g., interaction between prover and verifier in Section 4) require online verification algorithms that can check a reasoning trace step-by-step. We term this online verification problem *online prefix verification*. At a high-level, in prefix verification, the learner observes a sequence of *prefixes* (partial reasoning traces) and is asked to verify whether the last step of each prefix is a valid inference from the previous reasoning steps. The learner then makes a *prefix*-soundness mistake in the case of false-positive and a *prefix*-completeness mistake in the case of false-negative.

Formally, the learner at timestep $t \in [T]$ is presented with an instance $z^{(t)}_{\text{pfx}} = (x^{(t)}, \tau^{(t)}_{1:\ell}) \in X \times \Sigma^{\leq L}$, with the promise that steps $\tau_{1:\ell-1}$ are correct, that is, $\mathcal{O}(x^{(t)}, \tau^{(t)}_{1:i}) = \text{YES}$ for all $i \in [\ell - 1]$. The learner must then decide whether step τ_ℓ is correct or not and predict $\hat{y}^{(t)}_{\text{pfx}} = \text{YES}$ or $\hat{y}^{(t)}_{\text{pfx}} = \text{NO}$ accordingly. Then the true label $y^{(t)}_{\text{pfx}}$ is revealed to the learner such that $y^{(t)}_{\text{pfx}} = \text{YES}$ if and only if $\mathcal{O}(x^{(t)}, \tau^{(t)}_{1:\ell}) = \text{YES}$ and $y^{(t)}_{\text{pfx}} = \text{NO}$ otherwise. In this setting, we will distinguish between the following two types of mistakes:

1. **(Prefix) soundness mistake**, which occurs when the learner accepts an incorrect prefix. Formally, the learner makes a prefix soundness mistake at round t if $\hat{y}^{(t)}_{\text{pfx}} = \text{YES}$ but $y^{(t)}_{\text{pfx}} = \text{NO}$.
2. **(Prefix) completeness mistake**, which occurs when the learner rejects a correct prefix. Formally, the learner makes a prefix completeness mistake at round t if $\hat{y}^{(t)}_{\text{pfx}} = \text{NO}$ but $y^{(t)}_{\text{pfx}} = \text{YES}$.

One may view prefix verification as a structured binary classification problem, where the "domain"

[2]For simplicity, we assume that all reasoning traces observed by the learner will have length exactly L. Our results also extend to variable length reasoning traces (up to some finite upper bound L).

consists of all prefixes of reasoning traces, where all but the last reasoning step are correct. However, this "domain" depends on the target function and would therefore be unknown to the learner.

3. Online Chain-of-Thought Verification

To build intuition and highlight the challenge of guaranteeing soundness, we study mistake bounds for finite classes of verifiers. We show an exponential separation between arbitrary online verification algorithms, which may incur both soundness and completeness errors, and algorithms that are constrained to be sound (i.e., never make a soundness mistake).

Proposition 3.1. *For any finite class of verifiers, H, there exists an online chain-of-thought verification algorithm that makes at most $O(\log |H|)$ mistakes. Moreover, there exists a class of verifiers, H', for which any sound chain-of-thought verification algorithm cannot make fewer than $|H'| - 1$ mistakes.*

Formal arguments are presented in Section A.1, where we further show that these bounds are tight. We note that depending on H, it may be possible to obtain sharper mistake bounds for learning sound verifiers (see Example A.3 and further concrete examples in Appendix C.4).

Proposition 3.1 highlights a sharp transition and motivates a more refined study of the trade-off between soundness and completeness mistakes.

3.1. Chain-of-Thought Verification with a Soundness Mistake Budget

We will be interested in analyzing the trade-off between soundness and completeness mistakes in chain-of-thought verification when the learner is allowed to make a small number of soundness mistakes, $k \in \mathbb{N}$. Specifically, the learner's objective is to minimize the number of overall mistakes subject to adhering to the soundness mistake budget, k. As motivation, we observe that even with $k = 1$, the verifier class used to show the $\Omega(|H|)$ lower bound in Proposition 3.1 can now be learned with no more than 1 mistake overall.

We now introduce a new variant of the familiar notion of mistake trees from online learning (*cf.* Appendix C.2 for relevant background). Our definitions are inspired by those in online learning with abstensions (Sayedi et al., 2010; Zhang & Chaudhuri, 2016).

Soundness-Completeness (SC)-mistake tree. An SC-mistake tree is a rooted binary tree, where internal nodes are examples in $X \times \Sigma^{\leq L}$. Every internal node, z, has exactly two children: z_s and z_c. There are two types of edges: (z, z_s) is a straight edge labeled NO and (z, z_c) is a curvy edge labeled YES. We say that an SC mistake tree is *shattered* by a hypothesis class H if for any root-to-leaf path that traverses nodes $z_1, z_2, z_3, \ldots, z_d$ there exists a hypothesis $h \in H$ such that the label of edge (z_i, z_{i+1}) is $h(z_i)$ for all $i \leq d$.

Definition 3.2. We say that an SC mistake tree is (k, m)-difficult if every root-to-leaf path that has at most k straight edges is of length at least m.

For example, Figure 1 shows a construction of an SC-mistake tree that is $(0, 2)$- and $(1, 1)$-difficult.

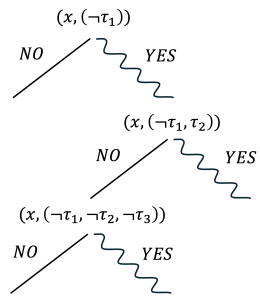

Figure 1. An SC-mistake tree for $H = \left\{ h_i : \{x\} \times \{0,1\}^4 \to \{0,1\} \;\middle|\; i \in [4],\ h_i(\tau) = \tau_i \wedge \bigwedge_{j \in [4] \setminus \{i\}} \neg\tau_j \right\}$. Let $X = \{x\}$ and $\Sigma = \{0, 1\}$, interpreting the output 0 as NO and 1 as YES.

Definition 3.3 (SC-Littlestone dimension)**.** For a class of verifiers, H, and a soundness mistake budget, $k \in \mathbb{N}$, the SC-Littlestone dimension is defined as $\text{SC-Ldim}(H, k) := \sup\{m \in \mathbb{N} :$ There exists a (k, m)-difficult mistake tree for $H\}$.

If $H = \varnothing$, then $\text{SC-Ldim}(H, k) = 0$ for all $k \in \mathbb{N}$. If for every $m \in \mathbb{N}$ there exists a (k, m)-difficult mistake tree for H, then we let $\text{SC-Ldim}(H, k) = \infty$. We note that for a finite class of verifiers, we have the following bound on the SC-Littlestone dimension.

Proposition 3.4. *For a finite class of verifiers, H with $|H| = n$, we have that $\text{SC-Ldim}(H, 0) \leq n - 1$ and more generally $\text{SC-Ldim}(H, k) \leq O\left(n^{1/(k+1)}\right)$. For $k \geq \lfloor \log_2 n \rfloor - 1$, we have that $\text{SC-Ldim}(H, k) = \lfloor \log_2 n \rfloor$.*

The proof of Proposition 3.4 involves showing that $\text{SC-Ldim}(H, k) \leq \max\left\{ m \in \mathbb{N} : \sum_{i=0}^{k+1} \binom{m}{i} \leq n \right\}$ and follows similarly to that of Zhang & Chaudhuri (2016). In this section's main result, we show that the SC-Littlestone dimension tightly characterizes learnability of chain-of-thought verifiers with a soundness mistake budget.

Theorem 3.5. *Consider a class of verifiers, H, a soundness mistake budget $k \in \mathbb{N}$ and $m \in \mathbb{N}$ such that $\text{SC-Ldim}(H, k) = m$. Then the following hold:*

(a) *There exists an online chain-of-thought verification algorithm that guarantees at most k soundness mistakes and at most m mistakes overall on any realizable sequence of examples.*
(b) *For any deterministic chain-of-thought verification algorithm that guarantees at most k soundness mistakes, there exists a realizable sequence of examples that forces the algorithm to make at least m mistakes overall.*

Algorithmic approach. Our algorithm's prediction rule (Algorithm 9) is driven by the SC-Littlestone dimension. Specifically, given a problem-trace pair $(x, (\tau_1, \ldots, \tau_L))$, the learner examines each prefix, $(\tau_1, \ldots, \tau_\ell)$ for $\ell \in [L]$. For each step ℓ, the learner uses the SC-Littlestone dimension as a potential function in order to compare the worst-case residual difficulty of the two possible actions: (1) certifying that the prefix up to step ℓ is correct, or (2) declaring that the trace's ℓ-th step is faulty. If wrong, prediction (2) would preserve the current soundness budget k (completeness mistake), while prediction (1) would consume one unit of budget (soundness mistake). Accordingly, at each step ℓ of the reasoning trace, the algorithm evaluates the SC-Littlestone dimension with budget k of the subspace of hypotheses that predict YES at step ℓ and the SC-Littlestone dimension with budget $k-1$ of the subspace of hypotheses that predict NO at step ℓ. The algorithm predicts error at the first index where the former option is at least as favorable as the latter; if no such index exists, it predicts ∞. When $k = 0$, the learner can no longer afford predicting a faulty step "too late", and therefore defaults to predicting the first position at which some hypothesis predicts failure.

After the index of the first faulty step is revealed, the learner updates the version space in the natural way. If it predicted a failure too late (soundness mistake), then the learner restricts to hypotheses that reject the true failing prefix and decrements the remaining soundness budget. If it predicted failure too early (completeness mistake), then the learner restricts to hypotheses that accept that prefix.

Proof Overview. Our main technical insight for the upper bound is an *online-to-online reduction* (Theorem A.4) from online chain-of-thought verification to online *prefix* verification (summarized in Algorithm 1). We then construct an online prefix verification algorithm (Algorithm 3) and prove a mistake bound based on the SC-Littlestone dimension (Theorem A.8). Theorems A.4 and A.8 establish Theorem 3.5(a).

Surprisingly, we find that in the case of a completeness mistake the online prefix verification algorithm need not be updated on the location of the first faulty step. This would suggest that online prefix verification is a strictly harder problem than chain-of-thought verification. Nonetheless, the optimality of our approach comes from showing an online-to-online reduction in the opposite direction. Namely, we show that (under a mild assumption) an online chain-of-thought verification algorithm can be transformed to an online prefix verification algorithm (Theorem A.5). This together with a lower SC-Littlestone dimension based lower bound on prefix verification (Theorem A.7) establishes the tight lower bound of Theorem 3.5(b).

3.2. Chain-of-Thought Verification with a Linear Cost Objective

In this section, we study the trade-off between soundness and completeness mistakes via a linear cost objective, motivated by downstream applications that may have a well-defined relative cost for each type of mistake. Specifically, in this model the learner incurs a cost of $\gamma_s \in \mathbb{R}_{\geq 0}$ for every soundness mistake and cost $\gamma_c \in \mathbb{R}_{\geq 0}$ for every completeness mistake. The learner's objective is to minimize the linear cumulative cost $\gamma_s M_s + \gamma_c M_c$, where M_s, M_c denotes the number of soundness and completeness mistakes (respectively) that the learner makes over T timesteps.

We now introduce the notion of a WSC-mistake tree that augments the SC-mistake tree definition by including weighted edges designed to handle the *real-valued* cost of each mistake.

Weighted Soundness-Completeness (WSC)-mistake tree. A WSC-mistake tree is a rooted binary tree, where internal nodes are examples in $X \times \Sigma^{\leq L}$. Every internal node, z, has exactly two children: z_s and z_c. There are two types of edges: (z, z_s) is a straight edge labeled NO and has weight $w(z, z_s) = \gamma_s$, whereas (z, z_c) is a curvy edge labeled YES and has weight $w(z, z_c) = \gamma_c$. We say that a WSC mistake tree is *shattered* by a hypothesis class H if for any root-to-leaf path that traverses nodes $z_1, z_2, z_3, \ldots, z_d$ there exists a hypothesis $h \in H$ such that the label of edge (z_i, z_{i+1}) is $h(z_i)$ for all $i \leq d$.

Definition 3.6 (WSC-Littlestone dimension)**.** For a class of verifiers, H, soundness mistake cost $\gamma_s \in \mathbb{R}_{\geq 0}$ and completeness mistake cost $\gamma_c \in \mathbb{R}_{\geq 0}$, the WSC-Littlestone dimension, WSC-Ldim(H, γ_s, γ_c)[3], is defined as the supremum $W \in \mathbb{R}$ such that there exists a WSC mistake tree, where all root-to-leaf paths have cumulative weight at least W.

If $H = \varnothing$, then WSC-Ldim$(H) = 0$. If for all $W \in \mathbb{R}_{\geq 0}$ there exists a WSC-mistake tree shattered by H, where

[3]For brevity, in the remainder of the paper we drop the dependence on γ_s, γ_c and write WSC-Ldim(H) instead.

all root-to-leaf paths have weight at least W, then we say that $\text{WSC-Ldim}(H) = \infty$.

We show that the WSC-Littlestone dimension characterizes the optimal cumulative loss among all deterministic chain-of-thought verification algorithms, which is the main result of this section.

Theorem 3.7. *Consider a class of verifiers, H, a soundness mistake cost $\gamma_s \in \mathbb{R}_{\geq 0}$ and a completeness mistake cost $\gamma_c \in \mathbb{R}_{\geq 0}$. Then the following hold:*

(a) There exists an online chain-of-thought verification algorithm that achieves at most WSC-Ldim(H) cumulative loss on any realizable sequence of examples.
(b) For any deterministic chain-of-thought verification algorithm there exists a realizable sequence of examples such that the algorithm will suffer cumulative loss at least W for every $W <$ WSC-Ldim(H).

In Sections A.2 and A.4, we provide a formal proof that is in the same spirit as prior results in online realizable regression (Attias et al., 2023). We summarize our algorithmic procedure in Algorithm 10.

4. Improving Weak Provers with Online Verification

We now present an application of our online verification guarantees to the problem of improving a weak prover, or even a weak collection of provers, into a highly-accurate prover. For this section, we assume our verifier-learning algorithm operates in the prefix verification model, where given a prefix $\tau_{1:\ell}$ such that $\tau_{1:\ell-1}$ is correct, it must output whether or not τ_ℓ is correct. Underlying these results is a generalization of classic mistake-bound-to-PAC conversions that involves repeated applications of the online algorithm to variations of a given instance and crucially uses the fact that our online learning guarantees apply to arbitrary sequences and not just to iid data.

Formally, we define a prover P as follows. For any problem instance $x \in X$, P produces a first step τ_1 of a reasoning trace from some distribution $P(x)$. Given x and τ_1, P produces a second step τ_2 from some distribution $P(x, \tau_1)$. In general, given an instance x and a partial reasoning trace $\tau_1, ..., \tau_\ell$ for $\ell < L$, P selects a next step $\tau_{\ell+1}$ from a distribution $P(x, \tau_1, ..., \tau_\ell)$.

We say that prover P is "α-good" for an instance x if it has at least an α probability of producing a correct next reasoning step conditioned on the previous steps being correct, where correctness is defined with respect to a labeling oracle $\mathcal{O}$ which *is the same as* the labeling oracle used for training the verifier. Formally,

Definition 4.1 (α-good prover for a problem instance with respect to oracle $\mathcal{O}$)**.** We say that a prover P is *α-good* for a problem instance $x \in X$ if for all $0 \leq \ell < L$, for all correct partial reasoning traces $\tau_1, ..., \tau_\ell$ (i.e., $\mathcal{O}(x, \tau_{1:\ell'}) = \text{YES}$ for all $\ell' \leq \ell$) produced by P, we have $\Pr_{\tau_{\ell+1} \sim P(x, \tau_1, ..., \tau_\ell)}[\mathcal{O}(x, \tau_1, ..., \tau_{\ell+1}) = \text{YES}] \geq \alpha$.

This definition naturally extends to a distribution D over problem instances as follows.

Definition 4.2 ((α, γ)-good prover for a distribution with respect to correctness oracle $\mathcal{O}$)**.** We say that a prover is *(α, γ)-good* for a distribution D over problems in X, if it is α-good for at least a γ probability mass of the problems under D, that is, $\Pr_{x \sim D}[P \text{ is } \alpha\text{-good on } x] \geq \gamma$.

Remark 4.3. In the context of our results, the notion of an α-good prover must imply that given a problem $x \in X$ any correct prefix $\tau_1, ..., \tau_\ell$ generated by the prover(s) can be extended to a correct full proof $\tau_1, ..., \tau_L$. Thus, this notion is most relevant for settings where the oracle $\mathcal{O}$ is checking both for logical consistency and progress. For example, in grading a student's homework, each step of the student's argument should not be fallacious *and* should make progress towards the problem's solution. In a planning domain, each step of the plan should both be feasible and helpful. In proving a theorem, each step τ_i could be proving a lemma that is relevant to proving the result. Note that an (α, γ)-good prover would require only that a γ fraction of x's satisfy this property. It would be interesting to consider relaxed conditions that might require limited backtracking in the reasoner-verifier interaction.

Note that if prover P is α-good for instance x, then it still might have only an α^L probability of producing a fully correct reasoning trace for x. This is exponentially small even for (say) $\alpha = 1/2$. Nonetheless, this will still be sufficient for us to "boost" it to a highly-accurate prover using a learned verifier.

In fact, we can weaken our condition further by considering a *set* of provers $\mathcal{P} = \{P_1, ..., P_k\}$ to be α-good for a problem instance x if, for each step ℓ, *at least one* of the provers $P_i \in \mathcal{P}$ has probability $\geq \alpha$ of producing a correct next step, with perhaps different provers for different steps. For instance, this could model a setting where solving an instance x requires first doing some calculus, then a proof by induction, and then a geometric argument, where different provers specialize in different kinds of reasoning. Formally,

Definition 4.4 (good *set* of provers)**.** We say that a *set* of provers $\mathcal{P} = \{P_1, ..., P_k\}$ is *α-good* for a problem instance $x \in X$ with respect to correctness oracle $\mathcal{O}$ if for all $0 \leq \ell < L$, for all correct partial reasoning traces $\tau_1, ..., \tau_\ell$ (i.e., $\mathcal{O}(x, \tau_{1:\ell'}) = \text{YES}$ for all

$\ell' \leq \ell$) generated by $\mathcal{P}$, there exists $i \in [k]$ such that $\Pr_{\tau_{\ell+1} \sim P_i(x, \tau_1, ..., \tau_\ell)}[\mathcal{O}(x, \tau_1, ..., \tau_{\ell+1}) = \text{YES}] \geq \alpha$. $\mathcal{P}$ is (α, γ)-good for distribution D if it is α-good for at least a γ probability mass of problems under D.

