Learning to Audit ML Models with Mental Models

Yao Rong[1,2]
yao.rong@l3s.de

Vaibhav Unhelkar[2,3]
vaibhav.unhelkar@rice.edu

[1]L3S Research Center, Leibniz University Hannover, Hannover, Germany
[2]Department of Computer Science, Rice University, Houston, TX, United States
[3]Ken Kennedy Institute, Rice University, Houston, TX, United States

Abstract

When auditing machine learning models, human experts sequentially inspect model predictions and explanations, gradually building a mental model of the model's behavior to uncover errors. This manual process is critical for safe deployment in high-stakes domains but fundamentally limited in scalability. In this work, we propose a novel method to simulate this human auditing behavior for automating the auditing process. We formalize auditing as a sequential decision-making problem, modeled as a Markov decision process. Our state representation summarizes the auditor's accumulated knowledge of model predictions and feature usage. Building on this formalization, we solve the MDP using reinforcement learning, yielding RLAuditor, which learns to select informative test samples to efficiently uncover model errors. We further show that our state can be interpreted as a simplified mental model of the audited system: under Gaussian assumptions, its update rule coincides with Bayesian belief revision, and it converges to the model's behavioral pattern at an interpretable rate. We validate our approach on multiple ML models and datasets across different data modalities. In a human subject study, we demonstrate that RLAuditor helps human auditors produce more accurate audit reports in less time.

1 Introduction

Auditing is a governance mechanism that evaluates machine learning (ML) models by analyzing their outcomes or simulating user interactions to identify, assess, and mitigate potential risks and harms [Mökander et al., 2023, Lam et al., 2024, Raji and Buolamwini, 2019]. Auditing ML models is crucial for ensuring their reliability and safety, particularly in high-stakes applications [Kelly et al., 2019, Zhang et al., 2020]. Regulatory frameworks such as the European AI Act mandate audits and testing to identify vulnerabilities, biases, and unintended behaviors in ML models. To formalize the concept of ML model auditing, Mökander et al. [2023] propose a three-layered framework consisting of governance audits, model audits, and application audits. While the first two types are designed to be conducted by model technology providers, i.e. the organizations developing ML models, the third focuses on downstream applications and must be conducted within domain-specific contexts.

In this paper, we focus on this third type, known as **application audits**. Adapting the taxonomy of Mökander et al. [2023] for general ML models, application audits can be defined as: "impact-oriented assessment of the risks posed by products and services built on top of pre-trained ML models." Application auditing *need to be conducted by end-users or application developers* who rely on end-user ML tools to build and evaluate customized applications by fine-tuning general-purpose ML models. Unlike large organizations, these users may have limited computational resources and may lack dedicated teams for model auditing. Moreover, *application audits must be conducted routinely* to address evolving requirements within specific domains.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

End-users currently rely on manual processes to conduct application audits. A typical application audit proceeds as follows: a human expert queries the model using an auditing dataset, observing its behavior, including *decisions and explanations* for the test samples. Alongside data samples, model explanations aid the audit process by revealing feature use and supporting the assessment of model behavior [Yadav et al., 2024, Balayn et al., 2022]; their usefulness is also validated in our human user study. The auditors identify erroneous samples and then iteratively select additional test samples from the dataset to further probe the model. The selection of test samples is guided by intuition to maximize the detection of problematic model behaviors. Identified behaviors are then categorized based on factors such as severity and error type, culminating in an audit report.[1] The entirely manual nature of the auditing process limits scalability, hinders the discovery of problematic model behaviors, and leads to inefficient use of domain experts' time. Therefore, more research is needed on **automated auditing methods** that complement existing manual approaches to better utilize domain experts' time and identify problematic model behaviors.

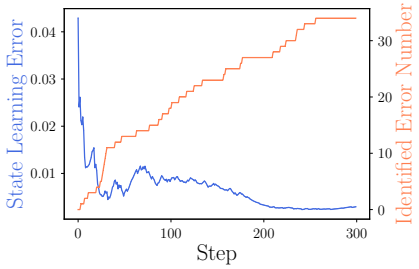


Figure 1: Application auditing is a procedure for acquiring accurate knowledge of the model (blue) to identify errors (orange).

Our key observation is that auditing is inherently sequential: auditors start by evaluating model predictions and explanations on a few exploratory instances to build an understanding of the model's behavior (compressed into a "state"). They then use this knowledge to strategically select subsequent instances, continuously balancing exploration (learning the model) and exploitation (using that knowledge to detect mistakes). As familiarity grows, auditors become more effective at identifying errors. Figure 1 illustrates the exploration-exploitation process during auditing illustrated by our proposed method. Concretely, we make the following contributions:

- We formulate this audit process as a **sequential decision-making problem**. We model the auditing process as a Markov decision process (MDP), where the state captures the auditor's accumulated knowledge about model predictions and feature usage, and actions correspond to selecting the next test sample.
- Building on this formalization, we propose **RLAuditor, a reinforcement learning (RL) based method for automating the auditing process.** RLAuditor learns to select informative test samples based on the current knowledge of the model encoded in the state.
- We show that RLAuditor's state operationalizes a human-auditor-inspired **mental model** [Johnson-Laird, 1983, Rouse and Morris, 1986] of the system under audit. Under Gaussian assumptions, our MDP state update rule coincides with Bayesian belief revision, and the state converges to the model's behavioral pattern at an interpretable rate.
- Through experiments on multiple ML models and datasets, including real-world applications in medical imaging and mathematical reasoning, we demonstrate the effectiveness of our approach. In a human subject study, we show that RLAuditor helps auditors produce more accurate reports in less time. [2]

2 Related Work

Research on auditing ML models has grown in response to the need for responsible AI deployment [Mökander et al., 2023, 2022]. Application audits are carried out routinely by end-users and developers with limited resources, who query a model and inspect its predictions and explanations sample by sample [Balayn et al., 2022, Wexler et al., 2019, Yadav et al., 2024, Lin and Jackson, 2023]. Existing methods can be grouped into two categories: facilitating human audits and automating the audit process. Model explanations play a key role in both, revealing feature use and supporting assessment of model behavior [Yadav et al., 2024, Balayn et al., 2022].