We now present the main result of this section and outline our algorithmic procedure and proof techniques. Formal arguments are presented in Section B.

Theorem 4.5. *(Informal Theorem B.1) Under the assumption that $\mathcal{O} = h^* \in H$ (there exists a perfect verifier in H), we can use an online verifier-learning algorithm V_H to take an (α, γ)-good set of provers $\mathcal{P}$ and, using a limited number of draws of problem instances $x \sim D$ and a limited number of calls to the labeling oracle $\mathcal{O}$, learn a highly-accurate prover $\mathsf{Wrap}(V_H, \mathcal{P})$.[4] Specifically, with probability $1 - \delta$, the prover $\mathsf{Wrap}(V_H, \mathcal{P})$ has the property that:*

1. *Given a problem instance x, $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs either "I don't know" or a proof $\boldsymbol{\tau}$.*
2. *The probability over $x \sim D$ that $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs "I don't know" is at most $(1 - \gamma) + \epsilon_c + \epsilon_s + \epsilon'$.*
3. *The probability over $x \sim D$ that $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs an incorrect proof is at most ϵ_s,*

where ϵ_c and ϵ_s are functions of the completeness and soundness mistake-bounds (respectively) of the learning algorithm V_H, and ϵ' is an additional error added in the training process. In particular, if V_H is a fully-sound verifier-learner then $\epsilon_s = 0$.

Algorithmic Approach and Proof Overview. Given an online verifier-learning algorithm V_H, a set of k provers $\mathcal{P}$, a value α such that $\mathcal{P}$ is (α, γ)-good for some $\gamma > 0$, and access to samples from D and to the labeling oracle $\mathcal{O}$ for training, the algorithm to create the prover $\mathsf{Wrap}(V_H, \mathcal{P})$ proceeds as follows.

First, as in the classic mistake-bound-to-PAC conversion of Littlestone (Littlestone, 1989), we draw two sets of examples S_1, S_2 from D, using S_1 to generate a series of hypothesis verifiers and S_2 to test and select from them. However, unlike the classic reduction, processing an example x will involve multiple calls to the verifier-learning algorithm and the labeling oracle. Specifically, given $x \in S_1$, we run all provers $P_j \in \mathcal{P}$ on x to generate candidate first-steps of a proof, and ask V_H to predict correctness of each one, testing these predictions against our labeling oracle. If any predictions were mistaken, we feed that mistake into V_H (making progress) and end the processing of this example x. If all predictions were accurate and at least one first step τ_1 was correct, we now run all provers $P_j \in \mathcal{P}$ on the trace (x, τ_1) to generate candidate second-steps, and so on. On the other hand, if all candidate first steps τ_{1j} were incorrect (or in general, all $\tau_{\ell j}$ at step ℓ were incorrect), then we simply re-run all provers in $\mathcal{P}$ to generate new candidate proof-steps, repeating this process until either we find a correct next step or we time out after $O(\frac{1}{\alpha} \log \frac{kL}{\epsilon'})$ tries, concluding that x must not have been in the good set for $\mathcal{P}$. Once we have completed processing an example, either by making a mistake, timing out, or finding a correct full proof, we then move on to the next $x \in S_1$.

Once we have completed processing S_1, we then test all verifiers produced on S_2. As usual, we may assume without loss of generality that V_H is a "conservative" algorithm, only changing its current hypothesis when a mistake is made. So, if V_H has total mistake-bound M, we have at most M hypothesis verifiers to test. One additional difference with the standard online-to-PAC conversion is that we care separately about soundness and completeness errors.

Finally, we use the final hypothesis verifier to create the prover $\mathsf{Wrap}(V_H, \mathcal{P})$ by processing each new example x as above, but without the labeling oracle (just assuming our verifier is accurate). If our verifier times out, we output "I don't know"; else we output the proof produced via this process.

5. Conclusions and Future Research

We provide the first results for learning chain-of-thought verifiers in an online fashion. We captured the asymmetric roles of soundness and completeness errors and proved tight mistake bounds via new Littlestone-style complexity measures, yielding optimal algorithms for Pareto and cost-sensitive tradeoffs. We also showed how such online verifiers can provably strengthen weak provers, and highlighted the effects of soundness and completeness errors in the downstream accuracy of the prover.

Open questions. First, it would be useful to identify conditions under which we can develop computationally efficient algorithms (while possibly sacrificing optimality). Another interesting direction is extending our results to the agnostic setting, possibly connecting with ideas from data-driven algorithm design (Sharma et al., 2019; Balcan et al., 2020; 2018; Balcan, 2021). Finally, it would be interesting to identify other conditions (beyond the (α, γ)-good prover) when verifiers can be used for "boosting" provers.

[4] In the language of Rohatgi et al. (2026), we are learning a verifier that can successfully boost $\mathcal{P}$ using action-level rejection sampling. Note however that we are only aiming to generate correct proofs, not to sample from the conditional.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbe, E., Bengio, S., Lotfi, A., Sandon, C., and Saremi, O. How far can transformers reason? the globality barrier and inductive scratchpad. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ahmadi, S., Yang, K., and Zhang, H. Strategic Littlestone dimension: Improved bounds on online strategic classification. *Advances in Neural Information Processing Systems*, 37:101696–101724, 2024.

Alon, N., Hanneke, S., Holzman, R., and Moran, S. A theory of PAC learnability of partial concept classes, 2021.

Altabaa, A., Montasser, O., and Lafferty, J. CoT information: Improved sample complexity under Chain-of-Thought supervision. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Amit, N., Goldwasser, S., Paradise, O., and Rothblum, G. N. Models that prove their own correctness. In *ICML Workshop on Theoretical Foundations of Foundation Models*, 2024.

Attias, I., Hanneke, S., Kalavasis, A., Karbasi, A., and Velegkas, G. Optimal learners for realizable regression: Pac learning and online learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, 2023.

Balcan, M.-F. Data-Driven Algorithm Design. In Roughgarden, T. (ed.), *Beyond the Worst-Case Analysis of Algorithms*, pp. 626–645. Cambridge University Press, Cambridge, 2021.

Balcan, M.-F., Dick, T., and Vitercik, E. Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

Balcan, M.-F., Dick, T., and Pegden, W. Semi-bandit optimization in the dispersed setting. In *Conference on Uncertainty in Artificial Intelligence*, pp. 909–918, 2020.

Balcan, M.-F., Blum, A., Li, Z., and Sharma, D. On learning verifiers and implications for Chain-of-Thought reasoning. *Advances in Neural Information Processing Systems*, 2025.

Chen, F., Huang, A., Golowich, N., Malladi, S., Block, A., Ash, J. T., Krishnamurthy, A., and Foster, D. J. The coverage principle: How pre-training enables post-training. *International Conference on Learning Representations (ICLR)*, 2026.

Cui, G., Yuan, L., Wang, Z., Wang, H., Li, W., He, B., Fan, Y., Yu, T., Xu, Q., Chen, W., Yuan, J., Chen, H., Zhang, K., Lv, X., Wang, S., Yao, Y., Peng, H., Cheng, Y., Liu, Z., Sun, M., Zhou, B., and Ding, N. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Daniely, A., Sabato, S., Ben-David, S., and Shalev-Shwartz, S. Multiclass learnability and the ERM principle. In *Proceedings of the 24th Annual Conference on Learning Theory*, pp. 207–232, 2011.

Delle Rose, V., Kozachinskiy, A., and Steifer, T. Effective Littlestone dimension. In *Proceedings of the 36th International Conference on Algorithmic Learning Theory (ALT)*, 2025.

Doron-Arad, I., Mehalel, I., and Mossel, E. A theory of online learning with autoregressive chain-of-thought reasoning, 2026. URL https://arxiv.org/abs/2605.06819.

Feng, G., Zhang, B., Gu, Y., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Feng, T., Trinh, T. H., Bingham, G., Hwang, D., Chervonyi, Y., Jung, J., Lee, J., Pagano, C., Kim, S.-h., Pasqualotto, F., et al. Towards autonomous mathematics research. *arXiv preprint arXiv:2602.10177*, 2026.

Filmus, Y., Hanneke, S., Mehalel, I., and Moran, S. Randomized Littlestone dimension. In *Proceedings of the 36th Conference on Learning Theory*, 2023.

Hanneke, S., Mehalel, I., and Moran, S. Sample complexity of autoregressive reasoning: Chain-of-Thought vs. end-to-end, 2026. URL https://arxiv.org/abs/2604.12013.

Hasrati, N. and Ben-David, S. On computable online learning. In *Proceedings of the 34th International Conference on Algorithmic Learning Theory (ALT)*, pp. 707–725, 2023.

Helmbold, D. P., Littlestone, N., and Long, P. M. Online learning with linear loss constraints. *Inf. Comput.*, 161(2):140–171, September 2000. ISSN 0890-5401.

Huang, A., Block, A., Foster, D. J., Rohatgi, D., Zhang, C., Simchowitz, M., Ash, J. T., and Krishnamurthy, A. Self-improvement in language models: The sharpening mechanism. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Ji, J., Liu, M., Dai, J., Pan, X., Zhang, C., Bian, C., Chen, B., Sun, R., Wang, Y., and Yang, Y. Beavertails: Towards improved safety alignment of LLM via a human-preference dataset. *Advances in Neural Information Processing Systems*, 36:24678–24704, 2023.

Joshi, N., Vardi, G., Block, A., Goel, S., Li, Z., Misiakiewicz, T., and Srebro, N. A theory of learning with autoregressive chain of thought. *Conference on Learning Theory (COLT)*, 2025.

Kim, J. and Suzuki, T. Transformers provably solve parity efficiently with chain of thought. In *International Conference on Learning Representations (ICLR)*, 2025.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Li, H., Lu, S., Chen, P.-Y., Cui, X., and Wang, M. Training nonlinear transformers for chain-of-thought inference: A theoretical generalization analysis. In *International Conference on Learning Representations (ICLR)*, 2025.

Li, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations (ICLR)*, 2024.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2:285–318, 1988.

Littlestone, N. From on-line to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, COLT '89, pp. 269–284. Morgan Kaufmann Publishers Inc., 1989.

Luo, L., Liu, Y., Liu, R., Phatale, S., Guo, M., Lara, H., Li, Y., Shu, L., Zhu, Y., Meng, L., Sun, J., and Rastogi, A. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Malach, E. Auto-regressive next-token predictors are universal learners. In *International Conference on Machine Learning (ICML)*, pp. 34417–34431. PMLR, 2024.

Merrill, W. and Sabharwal, A. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations (ICLR)*, 2024.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. In *arXiv preprint arXiv:2112.00114*, 2021.

Rohatgi, D., Shetty, A., Saless, D., Li, Y., Moitra, A., Risteski, A., and Foster, D. J. Taming imperfect process verifiers: A sampling perspective on backtracking. In *International Conference on Learning Representations (ICLR)*, 2026.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Sayedi, A., Zadimoghaddam, M., and Blum, A. Trading off mistakes and don't-know predictions. *Advances in Neural Information Processing Systems*, 23, 2010.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Shao, Z., Luo, Y., Lu, C., Ren, Z. Z., Hu, J., Ye, T., Gou, Z., Ma, S., and Zhang, X. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Sharma, D., Balcan, M.-F., and Dick, T. Learning piecewise lipschitz functions in changing environments. In *International Conference on Artificial Intelligence and Statistics*, 2019.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025.

Sridharan, K. and Rakhlin, A. Online learning with predictable sequences. In *Proceedings of the 24th Conference on Learning Theory (COLT)*, pp. 993–1019, 2011.

Suzgun, M. and Kalai, A. T. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.

Tsilivis, N., Malach, E., Ullrich, K., and Kempe, J. How reinforcement learning after next-token prediction facilitates learning. In *NeurIPS 2025 Workshop on Principles of Generative Modeling (PriGM)*, 2025.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Wang, Z., Nichani, E., Bietti, A., Damian, A., Hsu, D., Lee, J. D., and Wu, D. Learning compositional functions with transformers from easy-to-hard data. In *Conference on Learning Theory (COLT)*, 2025.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

Wen, K., Zhang, H., Lin, H., and Zhang, J. From sparse dependence to sparse attention: Unveiling how chain-of-thought enhances transformer sample efficiency. In *International Conference on Learning Representations (ICLR)*, 2025.

Zhang, C. and Chaudhuri, K. The extended Littlestone's dimension for learning with mistakes and abstentions. In *Conference on Learning Theory*, pp. 1584–1616, 2016.

A. Online Chain-of-Thought Verification

We now begin presenting our results and proofs in full detail. This section provides a resolution to the overarching questions (a) and (b) highlighted in the introduction.

A.1. Warm-up: Sound vs. Non-sound Chain-of-Thought Verification

We start our analysis with a warm-up that highlights a clear distinction in terms of optimal mistake bounds between sound and non-sound chain-of-thought verification.

Proposition A.1. *For any finite class of verifiers, H, there exists an online verification algorithm that makes at most $O(\log |H|)$ mistakes. Moreover, there exists a class of verifiers such that any online learning algorithm must make at least $\Omega(\log |H|)$ mistakes in expectation.*

Proof. We consider the majority vote verification algorithm, which given example $(x, \tau_{1:L})$ sequentially verifies each subtrace $(x, \tau_{1:\ell})$. It accepts the entire reasoning trace only if a majority of verifiers consistent with the history so far (initially the full set H) accept each sub-trace $(x, \tau_{1:\ell})$, and else rejects at the first step where the majority fails to accept. Each time there is a mistake, we discard all inconsistent verifiers. We claim that this algorithm suffers at most $\log |H|$ mistakes. Indeed, if we make a soundness mistake and accept an incorrect $(x_0, \tau_{1:\ell})$ (either accept (x_0, τ) or say the first mistake is after step ℓ), it must be that at least half of the surviving verifiers accept $(x_0, \tau_{1:\ell})$ and we can discard them. Similarly, if we wrongly mark $(x_0, \tau_{1:\ell})$ as incorrect, we again discard at least half the remaining consistent verifiers. The perfect verifier h^* never makes a mistake and is therefore never discarded. Thus, the number of consistent verifiers after M mistakes is at most $|H|/2^M$ and at least one. This implies $M \leq \log |H|$ always.

For the lower bound, consider the setting, where $X = \{x\}, \Sigma = \{0, 1\}$, and a correct proof is an unknown length-L binary sequence, b^*. Then we can define the class of verifiers, $H = \{h_b : b \in \{0, 1\}^L\}$, where

$$h_b(x, \tau_{1:\ell}) = \begin{cases} \text{YES} & \text{if } \tau_\ell = b_\ell \\ \text{NO} & \text{if } \tau_\ell \neq b_\ell \end{cases}$$

The adversary can then force $\Omega(L) = \Omega(\log |H|)$ mistakes in expectation as follows. Set $y^{(0)} = 1$. At timestep t present $\tau^{(t)} = (b^*[: y^{(t-1)}], \mathbf{1}_{L-t})$. Then set the location of the first incorrect step $y^{(t)} = y^{(t-1)} + 1$ or $y^{(t)} = y^{(t-1)} + 2$ with probability $1/2$ each. Then any verification algorithm will make a mistake with probability at least $1/2$ for at least $L/2$ timesteps. □

Proposition A.2. *For any finite class of verifiers, H, there is a sound online verification algorithm that makes at most $|H|$ mistakes. Moreover, there exists a class of verifiers, H, for which any sound verification algorithm cannot make fewer than $|H| - 1$ mistakes.*

Proof. A sound verification algorithm must accept (x, τ_i) only if *all* verifiers consistent with the history so far accept. If we incorrectly reject a correct trace (x, τ_i), then at least one of the consistent verifiers (other than h^*) rejects it and we discard all such verifiers. Thus, the size of consistent verifiers decreases by at least one on each mistake, so we can make no more than $|H| - 1$ mistakes.

For the lower bound, consider a setting with a single problem statement $X = \{x\}$ and Σ such that for any $t \in \mathbb{N}$ the learner is given an example $(x, \tau_1^{(t)}, \ldots, \tau_L^{(t)}) \in \{x\} \times \Sigma^L$. We let $\tau^* = (\tau_1^*, \ldots, \tau_L^*) \in \Sigma^L$ be the only incorrect reasoning trace for problem instance x and define the class of verifiers, $H = \{h_\sigma : \{x\} \times \Sigma^*, \mapsto \{\text{YES}, \text{NO}\}, \sigma \in \Sigma^L \mid h_\sigma(x, \tau) = \text{NO} \text{ iff } \tau = \sigma\}$, such that $h_{\tau^*} \in H$ is the only correct verifier. We now consider any sound verification algorithm and an adversary that chooses the realizable sequence of examples. For any reasoning trace $\tau_{1:L}^{(t)}$ the adversary chooses, there will always be one verifier $h^{(t)} \in H$ consistent with the examples seen so far that predicts NO on $\tau_{1:L}^{(t)}$. Therefore, the sound verifier will always predict L as the first faulty step for any such instance. An adversary will then reveal the true label, ∞, and $h^{(t)}$ will be the only verifier removed from the set of verifiers consistent with the history so far at this timestep. Therefore, the adversary can guarantee at least $|H| - 1$ mistakes against the sound verifier. □

Depending on H, it may be possible to obtain sharper mistake bounds for learning sound verifiers compared to Theorem A.2. For example, in the following (adapted from (Balcan et al., 2025)), we show that there is a sound online verification algorithm that achieves a $O(\log |H|)$ mistake bound. We provide additional examples in Appendix C.4.

Example A.3 (Sound verification of discrete puzzles)**.** We consider the problem of verifying a discrete puzzle given an online sequence of (possibly incorrect) solutions to the puzzle. These puzzles can be encoded as a directed graph where the nodes correspond to puzzle states and edges define reasoning steps that correspond to moves or transitions between the states. The precise rules of the puzzle are encoded by some h^* (to be learned) from a class H of verifiers defined over the graph. As a concrete example, consider the *river-crossing* puzzle. The rules that will govern h^* are as follows. A farmer must transport a fox, chicken, and corn across a river using a boat that only holds the farmer plus one item. The constraints are (1) the fox cannot be left alone with the chicken, and (2) the chicken cannot be left alone with the corn. We model the puzzle as a directed graph $G = (V, E)$. A state $s \in V$ is a vector $s \in \{0, 1\}^4$ giving the river bank of (farmer, chicken, fox, corn), where 0 is the left bank and 1 is the right bank. A directed edge $(s, s') \in E$ exists iff the move from s to s' satisfies all the constraints. The start and goal states are $s_0 = (0, 0, 0, 0)$ and $s_g = (1, 1, 1, 1)$. A solution is a valid path from s_0 to s_g. Now consider prompts $x = (V, E_0)$ where $E_0 \subseteq E$ is a set of edges revealed in the prompt, and let a reasoning trace be a sequence of vertices, $\tau \in V^*$. We define a class of verifiers $H_L = \{h_{\tilde{E}} : \tilde{E} \in E\}$ for a fixed maximum length of traces L. For any hidden edge set $\tilde{E} \subseteq E$, define a verifier $h_{\tilde{E}}$ that outputs YES on $(x = (V, E_0), \tau_{1:t})$ if (1) $t = 1$ and $\tau_1 = s_0$, (2) $2 \leq t \leq L - 1$ and $(\tau_{t-1}, \tau_t) \in E_0 \cup \tilde{E}$, or (3) $t = L$, $(\tau_{L-1}, \tau_L) \in E_0 \cup \tilde{E}$, and $\tau_L = s_g$. Otherwise it outputs NO. We will show that there is a sound online verification algorithm with a mistake bound of $|E^*| \leq |E| = \log |H| = \log(2^{|E|})$. The online verifier maintains a list $\overline{E}$ of edges, initially empty, and accepts proofs for which the paths consist of edges from the set $\overline{E} \cup E_0$, where E_0 is the set of edges in the problem statement. Each time the verifier makes a mistake, it must have rejected an edge e' in $E^* \setminus \overline{E}$, and it can simply update $\overline{E}$ to $\overline{E} \cup \{e'\}$.

A.2. Two-way Online-to-Online Reduction to Verifying Prefixes

In this section, we show that online chain-of-thought verification reduces to the problem of verifying *prefixes* of reasoning traces. At a high level, in this setting the learner observes a sequence of prefixes of varying length and must predict whether the last step of those prefixes is correct.

We explicitly construct a chain-of-thought verification algorithm from a prefix verification algorithm $\mathcal{A}_{\text{pfx}}$, as shown in Algorithm 1. Given a labeled example $\big((x, \tau_{1:\ell}), y\big)$ where $(x, \tau_{1:\ell}) \in X \times \Sigma^{\leq L}$ and $y \in \{\text{YES}, \text{NO}\}$, the call $\mathcal{A}_{\text{pfx}}.\textsc{Update}((x, \tau_{1:\ell}), y)$ updates the internal parameters of $\mathcal{A}_{\text{pfx}}$.

Theorem A.4. *Let H be a class of verifiers. Suppose there exists an online algorithm $\mathcal{A}_{\text{pfx}}$ for the prefix online verification problem that for class H achieves soundness and completeness mistake bounds M_s and M_c, respectively. Then there exists an online algorithm $\mathcal{A}$ for the chain-of-thought verification problem with the same soundness and completeness mistake bounds.*

Proof. We explicitly construct a chain-of-thought verification algorithm from a prefix verification algorithm $\mathcal{A}_{\text{pfx}}$, as shown in Algorithm 1. Given a labeled example $\big((x, \tau_{1:\ell}), y\big)$ where $(x, \tau_{1:\ell}) \in X \times \Sigma^{\leq L}$ and $y \in \{\text{YES}, \text{NO}\}$, the call $\mathcal{A}_{\text{pfx}}.\textsc{Update}((x, \tau_{1:\ell}), y)$ updates the internal parameters of $\mathcal{A}_{\text{pfx}}$.

We observe that algorithm $\mathcal{A}$ makes a completeness mistake at timestep t, only if $\mathcal{A}_{\text{pfx}}(x^{(t)}, \tau^{(t)}_{1:\hat{y}^{(t)}}) = \text{NO}$, when $\mathcal{O}(x^{(t)}, \tau^{(t)}_{1:\hat{y}^{(t)}}) = \text{YES}$. In that case, Step 1 guarantees that $\mathcal{A}_{\text{pfx}}$ also makes a completeness mistake at that timestep. Similarly, algorithm $\mathcal{A}$ makes a soundness mistake at timestep t, only if $\mathcal{A}_{\text{pfx}}(x^{(t)}, \tau^{(t)}_{1:y^{(t)}}) = \text{YES}$, when $\mathcal{O}(x^{(t)}, \tau^{(t)}_{1:y^{(t)}}) = \text{NO}$. In that case, Step 1 guarantees that $\mathcal{A}_{\text{pfx}}$ also makes a soundness mistake at that timestep.

Since the feedback passed into $\mathcal{A}_{\text{pfx}}$ is always consistent with the oracle $\mathcal{O}$, the sequence of examples used to update $\mathcal{A}_{\text{pfx}}$ remains realizable by H, which concludes the proof.

Algorithm 1 Constructing $\mathcal{A}$ using $\mathcal{A}_{\text{pfx}}$

Input: prefix chain-of-thought verification algorithm $\mathcal{A}_{\text{pfx}}$
for $t \in [T]$ **do**
  Observe example $z^{(t)} = \big(x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_L^{(t)})\big)$
  Predict $\hat{y}^{(t)} = \min\big(\{\ell \in [L] : \mathcal{A}_{\text{pfx}}(x^{(t)}, \tau_{1:\ell}^{(t)}) = \text{NO}\} \cup \{\infty\}\big)$
  Receive correct label $y^{(t)} \in [L] \cup \{\infty\}$
  if $\hat{y}^{(t)} < y^{(t)}$ **then**
    $\mathcal{A}_{\text{pfx}}.\textsc{Update}((x^{(t)}, \tau_{1:\hat{y}^{(t)}}^{(t)}), \text{YES})$
  else if $\hat{y}^{(t)} > y^{(t)}$ **then**
    $\mathcal{A}_{\text{pfx}}.\textsc{Update}((x^{(t)}, \tau_{1:y^{(t)}}^{(t)}), \text{NO})$
  end if
end for

□

We highlight that in the reduction used to prove Theorem A.4, when algorithm $\mathcal{A}$ makes a completeness mistake, the location of the first faulty step is not passed into algorithm $\mathcal{A}_{\text{pfx}}$ (Step 1). This would suggest that prefix chain-of-thought verification is more difficult than chain-of-thought verification. However, somewhat surprisingly, via the following result we show that that is not the case. Specifically, under a mild assumption on the hypothesis class (there exists a reasoning step that every verifier in the class rejects) we give a reduction in the opposite direction.

Theorem A.5. *Let $H \subseteq \{\text{YES}, \text{NO}\}^{X \times \Sigma^*}$ be a class of verifiers, where there exists $F \in \Sigma$ such that for every $h \in H$, every $x \in X$, and every correct prefix $\tau_{1:\ell-1}$, $h(x, (\tau_{1:\ell-1}, F)) = \text{NO}$. Suppose there exists an online algorithm $\mathcal{A}$ for the online verification model that for class H achieves soundness and completeness mistake bounds M_s and M_c, respectively. Then there exists an online algorithm $\mathcal{A}_{\text{pfx}}$ for the prefix chain-of-thought verification model with the same soundness and completeness mistake bounds.*

Proof. Given $\mathcal{A}$ we will construct algorithm, $\mathcal{A}_{\text{pfx}}$ as shown in Algorithm 2.

Now suppose that at some timestep t, $\mathcal{A}_{\text{pfx}}$ made a prefix soundness mistake, that is $\hat{y}_{\text{pfx}}^{(t)} = \text{YES}$ but $y_{\text{pfx}}^{(t)} = \text{NO}$. Then it must have been that $\mathcal{A}(z^{(t)}) > \ell$ and Step 2 guarantees that $\mathcal{A}$ will make a soundness mistake at this timestep too.

Suppose now that at some timestep t, $\mathcal{A}_{\text{pfx}}$ made a prefix completeness mistake, that is $\hat{y}_{\text{pfx}}^{(t)} = \text{NO}$ but $y_{\text{pfx}}^{(t)} = \text{YES}$. Then it must have been that $\mathcal{A}(z^{(t)}) \leq \ell$. Since we know that steps $\tau_1^{(t)}, \ldots, \tau_{\ell-1}^{(t)}$ are correct, and that token F is rejected by every verifier in the H, it must be that $\mathcal{O}(z^{(t)}) = \ell + 1$ and Steps 2 and 2 guarantee that $\mathcal{A}$ will also make a completeness mistake at this timestep.

Notice that in any case, the feedback provided to $\mathcal{A}$ is always consistent with the oracle $\mathcal{O}$, so the sequence of examples that $\mathcal{A}$ observes remains realizable by the class, H. □

Remark A.6. Theorem A.4 and Theorem A.5 establish the equivalence between chain-of-thought verification and prefix verification. Therefore, in what is to follow we will mostly focus on the latter, dropping the term "prefix" when it is clear from context.

A.3. Prefix Verification with a Soundness Mistake Budget

In this section, we study the trade-off between soundness and completeness mistakes in prefix verification when the learner is allowed to make a small number of soundness mistakes, $k \in \mathbb{N}$. Specifically, the learner aims to minimize the number of overall mistakes while respecting the soundness mistake budget, k.

Algorithm 2 Constructing $\mathcal{A}_{\text{pfx}}$ using $\mathcal{A}$

Input: chain-of-thought verification algorithm $\mathcal{A}$
for $t \in [T]$ **do**
  Observe example $z_{\text{pfx}}^{(t)} = \left(x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_\ell^{(t)})\right) \in X \times \Sigma^{\leq L}$
  Construct $z^{(t)} = \left(x^{(t)}, (\tau_1^{(t)}, \ldots, \tau_\ell^{(t)}, F, \ldots, F)\right) \in X \times \Sigma^L$
  Compute $\hat{y}^{(t)} \leftarrow \mathcal{A}(z^{(t)})$
  if $\hat{y}^{(t)} \leq \ell$ **then**
    Predict $\hat{y}_{\text{pfx}}^{(t)} = \text{NO}$
  else
    Predict $\hat{y}_{\text{pfx}}^{(t)} = \text{YES}$
  end if
  Receive correct label $y_{\text{pfx}}^{(t)} \in \{\text{YES}, \text{NO}\}$
  if $y_{\text{pfx}}^{(t)} = \text{NO}$ **then**
    $\mathcal{A}.\text{UPDATE}(z^{(t)}, \ell)$
  else
    if $\ell < L$ **then**
      $\mathcal{A}.\text{UPDATE}(z^{(t)}, \ell + 1)$
    else
      $\mathcal{A}.\text{UPDATE}(z^{(t)}, \infty)$
    end if
  end if
end for

We recall the definition of an SC-mistake tree and argue that it fully describes the strategy of an adversary against a deterministic prefix verification algorithm. Specifically, the adversary can ask the learner to verify internal nodes of the tree starting from the root. Say that for an internal node, z, the learner outputs NO. Then the adversary reveals the correct label, $(z, z_c) = \text{YES}$ (forcing the learner to make a completeness mistake) and follows the curvy downward edge. If the learner outputs YES, then the adversary reveals the true label $(z, z_s) = \text{NO}$ (forcing the learner to make a soundness mistake). Since the learning algorithm is deterministic, the interaction between learner and adversary forms a sequence of labeled examples in a path down the SC-mistake tree, where every curvy edge in the path corresponds to a completeness mistake and every straight edge corresponds to a soundness mistake that the learner made.

With the above interpretation of a shattered SC mistake tree, it follows that the SC-Ldim(H, k) lower bounds the number of overall mistakes of any deterministic online verification algorithm that guarantees at most k soundness mistakes.

Theorem A.7. *Consider a class of verifiers, H and any $k \in \mathbb{N}$. There exists a strategy of an adversary that defines a realizable sequence of reasoning traces, such that any deterministic algorithm that guarantees at most k soundness mistakes can be forced to make at least SC-Ldim(H, k) mistakes overall.*

Proof. By definition of the SC-Littlestone dimension, there must exist a (k, m)-difficult mistake tree for H. We consider a deterministic learner and a strategy of the adversary that presents the reasoning trace examples in the internal nodes of the tree, starting from the root. The adversary then reveals the true label of the instance: if the learner predicted the first faulty trace to be at some step $i = \text{NO}$ then the true label is set to $v_c = \text{YES}$, otherwise it is set to $v_s = \text{NO}$. The adversary then follows the downward edge that corresponds to the true label. Since the tree is full, the adversary can guarantee a mistake at every node. Since the learner is deterministic, her interaction with the adversary follows a root-to-leaf path, every curvy edge of which corresponds to a completeness mistake and every straight edge corresponds to a soundness mistake. The path must have at most k straight edges, since the learner guarantees at most k soundness mistakes. Since the tree is (k, m)-difficult, we also have that every such path has length at least m, which concludes the proof. □

Algorithm 3 Prefix verification with at most k soundness mistakes

Input: class of verifiers H; soundness mistake budget k
$H^{(1)} \leftarrow H$ — set initial version space
for $t \in [T]$ **do**
  Observe example $z^{(t)} = \left(x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_{\ell^{(t)}}^{(t)})\right) \in X \times \Sigma^{\leq L}$
  Compute $\mathcal{Y}^{(t)} = \{h(z^{(t)}) \mid h \in H^{(t)}\}$
  if $\mathcal{Y}^{(t)} = \{\text{YES}\}$ **then**
    Predict $\hat{y}^{(t)} \leftarrow \text{YES}$
  else if $k = 0$ ***or*** $\text{YES} \notin \mathcal{Y}^{(t)}$ **then**
    Predict $\hat{y}^{(t)} \leftarrow \text{NO}$ — must reject
  else
    Compute $m_c \leftarrow \text{SC-Ldim}\left(H^{(t)}[(z^{(t)}, \text{YES})], k\right)$
    Compute $m_s \leftarrow \text{SC-Ldim}\left(H^{(t)}[(z^{(t)}, \text{NO})], k-1\right)$
    if $m_c \leq m_s$ **then**
      Predict $\hat{y}^{(t)} \leftarrow \text{NO}$
    else
      Predict $\hat{y}^{(t)} \leftarrow \text{YES}$
    end if
  end if
  Receive correct label $y^{(t)} \in \{\text{YES}, \text{NO}\}$
  $H^{(t+1)} \leftarrow H^{(t)}[(z^{(t)}, y^{(t)})]$ — update version space
  if $\hat{y}^{(t)} = \text{YES}$ ***and*** $y^{(t)} = \text{NO}$ **then**
    $k \leftarrow k - 1$ — update soundness mistake budget
  end if
end for

We now present Algorithm 3, which we show that for any soundness mistake budget $k \in \mathbb{N}$ guarantees at most SC-Ldim(H, k) overall mistakes. For a class of verifiers, H, example $(x, \tau) \in X \times \Sigma^{\leq L}$, and label $y \in \{\text{YES}, \text{NO}\}$, we write $H[((x, \tau), y)] := \{h \in H \mid h(x, \tau) = y\}$.

Theorem A.8. *Consider a class of verifiers, H, and a soundness mistake budget $k \in \mathbb{N}$ such that SC-Ldim$(H, k) = m$. Then Algorithm 3 guarantees at most k soundness mistakes and at most m mistakes overall on any realizable sequence of examples.*

Proof. We proceed by induction on $m + k$.

Base case 1. For $m = 0$ and $k \geq 0$, there does not exist a (k, m)-difficult SC mistake tree, so every $h \in H$ must predict unanimously on every instance. Therefore, Lines 3 and 3 of Algorithm 3 guarantee that the algorithm predicts correctly on every instance.

Base case 2. For $k = 0$ and $m \geq 0$, we proceed with induction on m. We have already proved the base case for $k = m = 0$. We now assume that the statement holds for SC-Ldim$(H, 0) = m - 1$ and we will prove it for SC-Ldim$(H, 0) = m$. Since $k = 0$, Algorithm 3 will always reject every instance. Consider the first timestep t, where the algorithm makes a mistake. Then it must be that SC-Ldim$(H^{(t)}[(z^{(t)}, \text{YES})], 0) \leq m - 1$, as otherwise, we would be able to construct a $(0, m + 1)$ difficult tree. From timestep $t + 1$ and onwards the adversary can only present sequences that are realizable by $H^{(t)}[(z^{(t)}, \infty)]$ and by the inductive hypothesis the desired statement holds.

Inductive Hypothesis. We consider $k \geq 1$ and $m \geq 1$ and assume that the desired statement holds for all $k' \leq k, m' \leq m$, and $k' + m' \leq k + m - 1$. We consider a class of verifiers, where SC-Ldim$(H) = m$. Consider the first timestep, t where the algorithm makes a mistake. Then there must exist a verifier in $H^{(t)}$ that predicts YES and one that predicts NO.

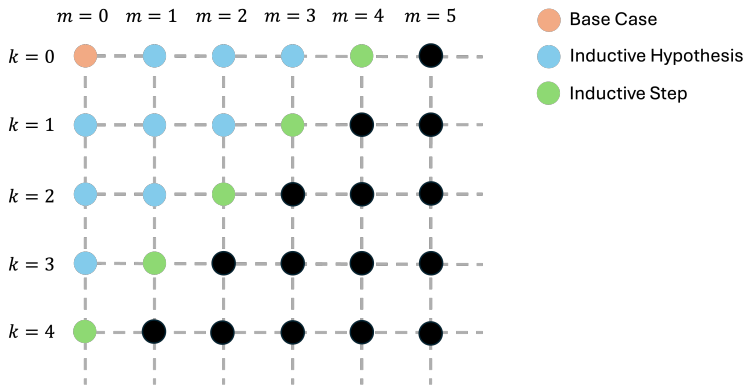

Figure 2. Visual representation of induction on $m + k$ used to prove Theorem A.8.

Inductive Step. We first consider the case, where the predicted label is $\hat{y}^{(t)} = \text{YES}$. It suffices to argue that $\text{SC-Ldim}(H^{(t)}[(z^{(t)}, \text{NO})], k-1) \leq m-1$. For the sake of contradiction, assume the opposite. Then by Line 3 of Algorithm 3, we know that $m \leq \text{SC-Ldim}(H^{(t)}[(z^{(t)}, \text{NO})], k-1) \leq \text{SC-Ldim}(H^{(t)}[(z^{(t)}, \text{YES})], k)$. This implies that there exists a $(k, m+1)$-difficult tree, which is a contradiction. When the predicted label is $\hat{y}^{(t)} = \text{NO}$, it suffices to prove that $\text{SC-Ldim}(H^{(t)}[(z^{(t)}, \text{YES})], k) \leq m-1$, which follows similarly to the argument above. From timestep $t+1$ and onwards the adversary is constrained to only presenting sequences of examples that are realizable by $H^{(t)}[(z^{(t)}, y^{(t)})]$ and since $\text{SC-Ldim}(H^{(t)}[(z^{(t)}, y^{(t)})] \leq m-1$ for any $y^{(t)}$, the desired statement follows from the inductive hypothesis. □

A.4. Prefix Verification with Linear Cost Objective

We show that any deterministic verification algorithm using H must suffer cumulative loss at least $\text{WSC-Ldim}(H)$.