Facilitating Manual Audits. Tools in this category often provide application-specific interfaces for exploring model behaviors [Balayn et al., 2022, Wexler et al., 2019, Ren et al., 2016, Zhang et al., 2018]. For example, the What-If Tool [Wexler et al., 2019] is a model-agnostic interactive

[1] A practical instance of this manual auditing process is detailed in Section 6.

[2] The implementation of our model is at: `https://github.com/yaorong0921/RLAuditor`

visualization tool for analyzing ML models. Another approach leverages generative models to expand auditing datasets, with humans reviewing and curating the most relevant test cases [Ribeiro and Lundberg, 2022, Rastogi et al., 2023, van Breugel et al., 2024]. While useful, these approaches rely on human effort, limiting scalability.

Automated Audits. Our work aligns with automated methods that aim to detect undesirable behaviors without heavy human involvement [Kang et al., 2018, Ma et al., 2018, Singla et al., 2021, Lourenço et al., 2019]. Examples include assertion-based detection of inconsistencies in video object detection [Kang et al., 2018] and learned observation assertions (LOA), which learn a prior over plausible outputs to rank potentially erroneous labels in autonomous-vehicle data [Kang et al., 2022]. Existing approaches are largely domain-specific and produce a *one-shot, static ranking* of the audit pool given a fixed model. Our method is a *sequential, state-conditioned policy*: each query updates a compact summary of what has been observed, and the next query depends on it.

Role of Model Explanations. Explanations support audits in both manual and automated settings. User studies demonstrate their effectiveness in manual audits [Balayn et al., 2022, Yadav et al., 2024], while automated approaches such as XAudit [Yadav et al., 2024] use explanations to reconstruct models and ensure the inclusion of relevant features. However, XAudit is limited to simple models, such as linear regression. Like LOA, it scores the pool once and returns a fixed ranking. In contrast, our work utilizes explanations to audit more complex models, including CNNs for image classification. A more comprehensive review of related work is in Appendix A.

3 Formalization and Algorithmic Solution

This section first defines the efficient application auditing problem (Theorem 1) and then introduces RLAuditor that solves it with reinforcement learning.

3.1 Problem Statement

In the context of application auditing, auditing an ML model $\mathcal{M} : X \to Y$ is defined as the detection of its errors in the auditing dataset $\mathcal{D}$. There are two levels of errors:

1. **Level 1**: Incorrect model predictions in the auditing dataset. Mathematically, this translates to the detection of input instances x in the dataset $\mathcal{D}$ for which the model's prediction $\mathcal{M}(x) \neq y$ differ from the ground truth $y \in Y$.
2. **Level 2**: Potential feature misuse, such as prohibited features or invalid combinations. These may not always cause incorrect predictions (Level 1), but must still be detected due to their potential harm. Mathematically, given a feature set $\mathcal{F}$, we formulate this as an auditor comprehending the features $F'_y \subseteq \mathcal{F}$ used by the model to predict output y, and assessing whether they align with the auditor-intended features $F_y \subseteq \mathcal{F}$. We denote by $C \subset \mathcal{D}$ the set of instances exhibiting an error of either level.

Given the potentially vast input space X and, consequently, the auditing dataset $\mathcal{D}$, manually identifying every instance of unintended model behavior is often impractical. This leads to *Efficient Application Auditing Problem*, which aims to identify the maximum number model errors by observing limited examples.

Definition 1 (Efficient Application Auditing Problem)**.** *Efficient application auditing is defined as the problem of selecting K input instances, denoted by the set C', that maximally overlap with the set of inputs on which the model makes errors. Formally, this translates to selecting $C' \subseteq \mathcal{D}$ such that $|C'| = K$ and $|C' \cap C|$ is maximized.*

3.2 RLAuditor: Application Auditing using Reinforcement Learning

We now introduce **RLAuditor**, an agent for solving the *Efficient Application Auditing* problem. First, inspired by how humans audit ML models, we frame auditing as a *sequential decision-making problem* and model it as an MDP. Next, we solve the designed MDP using RL to obtain RLAuditor's auditing policy.

Auditing as a Markov Decision Process. Formally, consider the model $\mathcal{M}$ (accompanied by an explanation algorithm). For an input data point $x \in X$, it outputs logits $\hat{y} \in \mathbf{R}^N$, one per class,

whose largest entry gives the predicted class label $\mathcal{M}(x) \in Y$, and an explanation $e \in \mathcal{E}$, where e is a feature attribution vector. The sequential decision-making problem then translates to learning a policy π for selecting the $(m+1)$-th element x_{m+1} of the set C', given the model behavior on the elements selected thus far $(x_1, \ldots, x_m)$. We approximate the policy as a Markovian policy $\pi(\cdot \mid s)$, where s denotes a suitable Markovian statistic derived from the problem inputs and the previously selected elements $(x_1, \ldots, x_m)$ of C. Correspondingly, we frame the sequential decision-making process as an MDP. Learning effective policies in this setting hinges on the design of the state representation $s \in \mathcal{S}$, followed by the appropriate specification of the remaining elements of the MDP tuple $(\mathcal{S}, \mathcal{A}, T, R)$. A detailed explanation is provided in Appendix F.