Theorem A.9. *For any hypothesis class, H, and any deterministic learning algorithm, there exists a realizable sequence such that the learner must suffer cumulative loss at least W for every $W < \text{WSC-Ldim}(H)$.*

Proof. Fix $W < \text{WSC-Ldim}(H)$. By definition, there exists a WSC mistake tree $\mathcal{T}$ shattered by H such that every root-to-leaf path in $\mathcal{T}$ has cumulative weight at least W. We consider a deterministic learning algorithm, V_H and we define an adaptive adversary that interacts with V_H as follows. Starting at the root of $\mathcal{T}$, at each internal node the adversary presents the label of the node (x, τ) to the learner and observes the learner's prediction $\hat{y} \in \mathcal{Y}$. At every node, the adversary can choose an outgoing edge inconsistent with the learner's prediction and induce loss equal to the edge's weight. To see this, we consider the following cases.

First, we consider the case where the learner's prediction is $\hat{y} = \text{YES}$. From the definition of a WSC mistake tree, we know that there exists a straight edge with weight γ_s and label NO that the adversary can choose to force the learner to make a soundness mistake and incur cost γ_s. We now consider the case, where the learner's prediction is $\hat{y} = \text{NO}$. Again, from the definition of a WSC-mistake tree, there must exist a curvy edge with weight γ_c and label YES that the adversary can choose to force the learner to make a completeness mistake and incur cost γ_c. The adversary proceeds by presenting the example to which the chosen outgoing edge leads in the tree $\mathcal{T}$ and repeats the process of finding the corresponding true label.

The learner is deterministic, so this sequence of examples will correspond to a root-to-leaf path in $\mathcal{T}$ and each edge will correspond to a mistake, where the loss of the learner is at least the weight of the edge. By definition of $\mathcal{T}$, however, every root-to-leaf path has cumulative weight at least W. Lastly, since $\mathcal{T}$ is shattered by H, there must exist an $h \in H$ that is consistent with the sequence of examples presented to the learner, which ensures realizability. □

We next present and analyze the performance of Algorithm 4. At a high-level, given a problem statement and a reasoning trace, the algorithm predicts the first faulty step that minimizes the worst-case sum of the immediate

Algorithm 4 Prefix verification with linear cost objective

Input: class of verifiers H, mistake costs $\gamma_s, \gamma_c \in \mathbb{R}_{\geq 0}$
$H^{(1)} \leftarrow H$ set initial version space
for $t \in [T]$ **do**
  Observe example $z^{(t)} = \left(x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_{\ell^{(t)}}^{(t)})\right)$
  Compute $\mathcal{Y}^{(t)} = \{h(z^{(t)}) \mid h \in H^{(t)}\}$ set of possible outputs
  if $|\mathcal{Y}^{(t)}| = 1$ **then**
    Predict $\hat{y}^{(t)} \in \mathcal{Y}^{(t)}$
  else
    Compute $m_c = \gamma_c + \text{WSC-Ldim}\left(H^{(t)}[(z^{(t)}, \text{YES})]\right)$
    Compute $m_s = \gamma_s + \text{WSC-Ldim}\left(H^{(t)}[(z^{(t)}, \text{NO})]\right)$
    if $m_c \leq m_s$ **then**
      Predict $\hat{y}^{(t)} \leftarrow \text{NO}$
    else
      Predict $\hat{y}^{(t)} \leftarrow \text{YES}$
    end if
  end if
  Receive correct label $y^{(t)} \in \mathcal{Y}$
  Set $H^{(t+1)} \leftarrow H^{(t)}[(z^{(t)}, y^{(t)})]$ Update version space
end for

loss and the complexity of the future version space, as captured by the WSC-Littlestone dimension. We denote the immediate loss for a learner's prediction $\hat{y}$ and a true prediction y as $\ell(\hat{y}, y) \in \{0, \gamma_s, \gamma_c\}$.

Theorem A.10. *For any class of verifiers, H, Algorithm 4 achieves cumulative loss at most WSC-Ldim(H) on any realizable sequence of examples.*

Proof. We will show that for every timestep t, $\ell(\hat{y}^{(t)}, y^{(t)}) \leq \text{WSC-Ldim}(H^{(t)}) - \text{WSC-Ldim}(H^{(t+1)})$. We consider the prediction $\hat{y}^{(t)}$ of Algorithm 4 in timestep t and get

$$\begin{aligned}\ell(\hat{y}^{(t)}, y^{(t)}) + \text{WSC-Ldim}(H^{(t+1)}) &\leq \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{y}^{(t)}, j) + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] \\ &= \min_{i \in \mathcal{Y}^{(t)}} \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(i, j) + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, j)])\right]\end{aligned}$$

We know that $\ell(i, i) = 0$ and WSC-Ldim ≥ 0, so $\max_j$ always picks $j \neq i$. Therefore, we get that

$$\begin{aligned}&\min_{i \in \mathcal{Y}^{(t)}} \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(i, j) + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] \\ &\quad = \min\left\{\ell(\text{YES}, \text{NO}) + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, \text{NO})]), \ell(\text{NO}, \text{YES}) + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, \text{YES})])\right\} \\ &\quad = \min\left\{\gamma_s + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, \text{NO})]), \gamma_c + \text{WSC-Ldim}(H^{(t)}[(z^{(t)}, \text{YES})])\right\} \\ &\quad \leq \text{WSC-Ldim}(H^{(t)}),\end{aligned}$$

where the last inequality follows from the definition of the WSC-Littlestone dimension.

□

We note that when $\gamma_s = \gamma_c = 1$, the WSC-Littlestone dimension recovers the definition of the Littlestone dimension. Moreover, when $\gamma_c = 1$ and $\gamma_s = d$, any finite class H has WSC-Ldim$(H) \leq O\left(\frac{d}{\log d} \log(|H|)\right)$. This recovers a previous result by Helmbold et al. (2000) on online binary classification with linear costs and finite hypothesis classes. We provide a formal argument in Proposition C.4.

Algorithm 5 Constructing $\mathsf{Wrap}(V_H, \mathcal{P})$

Input: set of k provers $\mathcal{P}$; values $\alpha, \epsilon, \epsilon', \delta$; step-by-step verification algorithm V_H with soundness mistake-bound M_s and completeness mistake-bound M_c

Draw set S_1 of $8\left(\frac{M_c+M_s}{\epsilon} + \log\frac{2}{\delta}\right)$ random examples from D

for *each* $x \in S_1$ **do**

  Run PROCESS-EXAMPLE$(x, \mathcal{P}, \alpha, \epsilon', V_H)$ Train V_H on x and $\mathcal{P}$

end for

Let $h_1, \ldots, h_M$ be the hypothesis verifiers produced in Step 5

Draw set S_2 of $O\left(\frac{1}{\epsilon} \cdot \frac{M_s+M_c}{\min(M_s,M_c)+1} \cdot \log\left(\frac{M_s+M_c}{\delta}\right)\right)$ examples from D

for *each* $x \in S_2$ **do**

  for *each* $h_i \in \{h_1, \ldots, h_M\}$ **do**

    Run TEST-HYPOTHESIS$(x, \mathcal{P}, \alpha, \epsilon', h_i)$

  end for

end for

Choose any h_i with

  empirical soundness error $\leq \frac{3\epsilon}{4} \cdot \frac{M_s}{M_s+M_c}$ and

  empirical completeness error $\leq \frac{3\epsilon}{4} \cdot \frac{M_c}{M_s+M_c}$

If no such h_i exists, halt with failure.

Use $h = h_i$ in Algorithm 8, creating $\mathsf{Wrap}(V_H, \mathcal{P})$

B. Improving Weak Provers with Online Verification

In this section, we state the formal theorem and proof, showing that an online prefix verification algorithm can be used to exponentially amplify the performance of a set of weak provers.

Theorem B.1. *Given an (α, γ)-good set of k provers $\mathcal{P}$, a verifier-learning algorithm V_H with stepwise completeness and soundness mistake-bounds M_c and M_s respectively, and given the value α, we can learn a prover $\mathsf{Wrap}(V_H, \mathcal{P})$ using a training sample of $O\left(\frac{M_s+M_c}{\epsilon} + \frac{1}{\epsilon}\left(\frac{M_s+M_c}{\min(M_s,M_c)+1}\right)\log\left(\frac{M_s+M_c}{\delta}\right)\right)$ problem instances $x \sim D$ and making $O\left(\left(\frac{M_s+M_c}{\epsilon}\right)\left(\frac{M_s+M_c}{\min(M_s,M_c)+1}\right)\left(\log\frac{M_s+M_c}{\delta}\right)\left(\frac{Lk}{\alpha}\log\frac{Lk}{\epsilon'}\right)\right)$ calls to a labeling oracle $\mathcal{O} = h^*$ such that with probability at least $1-\delta$:*

1. *Given a problem instance x, $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs either "I don't know" or a proof $\boldsymbol{\tau}$.*
2. *The probability over $x \sim D$ that $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs "I don't know" is at most $(1-\gamma) + \epsilon_c + \epsilon_s + \epsilon'$.*
3. *The probability over $x \sim D$ that $\mathsf{Wrap}(V_H, \mathcal{P})$ outputs an incorrect proof is at most ϵ_s,*

where $\epsilon_c = \frac{\epsilon M_c}{M_c+M_s}$ and $\epsilon_s = \frac{\epsilon M_s}{M_c+M_s}$.

Proof. We begin by drawing a set S_1 of $8\left(\frac{M_c+M_s}{\epsilon} + \log\frac{2}{\delta}\right)$ random examples from D. We then run Process-Example (Algorithm 6) on each example $x \in S_1$. Each run of Process-Example makes $O(\frac{Lk}{\alpha}\log\frac{Lk}{\epsilon'})$ calls to the labeling oracle $\mathcal{O}$ and results in either V_H making a mistake and updating its current hypothesis h or not. Let $err^s(h)$ denote the probability that h makes a soundness mistake when running Process-Example(x) on a random $x \sim D$ (probability taken over the draw from D and the randomness in the provers in $\mathcal{P}$). Similarly, let $err^c(h)$ denote the probability that h makes a completeness mistake when running Process-Example(x) on a random $x \sim D$. Notice that if each h produced had $err^s(h) \geq \frac{\epsilon}{2}\frac{M_s}{M_s+M_c}$ or $err^c(h) \geq \frac{\epsilon}{2}\frac{M_c}{M_s+M_c}$, then the expected number of soundness mistakes would be greater than $2M_s$ or the expected number of completeness mistakes would be greater than $2M_c$. By the Martingale Chernoff argument of Littlestone (1989), we can therefore conclude that with probability $\geq 1-\delta/2$, at least one of the h produced has $err^s(h) \leq \frac{\epsilon}{2}\frac{M_s}{M_s+M_c}$ and $err^c(h) \leq \frac{\epsilon}{2}\frac{M_c}{M_s+M_c}$.

We now draw a fresh set S_2 of $O\left(\frac{1}{\epsilon}\left(\frac{M_s+M_c}{\min(M_s,M_c)+1}\right)\log\left(\frac{M_s+M_c}{\delta}\right)\right)$ examples to test the at most $M_s + M_c$

Algorithm 6 PROCESS-EXAMPLE

Input: example x; set of k provers $\mathcal{P}$; values α, ϵ'; step-by-step verification algorithm V_H
for $\ell = 1$ **to** L **do**
  Set $i \leftarrow 1$
  while $i \leq \left\lceil \frac{1}{\alpha} \ln \frac{kL}{\epsilon'} \right\rceil$ **do**
    Sample $\tau_\ell^{(i,j)} \sim P_j(x, \tau_1, \ldots, \tau_{\ell-1})$ for all $P_j \in \mathcal{P}$ — ask each prover for next step
    Set $v_\ell^{(i,j)} = V_H(x, \tau_1, \ldots, \tau_{\ell-1}, \tau_\ell^{(i,j)}) \in \{\text{YES}, \text{NO}\}$ — run V_H
    Set $y_\ell^{(i,j)} = \mathcal{O}(x, \tau_1, \ldots, \tau_{\ell-1}, \tau_\ell^{(i,j)}) \in \{\text{YES}, \text{NO}\}$ — query labeling oracle
    if $v_\ell^{(i,j)} = \text{NO}$ *for all* $j \in [k]$ **then**
      Set $i = i + 1$ — discard and try again
    else
      Set $\tau_\ell \leftarrow \tau_\ell^{(i,j)}$ for any $j \in [k]$ with $v_\ell^{(i,j)} = \text{YES}$ — append step
      Break and continue the for-loop — exit while; continue with for-loop
    end if
  end while
  if $v_{\ell'}^{(i,j)} \neq y_{\ell'}^{(i,j)}$ *for any* $\ell' \leq \ell$, *any* i, *any* $j \in [k]$ **then**
    V_H.Update$((x, \tau_{1:\ell'-1}, \tau_{\ell'}^{(i,j)}), y_{\ell'}^{(i,j)})$ — make progress on learning verifier
    Return — timeout (no proof found)
  end if
end for
if $\exists\, \ell \leq L$ *such that* $\mathcal{O}(x, \tau_{1:\ell}) = \text{NO}$ **then**
  V_H.Update$((x, \tau_{1:\ell}), \text{NO})$ — learning progress
end if

hypotheses $h_1, ..., h_M$ produced above, by running each of them through Test-Hypothesis (Algorithm 7), which simulates the use of the hypothesis at test-time (Algorithm 8), checking to see if any soundness or completeness mistake is made. Note that if $M_s = 0$ then *all* h_i will have $err^s(h_i) = 0$ and similarly if $M_c = 0$ then *all* h_i will have $err^c(h_i) = 0$, so we may wlog focus on the case that $M_s \geq 1$ and $M_c \geq 1$. Therefore, our sample S_2 is large enough that by Chernoff bounds, with probability $\geq 1 - \delta/2$, (a) at least one of the hypotheses h_i tested will have empirical soundness error at most $\frac{3\epsilon}{4} \frac{M_s}{M_s+M_c}$ and empirical completeness error at most $\frac{3\epsilon}{4} \frac{M_c}{M_s+M_c}$, and (b) all such hypotheses h_i will have $err^s(h_i) \leq \epsilon \frac{M_s}{M_s+M_c}$ and $err^c(h_i) \leq \epsilon \frac{M_c}{M_s+M_c}$.

Finally, we choose one such h_i and use it in Algorithm 8, creating the prover $\mathsf{Wrap}(V_H, \mathcal{P})$. The probability that the prover returns "I don't know" in Step 12 is at most the probability $1 - \gamma$ that $\mathcal{P}$ is not α-good for x, plus the probability ϵ' that the algorithm reaches its timeout even if no mistake is made (*i.e.*, the prover fails to generate a proof where all L steps are correct), plus the probability ϵ_c of a completeness mistake plus the probability ϵ_s of a soundness mistake (that could lead to a state where there is no correct continuation of the proof). The probability the prover returns an incorrect proof in Step 13 is the probability ϵ_s of a soundness mistake, as desired. The entire process is described formally in Algorithm 5. □

Algorithm 7 TEST-HYPOTHESIS

Input: example x; set of k provers $\mathcal{P}$; values α, ϵ'; verifier $h : X \times \Sigma^* \mapsto \{\text{YES}, \text{NO}\}$
Run Algorithm 8$(x, \mathcal{P}, \alpha, \epsilon', h)$, recording all predictions made by h
if *Algorithm 8 returned a proof* $\boldsymbol{\tau}$ **then**
  if $\mathcal{O}(x, \tau_{1:\ell}) = \text{NO}$ *for any* $\ell \leq L$ **then**
    Return `"Soundness Mistake"`
  end if
else
  if *h made any mistakes* **then**
    Return the type of the first mistake (`"Completeness Mistake"` or `"Soundness Mistake"`)
  end if
end if
Return `"Correct"`

Algorithm 8 Combining set of provers $\mathcal{P}$ and learned verifier h to create $\mathsf{Wrap}(V_H, \mathcal{P})$

Input: prompt $x \in X$; set of k provers $\mathcal{P}$; values α, ϵ'; verifier $h : X \times \Sigma^* \mapsto \{\text{YES}, \text{NO}\}$
for $\ell = 1$ **to** L **do**
  Set $i \leftarrow 1$
  while $i \leq \left\lceil \frac{1}{\alpha} \ln \frac{kL}{\epsilon'} \right\rceil$ **do**
    Sample $\tau_\ell^{(i,j)} \sim P_j(x, \tau_1, \ldots, \tau_{\ell-1})$ for all $P_j \in \mathcal{P}$ — ask each prover for next step
    Set $v_\ell^{(i,j)} = h(x, \tau_1, \ldots, \tau_{\ell-1}, \tau_\ell^{(i,j)}) \in \{\text{YES}, \text{NO}\}$ — run the verifier
    if $v_\ell^{(i,j)} = \text{NO}$ *for all* $j \in [k]$ **then**
      $i = i + 1$
      Continue — discard and try again
    else
      Set $\tau_\ell \leftarrow \tau_\ell^{(i,j)}$ for any $j \in [k]$ with $v_\ell^{(i,j)} = \text{YES}$ — append step
      Break — exit while; continue for-loop
    end if
  end while
  If end-while reached, return `"I don't know"` — no correct next step found; timeout
end for
Return $\tau_1, \ldots, \tau_L$ — return proof found

C. Omitted Details and Proofs

C.1. Additional Related Work

We provide below more detailed background and context of relevant literature, organized by its relationship to our work.

Expressiveness of chain-of-thought reasoning. Chain-of-thought prompting (Wei et al., 2022), where models produce step-by-step reasoning traces, has become a dominant paradigm for complex reasoning tasks (Nye et al., 2021; Kojima et al., 2022). A growing theoretical literature explains why intermediate reasoning steps expand model capabilities. Li et al. (2024) show that constant-depth transformers with chain-of-thought can solve inherently serial problems (such as composing permutations) that are impossible without intermediate steps, establishing a provable separation between standard and CoT-augmented transformers. Merrill & Sabharwal (2024) prove that transformers with polynomial-length chain-of-thought can express any problem solvable in polynomial time, and Feng et al. (2023) provide a theoretical perspective on why CoT helps by analyzing the circuit complexity of CoT-augmented computation. From a learning-theoretic perspective, Malach (2024) show that auto-regressive next-token predictors with intermediate tokens are universal learners. The CoT structure—exposing intermediate reasoning steps—is precisely what makes step-level verification both possible

and meaningful, as it provides the sequential structure that our framework exploits.