State Space $s \in \mathcal{S}$. To ensure effective auditing, the state s should represent the auditor's knowledge of the model, based on its performance and explanations on previously selected test cases $(x_1, \ldots, x_m)$. At the same time, to enable tractable policy learning, the state s should be both compact and Markovian. Designing such a state is non-trivial. In this work, we explore several potential state features and evaluate their relative advantages through ablation studies. Based on the results of these studies (presented in Section 5.2), we adopt a state representation for RLAuditor that includes a summary of: (i) the model's prediction performance across classes, and (ii) the features used by the model, which can be reflected by feature attribution scores. Concretely, we selected $s = [\phi_{\text{class}}, \phi_{\text{explanation}}, n]$, where $\phi_{\text{class}} \in \mathbf{R}^{N \times N}$ is the matrix for class prediction information, N is the number of classes; and $\phi_{\text{explanation}} \in \mathbf{R}^{N \times d}$ represents the transform matrix from features and classes, and d is the dimension of the feature embeddings. The vector $n \in \mathbf{N}^N$ indicates the number of samples seen so far in each class. Concretely, for ϕ_{class} we use the prediction logits for each viewed sample as the knowledge representation for the classifier. Each row of the matrix $\phi_{\text{class}} \in \mathbf{R}^{N \times N}$ corresponds to a predicted class, and the entry at position (i, j), denoted as $\phi_{\text{class}}(i, j)$, represents the averaged logits of the classifier for class j among the samples predicted as class i. Formally, $\phi_{\text{class}}(i, j)$ is defined as:

$$\phi_{\text{class}}(i, j) = \frac{1}{|\mathcal{U}_i|} \sum_{x_k \in \mathcal{U}_i} \hat{y}_{k,j},$$

where $\mathcal{U}_i$ is the set of all *past* samples predicted as class i, and $\hat{y}_{k,j}$ is the logit output of the classifier for class j given the input sample x_k. Moreover, $|\mathcal{U}_i|$ equals n_i, indicating the number of viewed samples in class i. This representation captures the confidence distribution of the classifier across all classes for samples predicted to belong to a specific class. Similarly, we define the matrix $\phi_{\text{explanation}} \in \mathbf{R}^{N \times d}$, where each row corresponds to a specific predicted class, and each entry captures the average feature embeddings for that class. Specifically, the entry at position (i, l), denoted as $\phi_{\text{explanation}}(i, l)$, represents dimension l of the average feature embedding among all samples predicted as class i. Formally, we define:

$$\phi_{\text{explanation}}(i, l) = \frac{1}{|\mathcal{U}_i|} \sum_{x_k \in \mathcal{U}_i} \psi_l(x_k, e_k),$$

where $\psi_l(x_k, e_k)$ denotes dimension l of the feature embedding for the sample x_k given e_k. To construct $\psi(x_k, e_k)$, we utilize e_k as a mask over the input data and extract the corresponding feature embeddings. These embeddings are then averaged to construct $\phi_{\text{explanation}}$, allowing us to capture class-specific feature representations.

Action $a \in \mathcal{A}$. The action $a = x$ simply corresponds to selecting the next element $x = x_{m+1}$ of C'. Thus, the Markovian policy $\pi(a \mid s)$ aims to learn how to select the next element of C' based on the current information about class predictions and explanations encoded in s.

Transition Model $T(s'|s, a)$. The transition function defines how the state is updated based on the current state and the selected action. To compute the next state, we first query the model $\mathcal{M}$ on the most recently selected input $x = a$, obtaining the corresponding model logits $\hat{y}$ and explanation e. Next, given the tuple $[x, \hat{y}, e]$, we define the transition for each state feature in the updated state $s' = [\phi'_{\text{class}}, \phi'_{\text{explanation}}, n']$ via the following deterministic functions:

$$\begin{aligned}
\phi'_{\text{class}}(i) &= \frac{1}{n_i + 1} \left(n_i \cdot \phi_{\text{class}}(i) + \hat{y} \right), \\
\phi'_{\text{explanation}}(i) &= \frac{1}{n_i + 1} \left(n_i \cdot \phi_{\text{explanation}}(i) + \psi(x, e) \right), \\
n'_i &= n_i + 1,
\end{aligned} \tag{1}$$

for the predicted class $i = \arg\max_j \hat{y}_j$. Note that s' is computed from the current state and the single new observation only, and earlier samples are never stored.

Reward Model $R(s, a)$**.** To leverage RL, a well-designed reward function is essential. Following the guideline that the "reward signal is your way of communicating to the agent what you want achieved, not how you want it achieved" [Sutton, 2018], we design a reward function that encourages the algorithm to select inputs where the model exhibits errors. Intuitively, the reward function informs the auditor whether the selected action successfully leads to the discovery of model errors. Mathematically, the reward function assigns a score of 1 to selected model inputs that reveal errors, and a score of 0 to those that do not. During training, assessing whether the model makes an error requires human in the loop to judge model outputs and provide this reward signal. These annotations, or human inputs, are required only during the training phase. During deployment of RLAuditor, no annotations are needed, enabling a fully automated auditing process.