Learning chain-of-thought reasoning and verification. On the learnability side, Joshi et al. (2025) formalize autoregressive learning with chain-of-thought, analyzing sample and computational complexity when reasoning chains are observed or latent, and Altabaa et al. (2025) show that learning guarantees can be independent of chain length under suitable assumptions. More broadly, a growing line of work studies the sample efficiency and training dynamics of learning with chain-of-thought: Wen et al. (2025) show that CoT reduces sample complexity from exponential to polynomial on parity-type tasks by inducing sparse attention patterns; Li et al. (2025) provide generalization analysis for training nonlinear transformers with CoT; Wang et al. (2025) prove statistical query lower bounds for learning compositional functions without curriculum, but show polynomial sample complexity is achievable with curriculum learning; and Kim & Suzuki (2025) prove that with CoT supervision, parity can be learned in one gradient step, whereas without CoT supervision exponential iterations are needed. Abbe et al. (2024) introduce the "globality barrier" to characterize when transformers can learn from scratchpads, showing that inductive scratchpads can break this barrier and enable out-of-distribution generalization. For verification, Balcan et al. (2025) give learning-theoretic foundations for chain-of-thought verification from a statistical perspective, and Amit et al. (2024) study self-proving models where the prover generates outputs along with proofs of correctness.

In more detail, Balcan et al. (2025) show that if the prompt-reasoning pairs are drawn from a fixed but unknown distribution (say chain-of-thoughts from a given trained LLM on random problems of interest), then it is possible to learn a verifier that correctly predicts the location of the first error in the reasoning (if one exists) with high probability, with a small sample complexity (linear in the VC dimension, $\log |H|$ for finite verifier function classes H). However, given feedback on mistakes in its reasoning, the LLM may produce a different reasoning which is out-of-distribution and we are no longer able to reliably verify. On the other hand, if we require the verifier to never accept an incorrect reasoning for prompts coming from a fixed distribution, they show a strong lower bound on the sample complexity, linear in the size of the verifier class H. Our work aims to develop a model for verification that overcomes this pessimistic lower bound by taking advantage of some weak assumptions on the correctness of LLM reasoning steps, while still allowing the LLM proofs to be adaptive.

Process reward models. A substantial body of empirical work has developed *process reward models* (PRMs) that provide step-level supervision for LLM reasoning. This literature pursues two distinct approaches. The first trains PRMs on *human annotations* of step-level logical correctness: Uesato et al. (2022) first compared process-based and outcome-based feedback, and Lightman et al. (2024) scaled this by collecting 800K human step-level labels, demonstrating that PRMs significantly outperform outcome reward models. These human-annotated PRMs effectively approximate ground-truth verifiers. The second approach avoids human annotation by training PRMs via *Monte Carlo estimation* (Wang et al., 2024; Luo et al., 2024), where a step's label is the fraction of sampled completions that reach the correct answer, or via *implicit reward derivation* from outcome models (Cui et al., 2025). These produce prover-specific value functions rather than ground-truth verifiers: a logically flawed step may score highly if the prover can recover from it. PRMs are widely used for best-of-N reranking and tree search (Lightman et al., 2024; Snell et al., 2025), and as dense reward signals for reinforcement learning (Setlur et al., 2025; Cui et al., 2025). Neither line of work provides formal learnability guarantees or analyzes the soundness-completeness tradeoff.

Online learning. The theoretical foundations of online learning are closely tied to combinatorial complexity measures, most notably the Littlestone dimension (Littlestone, 1988), which characterizes learnability and optimal mistake bounds in adversarial online classification. In his seminal COLT 1989 paper, Littlestone (1989) established an online-to-batch conversion, showing that online learnability implies PAC learnability with sample complexity controlled by the online mistake bound. A key challenge in using Littlestone's argument in our setting of boosting weak provers is that the reasoning steps depend on the problem and context and are not i.i.d. Recent work has significantly refined this picture: randomized variants of the Littlestone dimension characterize expected mistake bounds for randomized learners (Filmus et al., 2023), while computability-aware, strategy-aware and effective versions clarify when finite-mistake learning is achievable by constrained learners (Hasrati & Ben-David, 2023; Ahmadi et al., 2024; Delle Rose et al., 2025). Another extension related to abstention-error trade-off inspires our formulations for different types of verifier mistakes (Sayedi et al., 2010; Zhang & Chaudhuri, 2016). Parallel to these developments, the online learning framework has been extended to sequential prediction problems with temporal structure (Sridharan & Rakhlin, 2011), where the adversary's

behavior is partially forecastable, e.g., given by a noisy autoregressive function.

C.2. Background: Littlestone Dimension

We give an overview of the concepts and techniques from online classification (Littlestone, 1988; Daniely et al., 2011) that are relevant to our work. We begin with the notion of *shattered mistake trees*.

Mistake tree. A mistake tree is a rooted binary tree, where each internal node corresponds to examples in some domain set X and each edge corresponds to a label in a a set Y (for example in binary classification $Y = \{-1, 1\}$ and in multiclass classification $Y = [L]$), such that the outgoing edges from a single node have different labels. We say that a mistake tree is *shattered* by a hypothesis class H if for any root-to-leaf path that traverses internal nodes $x_1, x_2, \ldots, x_d$ there exists a hypothesis $h \in H$ such that the label of edge (x_i, x_{i+1}) is $h(x_i)$ for all $i \leq d$.

Intuitively, a mistake tree describes the strategy of an adversary against a deterministic learning algorithm. Internal nodes correspond to the examples that the adversary will present to the learner starting from the root. The pair of outgoing edges corresponds to the adversary's ability to induce a mistake no matter what the learner predicts. For example, if $Y = \{-1, 1\}$ and the learner predicts -1, the adversary can reveal label $+1$ and follow the corresponding downward edge to the next example. Therefore, the interaction between learner and adversary corresponds to a root-to-leaf path and shattering ensures that the sequence of examples presented to the learner is realizable by the hypothesis class H.

The Littlestone dimension captures the intuition that the more complex a hypothesis class is, the deeper the mistake trees it should be able to shatter.

Definition C.1 (Littlestone dimension). The Littlestone dimension of a hypothesis class H, denoted as $\mathrm{Ldim}(H)$, is the maximum integer d such that there exists a complete mistake tree of depth d that is shattered by H.

With the preceding understanding of a shattered mistake tree, it immediately follows that the Littlestone dimension lower bounds the number of mistakes of any deterministic learning algorithm. Littlestone (1988) showed that there exists an online learning algorithm, the Standard Optimal Algorithm (SOA), that achieves this lower bound. These two results completely characterize the optimal mistake bounds in online classification. Later, Daniely et al. (2011) extended these results to online multiclass classification.

C.3. Summary of Algorithms for Chain-of-Thought Verification

C.4. Additional Examples

Example C.2. (Structured Verifiers) Consider a setting where generating a solution to a problem naturally decomposes into intermediate steps, where each step needs to be verified against different criteria. For example, automated hiring involves multi-stage application process, where each component of an application requires different criteria of evaluation (CV requires a keyword similarity checking model vs. a coding interview must be checked for correctness). Similarly, in automated itinerary planning, different components of the plan require different types of checks (lodging requires checking availability of nearby hotels vs. activity selection requires aligning with user preferences). Concretely, we consider verifiers that can be decomposed into a product of step-specific mini-verifiers, $H = H_1 \times H_2 \times \cdots \times H_L$. where each mini-verifier $h_i \in H_i$ verifies the ith step of a reasoning trace i.e., $h_i : X \times \Sigma^i \to \{\text{YES}, \text{NO}\}$. Then every verifier in $h \in H$, where $h : X \times \Sigma^L \to \mathcal{Y}$ can be described as a list of L mini-verifiers $(h_1, \ldots, h_L)$

$$h(x, \tau = (\tau_1, \ldots, \tau_L)) = \min \{i \in [L] \text{ such that } h_i(x, (\tau_1, \ldots, \tau_i) = \text{NO}\}.$$

We argue that when $|H_i| \leq k$, there exists a sound verifier that makes at most $(k-1)L$ completeness (false-negative) mistakes. Consider the algorithm that outputs the first step $i \in [L]$ for which at least one function in H_i that is consistent with the history so far outputs NO. Then for every completeness mistake the learner can discard at least 1 mini-verifier. Therefore the learner can make at most $(k-1)L$ completeness mistakes.

Below we present an example of an infinite class of verifiers that can be learned with a finite mistake bound. This is an adaptation of a similar example presented in prior work (Joshi et al., 2025; Balcan et al., 2025).

Algorithm 9 Chain-of-Thought verification with soundness budget k

Input: class of hypotheses H; soundness mistake budget k
$H^{(1)} \leftarrow H$ initialize version space
for $t \in [T]$ **do**
  Observe example $z^{(t)} = (x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_L^{(t)})) \in X \times \Sigma^L$
  if $k = 0$ **then**
    Predict $\hat{y}^{(t)} \leftarrow \min\{\ell \in [L]$ such that $\exists h \in H^{(t)} : h(x^{(t)}, \tau_1, \ldots, \tau_\ell) = \text{NO}\}$
  else
    for $\ell \in [L]$ **do**
      Compute $m_c \leftarrow \text{SC-Ldim}\Big(\{\, h \in H^{(t)} : h(x, \tau_1, \ldots, \tau_\ell) = \text{YES} \,\}, k\Big)$
      Compute $m_s \leftarrow \text{SC-Ldim}\Big(\{\, h \in H^{(t)} : h(x, \tau_1, \ldots, \tau_\ell) = \text{NO} \,\}, k-1\Big)$
      if $m_c \leq m_s$ **then**
        Predict $\hat{y}^{(t)} \leftarrow \ell$
        break
      end if
      if $\ell = L$ **then**
        Predict $\hat{y}^{(t)} \leftarrow \infty$
      end if
    end for
  end if
  Receive correct label $y^{(t)} \in [L] \cup \{\infty\}$
  if $y^{(t)} < \hat{y}^{(t)}$ **then**
    Update $H^{(t+1)} \leftarrow \{\, h \in H^{(t)} : h(x^{(t)}, \tau_1^{(t)}, \ldots, \tau_{y^{(t)}}^{(t)}) = \text{NO} \,\}$
    Set $k \leftarrow k - 1$ update soundness mistake budget
  else if $y^{(t)} > \hat{y}^{(t)}$ **then**
    Update $H^{(t+1)} \leftarrow \{\, h \in H^{(t)} : h(x^{(t)}, \tau_1, \ldots, \tau_{\hat{y}^{(t)}}) = \text{YES} \,\}$
  end if
end for

Example C.3 (Linear Threshold Verifiers with a Margin). Let $X, \Sigma = \{x \in \mathbb{R}^d : \|x\|_2 \leq r\}$ and consider the class of d-dimensional linear threshold verifiers:

$$H = \{h_{w_0,w}(x, \tau) \mapsto \mathbb{1}\{w_0 x + w[-l:]\tau[-l:] \geq 0\} \mid w_0 \in \mathbb{R}, w \in \mathbb{R}^d, l = \min\{d-1, |\tau|\}\}$$

Now for a margin $\gamma > 0$, we define a sequence $(x^{(1)}, \tau_{1:L}^{(1)}, y^{(t)}), \ldots, (x^{(T)}, \tau_1 : L^{(T)}, y^{(T)}) \in X \times \Sigma^L \times [L] \cup \{\infty\}$ to be "margin-γ realizable by H" if there exists a single $h_{w_0^*, w^*} \in H$ such that

$$y^{(t)} = \min\{l : w_0^* x^{(t)} + w^*[-l:]\tau^{(t)}[-l:] < 0\} \text{ and } |\langle (w_0^*, w^*), (x^{(t)}, \tau^{(t)})\rangle| \geq \gamma.$$

By classical results on learning linear thresholds with a margin (Rosenblatt, 1958), it follows that if we restrict mistake trees to only contain nodes in $\{(x, \tau) : |(w_0^*, w^*), (x, \tau)| \geq \gamma\}$ then $\text{Ldim}(H) \leq O(R^2/\gamma^2)$.

C.5. Omitted Proofs

Proposition C.4. *For $\gamma_c = 1$, $\gamma_s = d$, and any finite class H, we have that WSC-Ldim$(H) \leq O\left(\frac{d}{\log d} \log(|H|)\right)$.*

Proof. For a fixed size of a hypothesis class, $|H| = N$, we will show an upper bound on the minimum weighted path among all WSC-mistake trees with at most N leaves. For simplicity, we only concern ourselves with the structure of the binary tree, as we can populate it with internal node instances and show that for each path there exist a consistent hypothesis ad hoc.

Algorithm 10 Chain-of-Thought verification with linear cost objective

Input: class of hypotheses H; mistake costs $\gamma_s, \gamma_c \in \mathbb{R}_{\geq 0}$
$H^{(1)} \leftarrow H$ initialize version space
for $t \in [T]$ **do**
  Observe example $z^{(t)} = (x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_L^{(t)})) \in X \times \Sigma^L$
  for $\ell \in [L]$ **do**
    Compute $\mathcal{Y}^{(t)} = \{h(x^{(t)}, \tau_{1:\ell}^{(t)}) \mid h \in H^{(t)}\}$ set of possible outputs
    if $\mathcal{Y}^{(t)} = \{\text{NO}\}$ **then**
      Predict $\hat{y}^{(t)} \leftarrow \ell$
      break
    else
      Compute $m_c = \gamma_c + \text{WSC-Ldim}\big(H^{(t)}[((x^{(t)}, \tau_{1:\ell}^{(t)}), \text{YES})]\big)$
      Compute $m_s = \gamma_s + \text{WSC-Ldim}\big(H^{(t)}[((x^{(t)}, \tau_{1:\ell}^{(t)}), \text{NO})]\big)$
      if $m_c \leq m_s$ **then**
        Predict $\hat{y}^{(t)} \leftarrow \ell$
        break
      end if
      if $\ell = L$ **then**
        Predict $\hat{y}^{(t)} \leftarrow \infty$
      end if
    end if
  end for
  Receive correct label $y^{(t)} \in [L] \cup \{\infty\}$
  if $y^{(t)} < \hat{y}^{(t)}$ **then**
    Update $H^{(t+1)} \leftarrow \{\, h \in H^{(t)} : h(x^{(t)}, \tau_1^{(t)}, \ldots, \tau_{y^{(t)}}^{(t)}) = \text{NO} \,\}$
  else if $y^{(t)} > \hat{y}^{(t)}$ **then**
    Update $H^{(t+1)} \leftarrow \{\, h \in H^{(t)} : h(x^{(t)}, \tau_1, \ldots, \tau_{\hat{y}^{(t)}}) = \text{YES} \,\}$
  end if
end for

To maximize the weight of the minimum path, we construct the tree as follows. Starting from the root node, we greedily split the node, v with the minimum root-to-v cumulative weight. Each node is split by adding an s-edge with weight $d > 1$ and a c-edge with weight 1. Letting $L(w) \in \mathbb{N}$ be the minimum number of leaf nodes needed to construct a tree with minimum path weight at least $w \in \mathbb{R}_{\geq 0}$, we get the following linear recurrence relation $L(w) = L(w-1) + L(w-d)$ where $L(w) = 1$ if $w \leq 0$. We will now show that $L(w) = \Theta(r^w)$, where r is the solution to

$$r^{-d} + r^{-1} = 1. \tag{1}$$

We start by showing that there exists some constant C_1 such that for all $w \in \mathbb{R}_{\geq 0}$ $L(w) \leq C_1 r^w$. We proceed by induction. For the base case, we choose $C_1 \geq \max_{0 \leq t \leq d} L(t) r^{-t}$, so the statement holds trivially. Assuming that the statement holds for all $t \leq T-1$, we get

$$\begin{aligned} L(T) &= L(T-1) + L(T-d) \\ &\leq C_1 r^{T-1} + C_1 r^{T-d} \\ &= C_1 r^T (r^{-1} + r^{-d}) \\ &= C_1 r^T. && \text{(from Equation (1))} \end{aligned}$$

A similar argument shows that there exists C_2 such that for all $w \in \mathbb{R}_{\geq 0}$ $L(w) \geq C_2 r^w$. We let

$W^* = \sup\{T : L(T) \leq N\}$. Using $L(T) = \Theta(r^T)$,

$$L(T) \leq N \implies T \leq \frac{\log N}{\log r}.$$

Therefore, $W^* = \frac{\log N}{\log r} + O(1)$. We will now show that $\frac{1}{\log r} = \Theta(\frac{d}{\log d})$. We have that $r^{-d} + r^{-1} = 1$. For $r = 1/u$, we get $u^d + u = 1$. So for large d, the solution is of the form $u = 1 - \delta$ for small $\delta > 0$. We have that $(1-\delta)^d \approx e^{-d\delta}$. So we get $e^{-d\delta} \approx \delta$ or $ye^y \approx d$ for $y = d\delta$. Equivalently, $W(d) = y \iff y \approx \ln d - \ln\ln d + o(1)$, where W is the Lambert W function. So we get that $\delta = \frac{\ln d}{d}$ and $u = \frac{1}{r} = 1 - \delta \implies \ln r \approx \delta = \frac{\ln d}{d}$. So we get that $W^* = \Theta(\frac{d}{\log d} \log N)$, which concludes the proof.

□

D. Alternate Mistake Model

In this section, we analyze learnability under an alternate model for the learner's types of mistakes. We are interested in the trade-off between soundness and completeness mistakes (at the sequence level this time) via a linear cost objective. Specifically, the learner incurs a cost of $\gamma_s \in \mathbb{R}_{\geq 0}$ for every sequence-level soundness mistake and cost $\gamma_c \in \mathbb{R}_{\geq 0}$ for every sequence-level completeness mistake. We additionally introduce the notion of a "location" mistake, for which the learner incurs cost $\gamma_l \in \mathbb{R}_{\geq 0}$. Formally, we distinguish between the following three types of mistakes:

1. **(Sequence-level) soundness mistake**, which occurs when the learner accepts an incorrect reasoning trace. Formally, the learner makes a soundness mistake in round t if $\hat{y}^{(t)} = \infty$ but $y^{(t)} \in [L]$.
2. **(Sequence-level) completeness mistake**, which occurs when the learner indicates error in a correct reasoning trace. Formally, the learner makes a completeness mistake in round t if $\hat{y}^{(t)} \in [L]$ but $y^{(t)} = \infty$.
3. **Location mistake**, which occurs when the learner rejects an incorrect reasoning trace, but mispredicts the location of the first faulty step. Formally, the learner makes a location mistake in round t if $\hat{y}^{(t)}, y^{(t)} \in [L]$ but $\hat{y}^{(t)} \neq y^{(t)}$.