Auditing Policy π**.** Lastly, given the MDP model of the auditing process $(\mathcal{S}, \mathcal{A}, T, R)$, we employ a widely used RL algorithm, Deep Q-Network (DQN), to learn the auditing policy for sequentially selecting the elements of C' [Mnih et al., 2015]. To learn this policy, we provide an auditing dataset denoted as $\mathcal{D}_{\text{train}} \subseteq \mathcal{X}$. We evaluate its performance on a test set $\mathcal{D}_{\text{test}} \subseteq \mathcal{X}$. Further implementation details for the training paradigm of RLAuditor are provided in Appendix F.

4 Theoretical Analysis

Next, we formally connect RLAuditor to how human auditors build mental models of ML models.

4.1 Mental Model Construction in Human Auditing

When auditing an ML model, a human auditor constructs and refines an internal representation (*mental model*) of the audited ML model's behavior from observed inputs and outputs. Mental models are simplified, task-relevant representations of a system that people use to explain and predict its behavior [Johnson-Laird, 1983, Rouse and Morris, 1986]. For an auditor, the task-relevant questions are: "for each class, what does the model tend to output, and what features does it tend to look at?" Consistent with this account, auditors in our user study summarized model failures by inspecting explanations of misclassified samples (Section 6, F3). Formally, we model the auditor as maintaining a posterior belief over the model's behavioral pattern $\theta_{\mathcal{M}}$:

$$P(\theta_{\mathcal{M}} \mid x_1, \hat{y}_1, e_1, \ldots, x_m, \hat{y}_m, e_m).$$

Bayesian belief update Let $\mathcal{H}_m = \{(x_k, \hat{y}_k, e_k)\}_{k=1}^m$ denote the observation history up to step m. After each new observation, an idealized auditor updates their belief about $\theta_{\mathcal{M}}$ via Bayes' rule (derivation in Appendix B):

$$P(\theta_{\mathcal{M}} \mid \mathcal{H}_m) = \frac{P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m)\, P(\theta_{\mathcal{M}} \mid \mathcal{H}_{m-1})}{P(\hat{y}_m, e_m \mid x_m, \mathcal{H}_{m-1})}. \tag{2}$$

4.2 Operationalizing the Mental Model

The posterior in Eq. 2 is defined over a general behavioral pattern $\theta_{\mathcal{M}}$. To make this concrete and tractable, we must choose a specific operationalization. We propose $\theta_{\mathcal{M}}$ as the class-conditional mean behavior:

$$\theta_{\mathcal{M}} = \{(\mu_1, \nu_1), \ldots, (\mu_N, \nu_N)\},$$

where $\mu_i = \mathbb{E}[\hat{y} \mid \text{predicted class } i]$ is the mean logit vector and $\nu_i = \mathbb{E}[e \mid \text{predicted class } i]$ is the mean feature attribution for samples predicted as class i. We justify this choice on three grounds. First, it is task-relevant: μ_i encodes the model's *confusion pattern* for class i (if μ_i places high weight on class j, the model tends to confuse i with j), and ν_i encodes *which features the model relies on* when predicting class i. Together, these two vectors are used in solving the specific task. Second, it is Bayesian-coherent: the state update rule in RLAuditor corresponds to Bayesian belief revision under this operationalization (Proposition 3). Third, it yields convergence with an interpretable rate: under representative sampling, the state converges to $\theta_{\mathcal{M}}$ at a rate that depends on the least-observed class, mirroring the difficulty human auditors face with rarely seen classes (Theorem 4).

State Update as Bayesian Belief Revision We make the following distributional assumption:

Assumption 2. *For samples predicted as class i, the logit vectors are i.i.d. draws from $\hat{y} \sim \mathcal{N}(\mu_i, \Sigma)$, and feature attributions are i.i.d. draws from $e \sim \mathcal{N}(\nu_i, \Gamma)$, where Σ and Γ are shared across classes.*

The model's output is deterministic given an input. The randomness arises from the data distribution inducing a distribution over logits and feature attributions. The Gaussian assumption is a standard modeling choice: for each predicted class, the logits and attributions arise from many diverse inputs, and their distributions are approximately Gaussian. The same argument applies to the explanation's feature embeddings $\psi(x, e)$. For predicted class i, the state transition rule upon observing a new logit vector $\hat{y}$ is:

$$\phi'_{\text{class}}(i) = \frac{1}{n_i + 1}\big(n_i \cdot \phi_{\text{class}}(i) + \hat{y}\big). \tag{3}$$

Proposition 3 (Online Bayesian Update)**.** *Under Assumption 2, the state update in Eq. 3 is an instance of the general Bayesian update in Eq. 2. Specifically, with prior $\mu_i \sim \mathcal{N}(\phi_{class}(i), \Sigma/n_i)$ and likelihood $\hat{y} \sim \mathcal{N}(\mu_i, \Sigma)$, the posterior mean equals $\phi'_{class}(i)$, and the posterior precision increases from n_i to $n_i + 1$.*

The same result holds for $\phi_{\text{explanation}}$: with prior $\nu_i \sim \mathcal{N}(\phi_{\text{explanation}}(i), \Gamma/n_i)$ and likelihood $e \sim \mathcal{N}(\nu_i, \Gamma)$, the explanation state update is also exact Bayesian belief revision. The full derivation is given in Appendix C. This result can be interpreted as follows: early in the audit (n_i small), each new observation shifts the state substantially because the prior is weak. As more samples are observed (n_i large), the state becomes stable and new observations have diminishing effect. Proposition 3 thus characterizes an idealized auditor. Human belief updating can be more conservative and order-dependent than Bayes' rule [Nourani et al., 2021].