For convenience, we drop the term "sequence-level" in the remainder of this section. The learner's objective is to minimize the linear cumulative cost $\gamma_s M_s + \gamma_c M_c + \gamma_l M_l$, where M_s, M_c, and M_l is the number of soundness, completeness, and location mistakes that the learner makes over T timesteps. Here we assume that $\gamma_s \geq \gamma_c \geq \gamma_l$.

As in Section A.3, our goal here is to define the right notion of a mistake tree that will capture the optimal strategy of an adversary against any online verification algorithm. However, here the optimal strategy of the adversary depends on the learner's prediction in a way that might not come down to a binary decision. Specifically, if a learner predicts the first faulty step of a reasoning trace to be $\hat{y}^{(t)} \in [L]$, the adversary could induce either a completeness mistake or a location mistake and for the latter the optimal label $y^{(t)} \in [L]$ for the adversary to choose might differ depending on the learner's prediction. Moreover, as is typical in mistake trees, edges are labeled with an element in the label set and in this setting a label in $[L]$ could be used to induce either a location or a completeness mistake, so the tree structure must be expressive enough to distinguish between the two. Another challenge is the real-valued cost for each mistake, which must be accounted for in the tree structure.

We now introduce the notion of an SCL-mistake tree, which has weighted edges of different types designed to handle the complexity of our online verification setting.

SCL-mistake tree. An SCL-mistake tree is a rooted binary tree, where every internal node is labeled with an example in $X \times \Sigma^L$ and every edge is labeled in $[L] \cup \{\infty\}$, such that the outgoing edges from a single node have different labels. There are 3 possible types of edges: an edge labeled with $i \in [L]$ may be either an *s-edge* or an *l-edge*. An edge labeled ∞ is always a *c-edge*. For every node, exactly one of the following must hold: (a) the node has two outgoing l-edges, or (b) the node has one outgoing s-edge and one outgoing c-edge. Each edge e has a corresponding weight $w(e)$, which is equal to $\gamma_s, \gamma_c, \gamma_l$ if e is an s-, c-, and l-edge, respectively.

Algorithm 11 SCL-SOA

Input: class of verifiers H, mistake costs $\gamma_s, \gamma_l, \gamma_c \in \mathbb{R}_{\geq 0}$
$H^{(1)} \leftarrow H$ — set initial version space
for $t \in [T]$ **do**
  Observe example $z^{(t)} = \left(x^{(t)}, \tau^{(t)} = (\tau_1^{(t)}, \ldots, \tau_L^{(t)})\right)$
  Compute $\mathcal{Y}^{(t)} = \{i \in [L] \cup \{\infty\} \mid \exists h \in H^{(t)} \text{ such that } h^{\text{Ver}-L}(z^{(t)}) = i\}$ — All possible first faulty steps
  if $|\mathcal{Y}^{(t)}| = 1$ **then**
    Predict $\hat{y}^{(t)} \in \mathcal{Y}^{(t)}$
  else
    for *each* $i \in \mathcal{Y}^{(t)}$ **do**
      Compute $m_i = \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(i,j) + \text{SCL-Ldim}\left(H^{(t)}[(z^{(t)}, j)]\right)\right]$
    end for
    Predict $\hat{y}^{(t)} = \arg\min_{i \in \mathcal{Y}^{(t)}} m_i$
  end if
  Receive correct label $y^{(t)} \in \mathcal{Y}$
  Set $H^{(t+1)} \leftarrow H^{(t)}[(z^{(t)}, y^{(t)})]$ — Update version space
end for

We say that an SCL mistake tree is *shattered* by a hypothesis class H if for any root-to-leaf path that traverses internal nodes $z_1, z_2, \ldots, z_d$ there exists a hypothesis $h \in H$ such that the label of edge (z_i, z_{i+1}) is $h^{\text{Ver-CoT}}(z_i)$ for all $i \leq d$.

We highlight that the SCL mistake tree is a binary tree, despite the adversary potentially needing more than two labels to play optimally (which would therefore require more than two children per node). We find that this subtlety is resolved when $\gamma_s \geq \gamma_c \geq \gamma_l$, which is a mild assumption in our online verification setting.

Definition D.1 (SCL-Littlestone dimension). For a class of verifiers, H, the SCL-Littlestone dimension, SCL-Ldim(H), is defined as the supremum $w \in \mathbb{R}$ such that there exists an SCL mistake tree, where all root-to-leaf paths have cumulative weight at least w.

If $H = \emptyset$, then SCL-Ldim$(H) = 0$. If for all $w \in \mathbb{R}_{\geq 0}$ there exists an SCL-mistake tree shattered by H, where all root-to-leaf paths have weight at least w, then we say that SCL-Ldim$(H) = \infty$. Next we show that any deterministic verification algorithm using H must suffer cumulative loss at least SCL-Ldim(H).

Theorem D.2. *For any hypothesis class, $\mathcal{H}$, and any deterministic learning algorithm, there exists a realizable sequence such that the learner must suffer cumulative loss at least W for every $W <$ SCL-Ldim(H).*

The proof of Theorem D.2 follows similarly to that of Theorem A.9, so we omit it.

We next analyze the performance of Algorithm 11. Here, we denote the immediate loss for a learner's prediction $\hat{y}$ and a true prediction y as $\ell(\hat{y}, y) \in \{0, \gamma_s, \gamma_c, \gamma_l\}$.

Theorem D.3. *For any class of verifiers, H, SCL-SOA (Algorithm 4) achieves cumulative loss at most SCL-Ldim(H) on any realizable sequence of examples.*

Proof. We will show that for every timestep t, $\ell(\hat{y}^{(t)}, y^{(t)}) \leq \text{SCL-Ldim}(H^{(t)}) - \text{SCL-Ldim}(H^{(t+1)})$. We consider the prediction $\hat{y}^{(t)}$ of Algorithm 4 in timestep t and get

$$\begin{aligned}\ell(\hat{y}^{(t)}, y^{(t)}) + \text{SCL-Ldim}(H^{(t+1)}) &\leq \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{y}^{(t)}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] \\ &= \min_{i \in \mathcal{Y}^{(t)}} \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(i,j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)])\right].\end{aligned}$$

From the definition of the SCL-Littlestone dimension, we also know that

$$\text{SCL-Ldim}(H^{(t)}) \geq \max_{\{k_1,k_2\} \subseteq \mathcal{Y}^{(t)}} \min_{i \in \{k_1,k_2\}} \left[\gamma_i + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)])\right]$$

where k_1, k_2 are the edges of the root node, $x^{(t)}$, in an SCL mistake tree. So it suffices to show that

$$\min_{i \in \mathcal{Y}^{(t)}} \max_{j \in \mathcal{Y}^{(t)}} \left[\ell(i,j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] \leq \max_{\{k_1,k_2\} \subseteq \mathcal{Y}^{(t)}} \min_{i \in \{k_1,k_2\}} \left[\gamma_i + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right]$$

First, we observe that if $|\mathcal{Y}^{(t)}| \leq 1$ we are done. Otherwise we consider the following three cases. **Case 1:**

$$\gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)]) \geq \max_{i \in \mathcal{Y}^{(t)}} \left[\gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right]$$

where we will denote with i^* the arg max of the RHS. Given that $\gamma_s \geq \gamma_l$, it follows that $x^{(t)}$ must have one outgoing s-edge labeled i^* and one outgoing c-edge labeled ∞, where

$$\begin{aligned} &\max_{\{k_1,k_2\} \subseteq \mathcal{Y}^{(t)}} \min_{i \in \{k_1,k_2\}} \left[\gamma_i + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right] \\ &= \min \left\{ \gamma_s + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i^*)]), \gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)]) \right\} \end{aligned} \tag{2}$$

If $\hat{i} \in [L]$, then $\max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] = \gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)])$ which is at most equal to (2), or $\hat{i}$ is suboptimal.

If $\hat{i} = \infty$, then

$$\begin{aligned} &\max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] \\ &= \min \left\{ \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)]), \gamma_s + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i^*)]) \right\} \end{aligned}$$

which again is at most equal to (2).

Case 2:

$$\gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)]) \leq \max_{i \in \mathcal{Y}^{(t)} \setminus \{\max \mathcal{Y}^{(t)}\}} \left[\gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right]$$

where we will denote the maximizing and the second maximizing element of the RHS $i_1^*, i_2^* \in [L]$, respectively. Then $x^{(t)}$ must have two distinct outgoing l-edges i_1^* and i_2^*, such that

$$\max_{\{k_1,k_2\} \subseteq \mathcal{Y}^{(t)}} \min_{i \in \{k_1,k_2\}} \left[\gamma_i + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right] = \gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_2^*)]) \tag{3}$$

If $\hat{i} = i_1^*$, then $\max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] = \gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_2^*)])$, which is exactly (3).

If $\hat{i} \in [L]$,but $\hat{i} \neq i_1^*$, then

$$\max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] = \gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_1^*)])$$

, which means that $\hat{i}$ is suboptimal.

If $\hat{i} = \infty$, then $\max_{j \in \mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)]) \right] = \gamma_s + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_1^*)])$, which means that $\hat{i}$ is suboptimal.

Case 3:

$$\begin{aligned} \max_{i \in \mathcal{Y}^{(t)} \setminus \{\max \mathcal{Y}^{(t)}\}} \left[\gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right] &\leq \gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)]) \\ &\leq \max_{i \in \mathcal{Y}^{(t)}} \left[\gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)]) \right] \end{aligned}$$

where we will denote the maximizing and the second maximizing element of the RHS $i_1^*, i_2^* \in [L]$, respectively. Then $x^{(t)}$ must have one outgoing s-edge i_1^* and one outgoing c-edge ∞, where

$$\max_{\{k_1,k_2\}\subseteq \mathcal{Y}^{(t)}} \min_{i\in\{k_1,k_2\}} \left[\gamma_i + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i)])\right] = \gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)] \tag{4}$$

If $\hat{i} = i_1^*$, then $\max_{j\in\mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] = \gamma_c + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, \infty)])$, which is equal to (4).

If $\hat{i} \in [L]$ but $\hat{i} \neq i_1^*$, then

$$\max_{j\in\mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] = \gamma_l + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_1^*)]),$$

which means that $\hat{i}$ is suboptimal.

If $\hat{i} = \infty$, then $\max_{j\in\mathcal{Y}^{(t)}} \left[\ell(\hat{i}, j) + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, j)])\right] = \gamma_s + \text{SCL-Ldim}(H^{(t)}[(z^{(t)}, i_1^*)])$, which means that $\hat{i}$ is suboptimal since $\gamma_s \geq \gamma_l$. □

We note that when $\gamma_s = \gamma_c = \gamma_l = 1$, the SCL-Littlestone dimension recovers the definition of the Littlestone dimension. We now present an example that shows an $O(L)$ separation between the setting, where $\gamma_l = 0$ and $\gamma_l = 1$. This highlights the added difficulty of learning verifiers that correctly predict the location of the first faulty step for a given reasoning trace.

Example D.4. Consider an example where correct reasoning traces correspond to satisfying assignments of an unknown conjunction of size L. Let $x_1, x_2, \ldots, x_L$ be Boolean variables, and let $H = \left\{h_S : \{0,1\}^L \to \{0,1\} \;\middle|\; h_S(x) = \bigwedge_{i\in S} x_i \wedge \bigwedge_{j\in T} \neg x_j,\ S, T \subseteq [L],\ |S| + |T| = L\right\}$ be the class of verifiers. Let h_{S^*} be the unknown target conjunction. A reasoning trace $\tau \in \{0,1\}^L$ is a Boolean assignment and is correct if and only if $h_{S^*}(\tau) = 1$. Let $\gamma_s = \gamma_c = 1$. If $\gamma_l = 0$, then the learner can learn h_{S^*} with at most 1 completeness mistake by rejecting every reasoning trace. A single correct proof will indicate exactly which literals are negated. On the other hand, if $\gamma_l > 0$, the learner can be forced to incur loss $\gamma_l \cdot L/2$. To see this, notice that an adversary can ask the learner to verify an assignment at timestep t and reveal the true incorrect step to be either step $2t - 1$ or $2t$, depending on which step the learner did not flag as incorrect. Thus, the adversary can guarantee a location mistake for at least $L/2$ timesteps.

E. Sequential Harnessing: Connections with Imitation Learning and Online Chain-of-Thought Generation

Many of the results in this paper involve reductions between different kinds of learning problems. In this section, we provide a generic sequential-learning reduction framework for reductions of this type. While this framework is not able to capture the asymmetry and trade-off between soundness and completeness mistakes, we show it can capture several important interactive learning settings, including imitation learning, chain-of-thought verification, verifier-guided reasoning, and online chain-of-thought generation. First, we place our framework in a broader sequential-learning context by identifying a common "harnessing"[5] argument that appears implicitly in several online reductions, especially in imitation learning. Then we show how this argument directly yields baseline online learnability guarantees for reasoning and verification systems. It also yields a clean online chain-of-thought generation result, while eliminating the $O(\log L)$ chain length factor from prior work (Joshi et al., 2025).

Notation. We first recall the usual online learning terminology. An online prediction problem P is defined over an instance (input) space I, a label (output) space Y, and predictors $H \subseteq Y^I$. A labeled sequence $G_T = ((i^{(1)}, y^{(1)}), \ldots, (i^{(T)}, y^{(T)})) \in \Gamma := (I \times Y)^*$ is *realizable* for P if there is a function $h^\star \in H$ such that $y^{(t)} = h^\star(i^{(t)})$ for every t. A (deterministic) online learner for P is a map from labeled histories to predictors $A : \Gamma \to Y^I$. The predictor $A(G_T)$ need not be in H. On round t, after seeing the history $G_{t-1} := \left((i^{(1)}, y^{(1)}), \ldots, (i^{(t-1)}, y^{(t-1)})\right)$, the

[5]The term "harnessing" or "scaffolding" (Suzgun & Kalai, 2024) has been recently coined to describe complementary models or tools that augment LLMs in certain specific ways.

learner sets $\hat{h}_t := A(G_{t-1})$ and predicts $\hat{y}^{(t)} = \hat{h}_t(i^{(t)})$, and then observes $y^{(t)}$. Its mistake count on the sequence is $\sum_t \mathbf{1}\{\hat{y}^{(t)} \neq y^{(t)}\}$. We write $\mathrm{MB}(A, \mathsf{P})$ for the smallest M such that A makes at most M mistakes on every realizable sequence for P, and $\mathrm{MB}(\mathsf{P}) := \inf_A \mathrm{MB}(A, \mathsf{P})$ for the optimal realizable mistake bound of the problem.

We now define some new definitions for the sequential harnessing framework. For the remainder of this section, fix two online learning problems: P_{base} over $(I_{\text{base}}, Y_{\text{base}}, H_{\text{base}})$, and P_{sys} over $(I_{\text{sys}}, Y_{\text{sys}}, H_{\text{sys}})$.

Definition E.1 (Harness). Fix a *base* instance-label space pair $(I_{\text{base}}, Y_{\text{base}})$ and a *system* instance-label space pair $(I_{\text{sys}}, Y_{\text{sys}})$. The base would correspond to the LLM in the context of LLM harnessing, although we will see that the abstraction is useful in a variety of settings. Let $\Gamma_{\text{base}} := (I_{\text{base}} \times Y_{\text{base}})^*$ be the set of finite labeled base *transcripts*. A *harness from* $(I_{\text{base}}, Y_{\text{base}})$ *to* $(I_{\text{sys}}, Y_{\text{sys}})$ is a deterministic map

$$S : I_{\text{sys}} \times \Gamma_{\text{base}} \to I_{\text{base}} \sqcup Y_{\text{sys}},$$

where the disjoint union denotes two types of harness actions, a base query or a final system output. As shown below, this captures a procedure that turns any base predictor $h : I_{\text{base}} \to Y_{\text{base}}$ into a system predictor by adaptively querying base examples and then returning one system output.

In a typical usage, the base transcript input to the harness is generated autoregressively. The harness induces a map from base to system predictors $S[\cdot] : Y_{\text{base}}^{I_{\text{base}}} \to Y_{\text{sys}}^{I_{\text{sys}}}$, as follows For $h : I_{\text{base}} \to Y_{\text{base}}$ and $i \in I_{\text{sys}}$, initialize the base transcript $\gamma_0 = \emptyset$. If $S(i, \gamma_{t-1}) = q_t \in I_{\text{base}}$, set $\gamma_t = (\gamma_{t-1}, (q_t, h(q_t)))$; if $S(i, \gamma_m) = y \in Y_{\text{sys}}$, halt and define harness map $S[h](i) = y$ and trace $\mathrm{Trace}_S^h(i) = \gamma_m$, the base transcript generated by S. We assume this recursion halts after finitely many base queries for every h and i. We will also need the notion of a decoder.

Definition E.2 (Decoder and harness-decoder consistency). A *decoder* is a function $D : I_{\text{sys}} \times Y_{\text{sys}} \to \Gamma_{\text{base}}$ that gives a base transcript given a system input and output. A harness-decoder pair (S, D) is *consistent*, if for every system predictor $g \in H_{\text{sys}}$ there is a base predictor for which the harness S matches its behavior and the decoder recovers its trace. That is, there exists $h_g \in H_{\text{base}}$, such that for every $i \in I_{\text{sys}}$, $g(i) = S[h_g](i)$ and $D(i, g(i)) = \mathrm{Trace}_S^{h_g}(i)$.

We show that given a consistent harness-decoder pair, it is possible to construct a system predictor that preserves the mistake bounds of the base predictor (Algorithm 12). This result is based on a simple inductive argument over the base transcript. The idea appears implicitly in prior work on online-to-online reductions in imitation learning (Ross et al., 2011). The reduction implies that in order to obtain a good system performance, it is sufficient to construct a consistent harness-decoder pair for the problems.

Theorem E.3 (Online-to-online reduction). *Let* P_{base} *and* P_{sys} *be two online learning problems as defined above. Suppose there is a harness* S *consistent with a decoder* D*. Let* $R_{S,D}$ *be the online-to-online reduction defined in Algorithm 12 Then, for every base learner* $A_{\text{base}} : \Gamma_{\text{base}} \to Y_{\text{base}}^{I_{\text{base}}}$,

$$\mathrm{MB}(R_{S,D}(A_{\text{base}}), \mathsf{P}_{\text{sys}}) \leq \mathrm{MB}(A_{\text{base}}, \mathsf{P}_{\text{base}}).$$

In particular, $\mathrm{MB}(\mathsf{P}_{\text{sys}}) \leq \mathrm{MB}(\mathsf{P}_{\text{base}})$.