Convergence of the Mental Model We now characterize how quickly the components ϕ_{class} and $\phi_{\text{explanation}}$ of the state s_m converge to $\theta_{\mathcal{M}}$.

Theorem 4 (Convergence Rate)**.** *Under Assumption 2, the state s_m after m observations satisfies:*

$$\|\phi_{class} - \theta^{(\mu)}_{\mathcal{M}}\|_\infty \leq \sigma\sqrt{\frac{2\log(2N^2/\delta)}{m_{\min}}}$$

with probability at least $1 - \delta$, where $m_{\min} = \min_i n_i$ is the number of observations in the least-observed class, and $\theta^{(\mu)}_{\mathcal{M}} = (\mu_1, \ldots, \mu_N)$. The same bound holds for $\|\phi_{explanation} - \theta^{(\nu)}_{\mathcal{M}}\|_\infty$ with N^2 replaced by Nd where d is the feature attribution dimension.

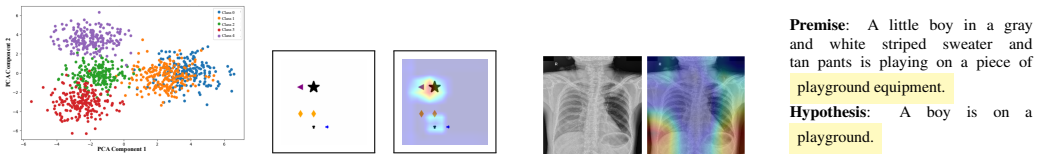
The proof, based on Hoeffding's inequality and a union bound, is given in Appendix D. Note that Theorem 4 is a property of the state *estimator*: the policy selects samples error-conditionally and thus does not in general satisfy Assumption 2 or inherit the guarantee, but random sampling during training partly restores the estimator and the measured state error stays close to the bound (Appendix E).

The convergence rate depends on $m_{\min}$, the least-observed class. This mirrors a familiar difficulty in human auditing: rare classes are harder to "understand" because fewer observations are available. Human auditors are similarly less confident about infrequently encountered failure modes. The bound formalizes the intuition: audit quality should degrade for underrepresented classes.

Remark 5 (Connection to mental models)**.** *Human auditors construct an internal representation of the model's behavioral pattern $\theta_{\mathcal{M}}$, i.e., a mental model [Johnson-Laird, 1983]. The state s is designed to capture the same task-relevant information that a human auditor's mental model would contain. Proposition 3 shows the update mechanism is Bayesian, and Theorem 4 shows the result converges to the target. The mental model is a property of the human process our method seeks to replicate, not a property we ascribe to the audited ML model.*

5 Experiments

We benchmark our RLAuditor on five datasets across diverse data modalities: two synthetic datasets `SynTab` and `SynImg`, the real-world image dataset VinDr-CXR [Nguyen et al., 2020], the textual dataset e-SNLI [Camburu et al., 2018], and the MATH dataset [Hendrycks et al., 2021] for auditing the mathematical reasoning of LLMs. Figure 2 lists example (input and explanation pair) of each dataset except for MATH. For the MATH dataset, the audited LLM's chain-of-thought reasoning serves as the explanation e. Then, we focus on the design choices of our algorithm, and the performance under limited supervision. Detailed experiment implementation can be found in Appendix G.



(a) (b) (c) (d)

Figure 2: Example of used datasets. (a) `SynTab`: PCA analysis of class features. (b) `SynImg`, (c) VinDr-CXR and (d) e-SNLI: Input x and its explanation e (highlighted in colors).

5.1 Comparison

We compare the overall performance of our proposed method with several baselines for both Level 1 and Level 2 tasks. The baselines used are as follows:

- Random: During testing, the exploration rate ϵ is set to 1, i.e., the next action is selected randomly.
- DNN: This baseline trains a deep neural network with the same architecture as the DQN, but in a supervised manner. It takes a single sample as input, represented as $[\hat{y}_k \mid \psi(x_k, e_k)]$, i.e., the logits of the classifier and the feature embedding extracted by applying its explanation as a mask over the input data (paralleling the state design s), and is trained with binary cross-entropy to predict whether that sample is an audit target (a Level 1 or Level 2 error). At test time, it scores every sample once and returns the top-K.
- AL (Active Learning): Active learning is used to select unlabeled samples for human review and labeling to improve model performance. In our comparison, we compare our method in selection with the query strategy Expected Gradient Length (EGL) in AL as given in [Settles, 2009].

To assess the performance, we measure the number of identified errors within a given K-step. K is set to 50, 500, 1000, 50, and 20 on the `SynTab`, `SynImg`, VinDr-CXR, e-SNLI, and MATH, respectively. Note that these Ks are smaller than the number of all errors existing in the test set, i.e., K is the upper bound for the errors detected in this experiment. We ensure that each sample is counted only once. For Level 2 errors, which we report them separately, we count only errors that exclusively belong to Level 2, ensuring no overlap with Level 1.