Proof of Theorem E.3. Let $A_{\text{sys}} := R_{S,D}(A_{\text{base}})$. Fix a realizable system sequence

$$G_T^{\text{sys}} = ((i^{(1)}, y^{(1)}), \ldots, (i^{(T)}, y^{(T)})),$$

and, by system realizability, choose a witness $g^\star \in H_{\text{sys}}$. By the harness-decoder consistency assumption, there is $h^\star \in H_{\text{base}}$ such that, for every $i \in I_{\text{sys}}$,

$$g^\star(i) = S[h^\star](i), \qquad D(i, g^\star(i)) = \mathrm{Trace}_S^{h^\star}(i).$$

The first identity is used to identify the true system label with the harness output

$$y^{(t)} = g^\star(i^{(t)}) = S[h^\star](i^{(t)}).$$

[6]For realizable system histories covered by Theorem E.3, the proof of Theorem E.3 shows that this disagreement exists whenever $\hat{y}_t \neq y_t$.

Algorithm 12 Online-to-online reduction $R_{S,D}$

Input: base learner $A_{\text{base}} : \Gamma_{\text{base}} \to Y_{\text{base}}^{I_{\text{base}}}$; harness $S : I_{\text{sys}} \times \Gamma_{\text{base}} \to I_{\text{base}} \sqcup Y_{\text{sys}}$; decoder $D : I_{\text{sys}} \times Y_{\text{sys}} \to \Gamma_{\text{base}}$
Output: system learner $A_{\text{sys}} : \Gamma_{\text{sys}} \to Y_{\text{sys}}^{I_{\text{sys}}}$
Function $A_{\text{sys}}(G)$
Let $G = ((i_1, y_1), \ldots, (i_m, y_m))$
Initialize base transcript $B_0 := \emptyset$
for $t = 1$ **to** m **do**
  Define the current base predictor $\hat{h}_t := A_{\text{base}}(B_{t-1})$
  Let $\hat{T}_t := \text{Trace}_S^{\hat{h}_t}(i_t)$ and $\hat{y}_t := S[\hat{h}_t](i_t)$ (transcript and output from the harness)
  if $\hat{y}_t \neq y_t$ **then**
    Let $T_t := D(i_t, y_t)$ be the decoded witness transcript
    Let (z_t, a_t) be the first base-label disagreement between $\hat{T}_t$ and T_t, with a_t taken from T_t[6]
    Define $B_t := (B_{t-1}, (z_t, a_t))$
  else
    Define $B_t := B_{t-1}$
  end if
end for
Define the final base predictor $\hat{h}_{m+1} := A_{\text{base}}(B_m)$
Return $S[\hat{h}_{m+1}]$
Return A_{sys}

Let B_t be the base history produced by Algorithm 12 after processing the first t labeled system examples. We prove by induction that B_t is realizable by $h^\star$, that $|B_t|$ is the number of system mistakes in the first t rounds, and that A_{base} makes a mistake on every example of B_t in order.

The claim is trivial for $B_0 = \emptyset$. Suppose it holds through round $t-1$, and let

$$\hat{h}_t := A_{\text{base}}(B_{t-1}), \qquad \hat{y}^{(t)} := S[\hat{h}_t](i^{(t)}).$$

If $\hat{y}^{(t)} = y^{(t)}$, Algorithm 12 sets $B_t = B_{t-1}$, so the induction claim is unchanged.

Now suppose $\hat{y}^{(t)} \neq y^{(t)}$. The decoded transcript is

$$T^\circ := D(i^{(t)}, y^{(t)}) = D(i^{(t)}, g^\star(i^{(t)})) = \text{Trace}_S^{h^\star}(i^{(t)}).$$

Here we used the second decoder-consistency identity. $S[\hat{h}_t](i^{(t)})$ and $S[h^\star](i^{(t)})$ have different final outputs. S is a deterministic map of the current system input and labeled base transcript, so two executions with the same transcript history take the same next action and cannot halt with different final outputs. Hence there is a first base-label disagreement between $\text{Trace}_S^{\hat{h}_t}(i^{(t)})$ and T°. Write this disagreement as $(z, h^\star(z))$ on the witness transcript. Then

$$\hat{h}_t(z) \neq h^\star(z),$$

so the appended example $(z, h^\star(z))$ is both realizable by the base-class witness $h^\star \in H_{\text{base}}$ and a genuine mistake of A_{base} on the current base history B_{t-1}. Algorithm 12 appends exactly this one example, hence $|B_t| = |B_{t-1}| + 1$ exactly when the system learner makes a mistake.

Thus the final base sequence B_T is realizable and A_{base} makes exactly $|B_T|$ mistakes on it, where $|B_T|$ is the number of system mistakes. Taking the worst case over realizable system sequences gives

$$\text{MB}(R_{S,D}(A_{\text{base}}), \mathsf{P}_{\text{sys}}) \leq \text{MB}(A_{\text{base}}, \mathsf{P}_{\text{base}}).$$

Taking the infimum over A_{base} gives $\text{MB}(\mathsf{P}_{\text{sys}}) \leq \text{MB}(\mathsf{P}_{\text{base}})$. □

Note that this template tracks only a single total mistake objective in contrast with the rest of the paper, where the main technical challenge is to separately track soundness and completeness mistakes. We now illustrate the versatility of this framework by applying it to previously studied problems as well as the settings in this work.

E.1. Imitation Learning as Sequential Harnessing

We now show that the sequential harnessing framework naturally recovers classical imitation learning reductions such as DAgger (Ross et al., 2011). Intuitively, the base predictor corresponds to a local policy that predicts the next action from the current state, while the system predictor corresponds to the induced trajectory-level policy.

Imitation learning setup. Let X denote a set of contexts/tasks, $\mathcal{S}$ denote a state space, $\mathcal{A}$ denote an action space, L denote the horizon length, and $U : X \times \mathcal{A}^{<L} \to \mathcal{S}$ denote the deterministic state-transition/unrolling map. Given a context $x \in X$ and an action sequence $a_{1:\ell-1} \in \mathcal{A}^{\ell-1}$, the induced state at time ℓ is $s_\ell = U(x, a_{1:\ell-1})$.

Let $\Pi \subseteq \mathcal{A}^{\mathcal{S}}$ denote a policy class. Each policy $\pi \in \Pi$ induces a trajectory predictor $g_\pi : X \to \mathcal{A}^L$ defined by

$$a_\ell = \pi(s_\ell), \qquad s_\ell = U(x, a_{1:\ell-1}), \qquad g_\pi(x) = (a_1, \ldots, a_L).$$

Base and system prediction problems. Define the base prediction problem by $I_{\text{base}} = \mathcal{S}, Y_{\text{base}} = \mathcal{A}, H_{\text{base}} = \Pi$. Thus, the base learner predicts a local action from a state. Define the system prediction problem by $I_{\text{sys}} = X, Y_{\text{sys}} = \mathcal{A}^L$, and $H_{\text{sys}} = \{g_\pi : \pi \in \Pi\}$. Thus, the system learner predicts an entire trajectory.

Harness construction. Let $\Gamma_{\text{base}} = (\mathcal{S} \times \mathcal{A})^*$. For $x \in X, \gamma = ((s_1, a_1), \ldots, (s_m, a_m)) \in \Gamma_{\text{base}}$, define the harness $S_{\text{IL}} : X \times \Gamma_{\text{base}} \to I_{\text{base}} \sqcup Y_{\text{sys}}$ by

$$S_{\text{IL}}(x, \gamma) = \begin{cases} U(x, a_{1:m}), & m < L, \\ (a_1, \ldots, a_L), & m = L. \end{cases}$$

That is, the harness repeatedly queries the current state, receives an action prediction from the base learner, and after L steps outputs the induced trajectory. For any policy $\pi \in \Pi$, the induced system predictor satisfies

$$S_{\text{IL}}[\pi](x) = g_\pi(x).$$

Decoder construction. Given a trajectory label $y = (a_1, \ldots, a_L) \in \mathcal{A}^L$, define the decoder $D_{\text{IL}} : X \times \mathcal{A}^L \to (\mathcal{S} \times \mathcal{A})^L$ by $D_{\text{IL}}(x, y) = \big((s_1, a_1), \ldots, (s_L, a_L)\big)$, where $s_\ell = U(x, a_{1:\ell-1})$. Thus, the decoder converts an expert trajectory into the corresponding sequence of state-action supervision examples.

Consistency of the harness-decoder pair. It is straightforward to verify the consistency condition of the sequential harnessing framework. Fix any $g \in H_{\text{sys}}$. By definition of H_{sys}, there exists a policy $\pi \in \Pi$ such that $g = g_\pi$. Let $h_g := \pi$. Then for every $x \in X$,

$$S_{\text{IL}}[h_g](x) = S_{\text{IL}}[\pi](x) = g_\pi(x) = g(x).$$

Moreover, if $g(x) = (a_1, \ldots, a_L)$, then the rollout trace generated by the harness equals

$$\text{Trace}^{\pi}_{S_{\text{IL}}}(x) = \big((U(x, \emptyset), a_1), (U(x, a_1), a_2), \ldots, (U(x, a_{1:L-1}), a_L)\big),$$

which is exactly $D_{\text{IL}}(x, g(x))$. Hence, $(S_{\text{IL}}, D_{\text{IL}})$ is a consistent harness-decoder pair.

Applying the sequential harnessing theorem now yields an online-to-online reduction in the imitation learning setting, and the argument appears implicitly in prior work (Ross et al., 2011).

E.2. Prefix and Full CoT Verifier Equivalence as Sequential Harnessing

We now show that the two online-to-online reductions in Section A.2 can be viewed as special cases of the sequential harnessing framework. In this subsection we ignore the distinction between soundness and completeness mistakes and instead use the standard 0–1 mistake model, where a learner makes a mistake whenever its predicted label differs from the true label. There is one technical point: prefix verification is a promised prediction problem, since the verifier is only required to label a prefix when all previous steps are correct. We formalize this using partial concept classes, following the framework of Alon et al. (2021). A partial online prediction problem is a triple $\mathsf{P} = (I, Y, H)$ with $H \subseteq (Y \cup \{\star\})^I$, where $\star$ denotes an undefined label. A labeled sequence $((i^{(1)}, y^{(1)}), \ldots, (i^{(T)}, y^{(T)})) \in (I \times Y)^*$ is realizable if there exists $h^\star \in H$ such that

$h^\star(i^{(t)}) = y^{(t)} \neq \star$ for every t. The learner still outputs total predictors in Y^I, and $\mathrm{MB}(A, \mathsf{P})$ is the worst-case number of mistakes over realizable labeled sequences in this partial sense.

The two online problems. Let the prefix-verification problem be $P_{\text{pfx}} = (I_{\text{pfx}}, Y_{\text{pfx}}, H_{\text{pfx}})$, where $I_{\text{pfx}} = X \times \Sigma^{\leq L}$, $Y_{\text{pfx}} = \{\text{YES}, \text{NO}\}$, and $H_{\text{pfx}} = H \subseteq (Y_{\text{pfx}} \cup \{\star\})^{I_{\text{pfx}}}$. For a prefix verifier $h \in H$, the label $h(x, \tau_{1:\ell})$ is defined exactly on the promised prefixes:

$$h(x, \tau_{1:\ell}) \neq \star \quad \Longleftrightarrow \quad h(x, \tau_{1:j}) = \text{YES} \text{ for every } j < \ell.$$

On such prefixes, h predicts whether the last step is valid. Thus, realizable prefix-verification sequences only contain prefixes satisfying the target-dependent premise, but the learner is not given this domain. Let the full chain-of-thought verification problem be $P_{\text{cot}} = (I_{\text{cot}}, Y_{\text{cot}}, H_{\text{cot}})$, where $I_{\text{cot}} = X \times \Sigma^L$ and $Y_{\text{cot}} = [L] \cup \{\infty\}$. Each $h \in H$ induces a full-trace verifier $g_h(x, \tau_{1:L}) = \min(\{\ell \in [L] : h(x, \tau_{1:\ell}) = \text{NO}\} \cup \{\infty\})$. Thus, $H_{\text{cot}} = \{g_h : h \in H\}$. The class H_{cot} is total, and we regard it as a partial class with no undefined labels.

Section A.2 proves reductions in both directions between these two online problems. We now show that both the reductions can be viewed as harness-decoder constructions, in the 0-1 mistake setting.

The harness itself is unchanged in the partial setting: Algorithm 12 applies S only to the total predictors output by the base learner. Partiality only changes which labeled sequences are realizable, and which decoded witness transcripts are valid supervision for the base learner.

Theorem E.4 (Promised online-to-online reduction). *Let P_{base} and P_{sys} be partial online prediction problems. Suppose there is a harness S consistent with a decoder D on the promise in the following sense: for every $g \in H_{\text{sys}}$ there exists $h_g \in H_{\text{base}}$ such that for every system input i with $g(i) \neq \star$, writing*

$$D(i, g(i)) = ((q_1, a_1), \ldots, (q_m, a_m)),$$

we have $h_g(q_r) = a_r \neq \star$ for every $r \leq m$, and the decoded transcript is exactly a valid execution transcript for the harness:

$$S(i, ((q_1, a_1), \ldots, (q_{r-1}, a_{r-1}))) = q_r \quad \textit{for every } r \leq m, \qquad S(i, D(i, g(i))) = g(i).$$

Let $R_{S,D}$ be the online-to-online reduction defined in Algorithm 12. Then, for every base learner $A_{\text{base}} : \Gamma_{\text{base}} \to Y_{\text{base}}^{I_{\text{base}}}$,

$$\mathrm{MB}(R_{S,D}(A_{\text{base}}), \mathsf{P}_{\text{sys}}) \leq \mathrm{MB}(A_{\text{base}}, \mathsf{P}_{\text{base}}).$$

In particular, $\mathrm{MB}(\mathsf{P}_{\text{sys}}) \leq \mathrm{MB}(\mathsf{P}_{\text{base}})$.

Proof. The proof is the same induction as in Theorem E.3, with the quantifiers restricted to defined labels. Fix a realizable system sequence

$$G_T^{\text{sys}} = ((i^{(1)}, y^{(1)}), \ldots, (i^{(T)}, y^{(T)}))$$

and choose a partial witness $g^\star \in H_{\text{sys}}$ such that $g^\star(i^{(t)}) = y^{(t)} \neq \star$ for every t. By the promise-consistency assumption, there is $h^\star \in H_{\text{base}}$ such that for each observed system input $i^{(t)}$, the decoded transcript

$$T^{(t)} := D(i^{(t)}, y^{(t)})$$

is a valid harness transcript ending in $y^{(t)}$, and every labeled example in $T^{(t)}$ is defined and labeled by $h^\star$. Let B_t be the base history produced by Algorithm 12 after the first t system examples. We prove by induction that every labeled example in B_t is defined and labeled by $h^\star$, and that A_{base} made a mistake on each such example.

The claim is trivial at $t = 0$. Suppose it holds through $t - 1$ and let $\hat{h}_t = A_{\text{base}}(B_{t-1})$. If $S[\hat{h}_t](i^{(t)}) = y^{(t)}$, the algorithm appends nothing. Otherwise, the decoded transcript

$$T^\circ := D(i^{(t)}, y^{(t)})$$

is a valid harness transcript ending in $y^{(t)}$ and is defined under $h^\star$. Since the learned harness execution ends in the different output $S[\hat{h}_t](i^{(t)})$, determinism of the harness implies a first base-label disagreement between

$\mathrm{Trace}_S^{\hat{h}_t}(i^{(t)})$ and T°. If this disagreement is (z, a) on the witness transcript, then $h^\star(z) = a \neq \star$ and $\hat{h}_t(z) \neq a$. Thus the algorithm appends a defined base example on which the current base predictor errs.

Therefore the number of system mistakes is at most the number of mistakes made by A_{base} on the realizable partial base sequence B_T. Taking the worst case over realizable partial system sequences gives the first inequality, and taking the infimum over base learners gives the second. □

E.2.1. From prefix verification to full CoT verification

Here the base problem is prefix verification and the system problem is full CoT verification, $P_{\text{base}} = P_{\text{pfx}}$ and $P_{\text{sys}} = P_{\text{cot}}$.

Harness. A base transcript is an element of $\Gamma_{\text{pfx}} = (I_{\text{pfx}} \times Y_{\text{pfx}})^*$. Given a full trace $z = (x, \tau_{1:L}) \in I_{\text{cot}}$ and transcript $\gamma = ((q_1, b_1), \ldots, (q_m, b_m))$, define the harness $S_{\text{pfx}\to\text{cot}} : I_{\text{cot}} \times \Gamma_{\text{pfx}} \to I_{\text{pfx}} \sqcup Y_{\text{cot}}$ as follows. If $m = 0$, the harness queries the first prefix, $S_{\text{pfx}\to\text{cot}}(z, \emptyset) = (x, \tau_{1:1})$. Suppose now that $q_j = (x, \tau_{1:j})$ for all $j \leq m$. If some previous answer equals NO, let $j^\star = \min\{j \leq m : b_j = \text{NO}\}$. The harness halts and outputs $j^\star$. If all previous answers are YES and $m < L$, the harness queries the next prefix, $S_{\text{pfx}\to\text{cot}}(z, \gamma) = (x, \tau_{1:m+1})$. Finally, if all previous answers are YES and $m = L$, the harness halts and outputs ∞.

Equivalently, for any total prefix predictor $\bar{h} : I_{\text{pfx}} \to Y_{\text{pfx}}$,

$$S_{\text{pfx}\to\text{cot}}[\bar{h}](x, \tau_{1:L}) = \min(\{\ell \in [L] : \bar{h}(x, \tau_{1:\ell}) = \text{NO}\} \cup \{\infty\}).$$

This is the only way the reduction uses the harness at inference time: it is always run with the total predictor produced by the base learner.

Decoder. For a full-trace label $y \in [L] \cup \{\infty\}$, define the decoder $D_{\text{pfx}\to\text{cot}} : I_{\text{cot}} \times Y_{\text{cot}} \to \Gamma_{\text{pfx}}$ as follows. If $y \in [L]$, define

$$D_{\text{pfx}\to\text{cot}}((x, \tau_{1:L}), y) = \big(((x, \tau_{1:1}), \text{YES}), \ldots, ((x, \tau_{1:y-1}), \text{YES}), ((x, \tau_{1:y}), \text{NO})\big).$$

If $y = \infty$, define

$$D_{\text{pfx}\to\text{cot}}((x, \tau_{1:L}), \infty) = \big(((x, \tau_{1:1}), \text{YES}), \ldots, ((x, \tau_{1:L}), \text{YES})\big).$$

Thus, the decoder reconstructs the sequence of prefix-verification interactions corresponding to the true location of the first error.

Consistency. Fix any system target $g \in H_{\text{cot}}$. By definition of H_{cot}, there exists a prefix verifier $h \in H_{\text{pfx}}$ such that $g = g_h$; this h is the base witness. For every full trace $z = (x, \tau_{1:L})$, let $W(z) = D_{\text{pfx}\to\text{cot}}(z, g(z))$. If $g(z) = y \in [L]$, then $W(z)$ consists of the labels YES on prefixes $1, \ldots, y-1$ and NO on prefix y, all of which are defined by the partial prefix verifier h. If $g(z) = \infty$, then $W(z)$ consists of the labels YES on all prefixes, again all defined by h. In both cases, replaying $W(z)$ through $S_{\text{pfx}\to\text{cot}}$ makes the harness query exactly those prefixes and halt with output $g(z)$. Thus $(S_{\text{pfx}\to\text{cot}}, D_{\text{pfx}\to\text{cot}})$ satisfies the promise-consistency condition of Theorem E.4. Applying Theorem E.4 therefore gives $\mathrm{MB}(P_{\text{cot}}) \leq \mathrm{MB}(P_{\text{pfx}})$, recovering the single-error version our result (Theorem A.4).

E.2.2. From full CoT verification to prefix verification

We now recover the reverse reduction. Here the base problem is full CoT verification and the system problem is prefix verification: $P_{\text{base}} = P_{\text{cot}}$ and $P_{\text{sys}} = P_{\text{pfx}}$. As in Theorem A.5, assume there exists a special token $F \in \Sigma$ such that every verifier rejects F after every correct prefix: $h(x, (\tau_{1:\ell-1}, F)) = \text{NO}$ for every $h \in H$, every $x \in X$, and every correct prefix $\tau_{1:\ell-1}$.

Harness. A base transcript is an element of $\Gamma_{\text{cot}} = (I_{\text{cot}} \times Y_{\text{cot}})^*$. Given a prefix instance $z_{\text{pfx}} = (x, \tau_{1:\ell}) \in I_{\text{pfx}}$, define its padded full trace by $\mathrm{pad}_F(x, \tau_{1:\ell}) = (x, \tau_1, \ldots, \tau_\ell, F, \ldots, F) \in X \times \Sigma^L$. Define the harness $S_{\text{cot}\to\text{pfx}} : I_{\text{pfx}} \times \Gamma_{\text{cot}} \to I_{\text{cot}} \sqcup Y_{\text{pfx}}$ as follows. Initially, the harness queries the padded trace: $S_{\text{cot}\to\text{pfx}}(z_{\text{pfx}}, \emptyset) = \mathrm{pad}_F(z_{\text{pfx}})$. After receiving a full-trace label $\hat{y} \in [L] \cup \{\infty\}$, the harness halts and outputs

$$S_{\text{cot}\to\text{pfx}}(z_{\text{pfx}}, ((\mathrm{pad}_F(z_{\text{pfx}}), \hat{y}))) = \begin{cases} \text{NO}, & \hat{y} \leq \ell, \\ \text{YES}, & \hat{y} > \ell. \end{cases}$$

Thus, the harness accepts the original prefix iff the padded full trace survives strictly beyond position ℓ. For any $h \in H$, let g_h denote its induced full-trace verifier. Then, on any promised prefix instance, equivalently on any $(x, \tau_{1:\ell})$ with $h(x, \tau_{1:\ell}) \neq \star$, $S_{\text{cot}\to\text{pfx}}[g_h](x, \tau_{1:\ell}) = h(x, \tau_{1:\ell})$.

Decoder. Define $D_{\text{cot}\to\text{pfx}} : I_{\text{pfx}} \times Y_{\text{pfx}} \to \Gamma_{\text{cot}}$ as follows. If the correct prefix label is NO, define $D_{\text{cot}\to\text{pfx}}((x, \tau_{1:\ell}), \text{NO}) = ((\text{pad}_F(x, \tau_{1:\ell}), \ell))$. If the correct prefix label is YES and $\ell < L$, define $D_{\text{cot}\to\text{pfx}}((x, \tau_{1:\ell}), \text{YES}) = ((\text{pad}_F(x, \tau_{1:\ell}), \ell + 1))$. Finally, if $\ell = L$, define $D_{\text{cot}\to\text{pfx}}((x, \tau_{1:L}), \text{YES}) = ((\text{pad}_F(x, \tau_{1:L}), \infty))$.

Consistency. Fix any system target $h \in H_{\text{pfx}}$ and let g_h denote its induced full verifier; this g_h is the base witness. Consider any promised prefix instance $(x, \tau_{1:\ell})$, equivalently any prefix with $h(x, \tau_{1:\ell}) \neq \star$. The decoded transcript contains a single full-trace example, and its label is defined because g_h is a total full CoT verifier. If $h(x, \tau_{1:\ell}) = \text{NO}$, the promise implies that the padded trace first fails at position ℓ, so the decoded label is ℓ. If $h(x, \tau_{1:\ell}) = \text{YES}$ and $\ell < L$, the forced-failure token makes the first failure occur at position $\ell + 1$. If $h(x, \tau_{1:L}) = \text{YES}$, the padded trace is the original full trace and the decoded label is ∞. In each case, replaying the decoded transcript through $S_{\text{cot}\to\text{pfx}}$ makes the harness halt with the prefix label $h(x, \tau_{1:\ell})$. Thus $(S_{\text{cot}\to\text{pfx}}, D_{\text{cot}\to\text{pfx}})$ satisfies the promise-consistency condition of Theorem E.4, assuming the forced-failure token F. Applying Theorem E.4 yields $\text{MB}(P_{\text{pfx}}) \leq \text{MB}(P_{\text{cot}})$, recovering the single-error version of our result (Theorem A.5).

E.3. Chain-of-Thought Generation via Sequential Harnessing

We now show how the sequential harnessing framework can be used to derive generalization guarantees for autoregressive Chain-of-Thought (CoT) generation. The key idea is that a mistake in a generated trajectory must correspond to a mistake in next-token prediction at the first disagreement location. Sequential harnessing therefore converts online learnability of next-token prediction into online learnability of entire reasoning trajectories.

Setup. Let Σ be a finite token alphabet and let $F \subseteq \Sigma^{\Sigma^*}$ be a class of next-token predictors. For a predictor $f \in F$ and prompt $x \in \Sigma^*$, define the autoregressive trajectory generated by f recursively as follows. Starting from prompt x, define $y_1 = f(x)$ and, for $s \geq 2$, define $y_s = f(x, y_{1:s-1})$. The induced length-T chain-of-thought generator is the map $g_f : \Sigma^* \to \Sigma^T$ defined by $g_f(x) = (y_1, \ldots, y_T)$. We denote the induced trajectory class by $F^{\text{CoT}-T} = \{g_f : f \in F\}$. Our goal is to learn the trajectory class $F^{\text{CoT}-T}$.

Base and system prediction problems. Define the base prediction problem $P_{\text{tok}} = (I_{\text{tok}}, Y_{\text{tok}}, H_{\text{tok}})$ by $I_{\text{tok}} = \Sigma^*$, $Y_{\text{tok}} = \Sigma$, and $H_{\text{tok}} = F$. Thus, a base instance is a token prefix and the label is the next token. Define the system prediction problem $P_{\text{CoT}} = (I_{\text{CoT}}, Y_{\text{CoT}}, H_{\text{CoT}})$ by $I_{\text{CoT}} = \Sigma^*$, $Y_{\text{CoT}} = \Sigma^T$, and $H_{\text{CoT}} = F^{\text{CoT}-T}$. Thus, a system predictor outputs an entire length-T reasoning trajectory.

Harness construction. Let $\Gamma_{\text{tok}} = (I_{\text{tok}} \times Y_{\text{tok}})^*$. Given a prompt x and transcript $\gamma = ((q_1, a_1), \ldots, (q_m, a_m)) \in \Gamma_{\text{tok}}$, define the harness $S_{\text{CoT}} : I_{\text{CoT}} \times \Gamma_{\text{tok}} \to I_{\text{tok}} \sqcup Y_{\text{CoT}}$ as follows. If $m < T$, the harness outputs the next-token prediction query $S_{\text{CoT}}(x, \gamma) = (x, a_{1:m})$. If $m = T$, the harness halts and outputs the trajectory $S_{\text{CoT}}(x, \gamma) = (a_1, \ldots, a_T)$. Thus, the harness repeatedly queries the current prefix, appends the predicted next token, and after T rounds outputs the full generated trajectory. For every $f \in F$, the induced system predictor satisfies $S_{\text{CoT}}[f](x) = g_f(x)$.

Decoder construction. Given a target trajectory $y = (y_1, \ldots, y_T) \in \Sigma^T$, define the decoder $D_{\text{CoT}} : I_{\text{CoT}} \times Y_{\text{CoT}} \to \Gamma_{\text{tok}}$ by $D_{\text{CoT}}(x, y) = \big(((x), y_1), ((x, y_1), y_2), \ldots, ((x, y_{1:T-1}), y_T)\big)$. Thus, the decoder converts a supervised chain-of-thought trajectory into the corresponding sequence of next-token supervision examples.

Consistency. Fix any system predictor $g_f \in H_{\text{CoT}}$ induced by some $f \in F$. Let $h_{g_f} = f$. Then for every prompt x, $S_{\text{CoT}}[f](x) = g_f(x)$. Moreover, the transcript generated by the harness while interacting with f on prompt x is exactly $D_{\text{CoT}}(x, g_f(x))$. Hence $(S_{\text{CoT}}, D_{\text{CoT}})$ is a consistent harness-decoder pair. Applying the sequential harnessing theorem therefore yields the following result.

Theorem E.5 (Online CoT generation via harnessing). *Let $F \subseteq \Sigma^{\Sigma^*}$ be a next-token prediction class and let $F^{\text{CoT}-T}$ denote the induced length-T autoregressive trajectory class. Then $MB(F^{\text{CoT}-T}) \leq MB(F)$. In particular, if $\text{Ldim}(F) = d < \infty$, then there exists an online learner for $F^{\text{CoT}-T}$ that makes at most d mistakes, independent of the trajectory length T.*

Proof. The mistake bound follows immediately from the sequential harnessing theorem and the consistency of $(S_{\text{CoT}}, D_{\text{CoT}})$. Concretely, suppose the learner predicts trajectory $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)$ while the target trajectory is $y = (y_1, \ldots, y_T)$. If $\hat{y} \neq y$, let $s^\star = \min\{s : \hat{y}_s \neq y_s\}$ be the first disagreement location. By definition of $s^\star$, the prefixes agree before time $s^\star$: $\hat{y}_{1:s^\star-1} = y_{1:s^\star-1}$.

Therefore, the trajectory mistake exposes a genuine next-token prediction mistake on the prefix $(x, y_{1:s^\star-1})$: $\hat{f}(x, y_{1:s^\star-1}) = \hat{y}_{s^\star} \neq y_{s^\star} = f^\star(x, y_{1:s^\star-1})$. Thus, every trajectory-level mistake induces a base-level next-token mistake, or, $MB(F^{\text{CoT}-T}) \leq MB(F)$.

Finally, if $\text{Ldim}(F) = d$, the Standard Optimal Algorithm for online realizable multiclass prediction makes at most d mistakes on F, implying the same mistake bound for $F^{\text{CoT}-T}$. □

We now obtain a PAC-learning consequence via the standard online-to-batch conversion.

Corollary E.6 (PAC learnability of CoT generation). *Suppose* $\text{Ldim}(F) = d < \infty$. *Then the autoregressive trajectory class* $F^{\text{CoT}-T}$ *is PAC learnable with sample complexity* $O\left(\frac{d\log(1/\epsilon)+\log(1/\delta)}{\epsilon}\right)$, *independent of the trajectory length* T.

Proof. By the previous theorem, $MB(F^{\text{CoT}-T}) \leq d$. Applying the standard realizable online-to-batch conversion for finite mistake-bound classes yields the desired sample complexity bound. □

Comparison with prior and concurrent work. The work of Joshi et al. (2025) introduced the autoregressive CoT learning model and gave general PAC guarantees based on ERM (consistency in the realizable setting). In particular, their end-to-end Littlestone-dimension guarantee has a $\log T$ dependence, and their available-CoT consistency bound also has a $\log T$ dependence. After the standard online-to-batch conversion, Theorem E.6 gives the analogous CoT-supervised guarantee with $\text{Ldim}(F)$ in place of $\text{Ldim}(F)\log T$. The algorithmic reason is that the learner is not an ERM learner; it runs a realizable online learner for F and updates only on the first wrong next-token witness in each failed rollout. The horizon-independent CoT phenomenon was also obtained concurrently by Doron-Arad et al. (2026) in their online learning framework. It was also obtained by Hanneke et al. (2026) through a different stable-compression approach, but their result is directly PAC and does not give the corresponding online mistake-bound statement.

E.4. Improving Weak Provers as Sequential Harnessing

We now reinterpret the verifier-assisted prover construction of Section B in the language of sequential harnessing. The base learner is now a verifier rather than a predictor of the next reasoning step. The harness uses the verifier interactively in order to search for a correct reasoning trajectory generated by one or more weak provers.

Let X denote a space of problem instances and let Σ denote a space of reasoning steps. Fix a horizon L. Let $H \subseteq \{\text{YES}, \text{NO}\}^{X \times \Sigma^*}$ denote a class of prefix verifiers. Recall that a verifier $h(x, \tau_{1:\ell})$ predicts whether the last reasoning step τ_ℓ is valid assuming the preceding prefix is correct. Suppose we are also given a collection of provers $\mathcal{P} = \{P_1, \ldots, P_k\}$. Given a problem instance x and a correct prefix $\tau_{1:\ell}$, prover P_i generates a distribution over candidate next reasoning steps. The key assumption is that the prover collection is α-good, i.e., for every correct prefix, at least one prover produces a correct next step with probability at least α. The goal is now to construct a strong prover, given access to a "perfect" prefix verifier.

Base and system problems. The base problem is prefix verification $P_{\text{ver}} = (I_{\text{ver}}, Y_{\text{ver}}, H_{\text{ver}})$, where $I_{\text{ver}} = X \times \Sigma^{\leq L}$, $Y_{\text{ver}} = \{\text{YES}, \text{NO}\}$, and $H_{\text{ver}} = H$. The system problem is full proof generation $P_{\text{gen}} = (I_{\text{gen}}, Y_{\text{gen}}, H_{\text{gen}})$, where $I_{\text{gen}} = X$ and $Y_{\text{gen}} = \Sigma^L \cup \{\perp\}$, where $\perp$ denotes abstention or time-out. A system predictor takes a problem instance x and either outputs a full reasoning trajectory or abstains.

Harness construction. Let $\Gamma_{\text{ver}} = (I_{\text{ver}} \times Y_{\text{ver}})^*$. Fix a sampling budget B. Given a problem instance x and transcript γ, define the harness $S_{\text{boost}} : X \times \Gamma_{\text{ver}} \to I_{\text{ver}} \sqcup (\Sigma^L \cup \{\perp\})$ as follows. The harness maintains a current verified prefix $\tau_{1:\ell}$, initially empty. At stage $\ell + 1$, the harness repeatedly samples candidate next steps from the provers $\tau_{\ell+1} \sim P_i(x, \tau_{1:\ell})$ for various $i \in [k]$. For each sampled candidate step, the harness queries the verifier on the extended prefix $(x, \tau_{1:\ell+1})$. If the verifier accepts, the harness appends the step and proceeds to the next stage. If the verifier rejects, the harness continues sampling. If no acceptable next step is found after B attempts, the harness halts and outputs $\perp$. If the harness successfully constructs a full trajectory $\tau_{1:L}$, it outputs that trajectory.

Thus, the harness uses the verifier as a local consistency oracle to search through the space of prover-generated continuations.

Decoder construction. Given a correct reasoning trajectory $\tau = (\tau_1, \ldots, \tau_L)$, define the decoder $D_{\text{boost}} : X \times \Sigma^L \to \Gamma_{\text{ver}}$ by

$$D_{\text{boost}}(x, \tau) = \big(((x, \tau_1), \text{YES}), ((x, \tau_{1:2}), \text{YES}), \ldots, ((x, \tau_{1:L}), \text{YES})\big).$$

Thus, the decoder maps a successful proof trajectory to the sequence of verification interactions certifying that every prefix is correct.

Consistency. Suppose the target verifier is $h^\star \in H$. Whenever the harness queries a correct prefix, $h^\star$ accepts. Therefore, conditioned on eventually sampling a correct next step at every stage, the harness outputs a correct full proof. Moreover, the sequence of verification interactions generated by the harness on a successful trajectory is exactly $D_{\text{boost}}(x, \tau)$. Thus, the prover-boosting procedure can be viewed as a sequential harnessing process in which the verifier guides autoregressive proof construction.

Connection to Section B. This example is closely related to the verifier-assisted prover construction in Section B. However, there are several important conceptual differences.

First, the present formulation is an online-to-online reduction. The sequential harnessing theorem states that if the verifier learner makes few online mistakes, then the induced prover-generation procedure also makes few online trajectory-level mistakes. By contrast, Section B studies an online-to-batch conversion. There, the verifier is first learned online and then converted into a PAC-style generalization guarantee for the wrapped prover.

Second, the present harnessing formulation does not directly learn a verifier from prover interaction. Instead, it assumes access to a correct prefix verifier (oracle, or human expert) and uses it to improve generation through interaction with weak provers. In contrast, Section B explicitly studies how a verifier itself can be learned online from oracle feedback while interacting with adaptive provers.

Third, the sequential harnessing abstraction only tracks ordinary trajectory-level mistakes. It does not distinguish between soundness and completeness errors. The central contribution of Section B is precisely that these two error types matter differently for prover amplification. In particular, soundness errors correspond to accepting incorrect reasoning steps and therefore directly control the probability that the wrapped prover outputs an incorrect proof. Completeness errors instead lead primarily to abstention or rejection of correct candidate steps.

Thus, sequential harnessing captures the structural online reduction underlying verifier-guided generation, while Section B develops the much more refined analysis needed to reason about asymmetric verifier errors, distribution shift induced by prover-verifier interaction, and PAC-style generalization guarantees for the resulting wrapped prover.