Figure 3 presents a comparison of our method with baselines on five datasets. Our RLAuditor outperforms the baselines on the four datasets and achieves competitive results on e-SNLI compared to AL. Notably, it surpasses AL in detecting Level 2 errors significantly, which are more complex. On the MATH dataset, RLAuditor achieves the best performance on combined Level 1 and Level 2 detection and a substantial gap on Level 2 errors (13.0 vs. 2.6 for the best baseline), demonstrating that RLAuditor also generalizes to auditing large language models. Our method consistently outperforms the supervised baseline, demonstrating the superiority of our RL-based approach. This also supports our argument that formulating the problem as a sequential decision-making task is more appropriate, and that RL is a suitable solution.

Comparison with prior automated auditing methods We further compare against the two existing automated auditors closest to our setting (introduced in Section 2): LOA [Kang et al., 2022], which ranks the pool by output atypicality, and XAudit [Yadav et al., 2024], which ranks samples by the model's reliance on known spurious features. We take the top-K of each ranking with the same K as in the previous experiments. XAudit applies only to linear models and is therefore evaluated on `SynTab` only. As Table 1 shows, RLAuditor finds the most errors overall on every dataset. LOA's

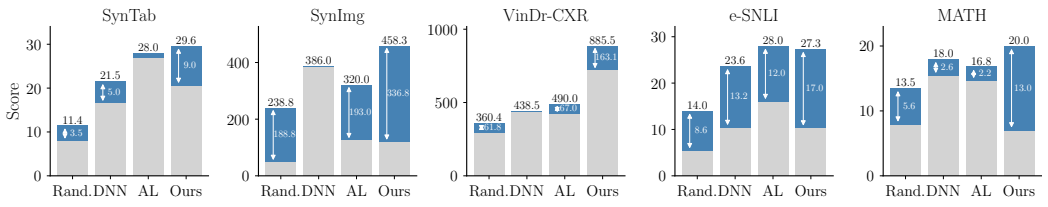


Figure 3: Auditing results for two levels of errors on five datasets, including the MATH dataset for auditing the mathematical reasoning of DeepSeek-R1. Bar ▬ represents the number of identified Level 2 errors. The total number of identified errors in both levels is listed above bars.

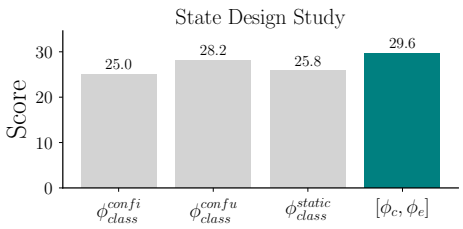


Figure 4: Comparison of different state designs. Light gray bars represent states without $\phi_{\text{explanation}}$.

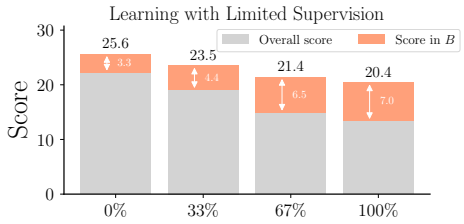


Figure 5: Comparison of scores when training with $p\%$ ($p \in \{0, 33, 67, 100\}$) of the labels for error type B.

atypicality score is only a proxy for error and is weakest at the top of its ranking, which is the region a small budget reaches (on MATH, 14.6 vs. 13.5 for random). XAudit finds slightly more Level 2 errors on `SynTab`, since it is told which features are spurious, but its selections concentrate on few classes. We examine this coverage of error types further in Appendix H.

5.2 State Design

In this section, we study the different state designs s in our RLAuditor algorithm. The intuition of the state is the information obtained from the viewed samples. Based on this information, an action a will be taken. Therefore, the representation of the information from past samples is essential.

We first study the effectiveness of the component ϕ_{class} and its design proposed in Section 3.2, which we refer to "prediction **confidence** scores" denoted as $\phi^{\text{confi}}_{\text{class}}(i, j)$, with the other two designs:

- Prediction **confusion** scores: ϕ_{class} has the same dimension as the previous design, but we use the format of confusion matrix of all previously seen samples as the knowledge representation for the classifier. Each row corresponds to the ground truth class, each column represents the predicted class, and the entry at position (i, c) represents the number of samples with ground truth class c that were predicted as class i. Formally, $\phi^{\text{confu}}_{\text{class}}(i, c)$ is defined as:

$$\phi^{\text{confu}}_{\text{class}}(i, c) = |\{x_k \in \mathcal{U} \mid y_k = c \text{ and } \mathcal{M}(x_k) = i\}|,$$

  where $\mathcal{U}$ is the set of all previously viewed samples, y_k is the ground truth label of sample x_k, and $\mathcal{M}(x_k)$ is the predicted label of sample x_k by the classifier.
- **Static** states: The two options above update s after viewing one sample, i.e., $\mathcal{U}$ is dynamic. In this setting, we use the *static* state representation, which is a fixed representation. Similar to the prediction confidence representation, $\phi^{\text{static}}_{\text{class}}(i, j)$ is defined as:

$$\phi^{\text{static}}_{\text{class}}(i, j) = \frac{1}{|\mathcal{D}_i|} \sum_{x_k \in \mathcal{D}_i} \hat{y}_{k,j},$$

  where $\mathcal{D}_i = \{x_k \in \mathcal{D}_{\text{test}} \mid \mathcal{M}(x_k) = i\}$ is the set of all test samples predicted as class i. Since $\mathcal{D}_{\text{test}}$ is fixed, the state remains static throughout the evaluation process.

Figure 4 shows the test scores using different state representations on `SynTab` in gray bars. Specifically, $\phi^{\text{confu}}_{\text{class}}$ achieves the best result. However, $\phi^{\text{confu}}_{\text{class}}$ requires the ground-truth class labels for x_k, which are not available in real-world scenarios. $\phi^{\text{static}}_{\text{class}}$ performs comparably to $\phi^{\text{confi}}_{\text{class}}$, but relies on access to the full test set, which is impractical during deployment as new data comes. Considering

Table 1: Comparison with prior automated auditing methods (mean $\pm$ std over 5 runs).

	LOA [Kang et al., 2022]		XAudit [Yadav et al., 2024]		RLAuditor (ours)	
	L1&2	L2	L1&2	L2	L1&2	L2
`SynTab`	15.6 $\pm$ 3.2	7.2 $\pm$ 2.9	15.6 $\pm$ 6.4	**11.2** $\pm$ 3.7	**29.6** $\pm$ 1.3	9.0 $\pm$ 0.8
`SynImg`	232.8 $\pm$ 2.5	198.0 $\pm$ 2.0	–	–	**458.3** $\pm$ 3.2	**336.8** $\pm$ 2.7
VinDr-CXR	729.6 $\pm$ 7.1	104.8 $\pm$ 5.0	–	–	**885.5** $\pm$ 7.2	**163.1** $\pm$ 6.8
e-SNLI	21.0 $\pm$ 2.2	15.0 $\pm$ 2.6	–	–	**27.3** $\pm$ 0.9	**17.0** $\pm$ 0.9
MATH	14.6 $\pm$ 1.9	2.8 $\pm$ 1.3	–	–	**20.0** $\pm$ 0.5	**13.0** $\pm$ 1.1

these practical constraints, we select $\phi_{\text{class}}^{\text{confi}}$ as the final design choice for our algorithm. We then study the effectiveness of $\phi_{\text{explanation}}$ in the state representation. Our final state $[\phi_{\text{class}}^{\text{confi}}, \phi_{\text{explanation}}]$ is denoted $[\phi_c, \phi_e]$. As shown in Figure 4, with the explanation information (ϕ_e), our agent learns to detect errors most effectively, highlighting the importance of using model explanations.

5.3 Learning with Limited Supervision

In this section, we aim to evaluate the performance of RLAuditor with limited training data for a specific error type (e.g., errors from a particular class, B). In this context, learning with limited supervision refers to identifying errors from B with no or partial access to the annotated errors in B. We highlight this setting, because it is labor-intensive in practice to collect human annotations for model errors. We use `SynTab` for evaluation. Class B contains 14 error samples of the 200. During training, we annotate $p\%$ of the class B (B is randomly selected) and leave the remainder unannotated, with $p \in \{0, 33, 67, 100\}$. The results are listed in Figure 5. We report the total number of discovered errors across all error types (all classes) in gray bars, and the scores for the class B in orange bars. We can see that with all the samples in B are labeled ($p = 100$) during training, the number of the identified error for B is the highest. However, leaving errors in B unlabeled results in the highest total scores. This is because the algorithm finds it easier to learn when there are fewer error types. Detection of B degrades as its annotations are removed, but RLAuditor still can generalize to the sparsely annotated error type. More experimental results can be found in Appendix H.

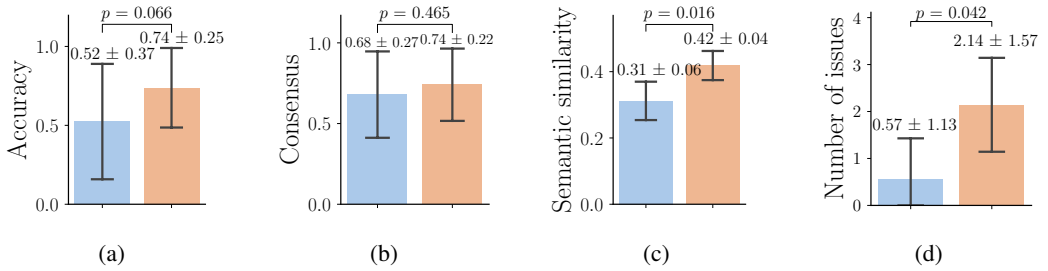
6 Evaluation with Human Auditors

One key challenge that RLAuditor aims to address is the scalability of practical audits. Effective auditing requires human-AI collaboration, as humans cannot scale alone and AI auditors need human judgment for accurate error labeling and learning domain knowledge. Thus, to inform future human-AI collaborative approaches to application auditing, we also conducted a human subject study to examine two research questions: **R1.** What strategies do human auditors use? **R2.** Can the samples selected by RLAuditor help humans better understand the model's behavior and errors?

Specifically, we implemented an auditing user interface (UI) that enables the study of interactions between human auditors and the RLAuditor. The dataset used for this task is a user intent dataset "Snips" [Coucke et al., 2018], which contains queries (short sentences) categorized into 7 intents. We finetuned a small pre-trained BERT variant model [Bhargava et al., 2021] to classify user intents, and this was the model to be audited. We adopted a between-subject design in which one group of participants was exposed to the RLAuditor-suggested samples (experimental group), while the other group received no suggested samples and used the same UI (control group). $N = 12$ participants were recruited through institutional email lists and have experience in evaluating and debugging ML models. Both groups include PhD students in computer science, making them well-qualified for the ML audit task. The participants were randomly and evenly assigned to two groups, and compensated $10 for their participation. Participants were tasked with summarizing model errors in an audit report, which included a set of question targeting Level 1 and Level 2 model errors. More details of the human user study can be found in Appendix I.

F1: RLAuditor helps humans generate more accurate audit reports. To assess the quality of the reports, we studied how participants identified different types of model errors. Participants in the experimental group were more likely to summarize misclassifications (Level 1 errors) and identify issues related to the model's feature usage (Level 2 errors). For each class, we asked participants to specify which other class(es) the model most frequently confused it with. Figure 6a presents the accuracy of misclassification identification (Level 1 errors), based on the ground-truth confusion matrix. The experimental group demonstrates higher accuracy. Figure 6b shows the pairwise Jaccard similarity within each group for the misclassification identification. The high consensus scores in both groups indicate strong agreement among participants on the identified errors. This suggests that the UI design influences the behavior of human auditors.

Regarding participants' analyses of model features, we observe that participants reported more similarly when using RLAuditor, as reflected in the report semantic similarity scores shown in Figure 6c. Participants summarized their observations on which features used by the model have issues or need improvement. With the help of RLAuditor, the experimental group demonstrated deeper reasoning and identified more issues (averagely two problems per class per user) related to the model's feature usage (Level 2 errors), as shown in Figure 6d.



(a) (b) (c) (d)

Figure 6: Human user study analysis among two groups: Control-; Experimental group. (a) Accuracy in identifying misclassifications. (b) Consensus in identifying misclassifications. (c) Participants' report similarity. (d) Number of issues identified in the model report.

F2: RLAuditor helps humans complete audits in less time. Participants were instructed to complete the task within 30 minutes but were allowed additional time if needed. We recorded the time each participant took to complete the task and analyzed the content of their reports. The average completion time for the control group was $37m55s \pm 6m45s$, while the experimental group completed the task in $32m26s \pm 4m38s$. The reduced time required by the experimental group highlights the effectiveness of the selected samples, as participants spent less time searching through the dataset. Notably, the efficiency gains are expected to become more as the dataset size increases. We also asked participants to rate how helpful they found the agent's suggestions in this audit task. The participants in the experimental group gave an average score of 5.75 ± 0.5 on a 7-point Likert scale, where 1 indicates "extremely useless" and 7 indicates "extremely helpful".

F3: Human auditors rely on model explanations for audits. In the participants' reported strategies, a common pattern emerged from both groups: 75% percent of participants reported relying on model explanations (highlighted tokens) and summarizing insights by analyzing misclassification explanations. 17% indicated they studied both correct and incorrect classifications in their audits. This aligns with prior work that addresses the usefulness of explanations in audit tasks [Yadav et al., 2024, Balayn et al., 2022]. It also validates the design of our RLAuditor, which incorporates model explanations to reflect human auditing strategies.

Based on findings **F1** and **F2**, we address research question **R2**: RLAuditor can assist humans in audits. The improvement comes from the agent clearly grouping problematic samples and presenting them. This allows humans to compare examples and better understand incorrect predictions or misused features. **F3** addresses **R1**, and it verifies that model explanations play an important role in application audits. More detailed analysis of the human user study can be found in Appendix I.

7 Conclusion

As ML models are deployed in real-world and safety-critical domains, auditing is essential to ensure their safe and responsible operation. In this work, we focus on application audits, which are important for assessing ML models but require domain-specific knowledge from diverse stakeholders and currently lack scalability. We formalize application audits as a sequential decision-making problem and propose RLAuditor to solve it, theoretically analyzing how our approach connects to human audits through mental models. Experiments across various datasets demonstrate the effectiveness of our formalization and RL-based solution. In a collaborative audit setting with human auditors, RLAuditor shows helpful in revealing model behaviors and uncovering issues. This work provides a new way to address the auditing problem and to support human auditors in human–AI collaboration for ML application audits.

Limitations and Future Work Our experiments show that RLAuditor successfully automates parts of application auditing for both traditional ML models and LLMs. However, scaling to larger models and more complex reasoning tasks will require further investigation into the design of expressive yet tractable state representations. RLAuditor also degrades on error types absent from the training annotations, it cannot guarantee coverage of classes it never samples, and loses its advantage when the audit budget is large or errors are not explanation-linked. A graded reward and online updates from human annotation trajectories are natural extensions. Our formalization should be viewed as expanding the toolkit for ML auditing and providing a foundation to future research in this area.

Acknowledgments and Disclosure of Funding

This work was largely performed while Yao Rong was a postdoctoral associate at Rice University's Human-AI Teaming Lab, supported by the Rice Academy of Fellows. The authors thank the anonymous reviewers and the members of the Ken Kennedy Institute for their constructive feedback. The authors declare no competing interests.

References

David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Agathe Balayn, Natasa Rikalo, Christoph Lofi, Jie Yang, and Alessandro Bozzon. How can explainability methods be used to support bug identification in computer vision models? In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2022.

Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in nli: Ways (not) to go beyond simple heuristics, 2021.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://aclanthology.org/D15-1075.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9539–9549. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8163-e-snli-natural-language-inference-with-natural-language-explanations.pdf.

Arjun Chandrasekaran, Viraj Prabhu, Deshraj Yadav, Prithvijit Chattopadhyay, and Devi Parikh. Do explanations make vqa models more predictable to a human? In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1036–1042, 2018.

Michael Chromik, Malin Eiband, Felicitas Buchner, Adrian Krüger, and Andreas Butz. I think i get your point, ai! the illusion of explanatory depth in explainable ai. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, pages 307–317, 2021.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*, 2018.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Engin Durmaz and M Borahan Tümer. Intelligent software debugging: A reinforcement learning approach for detecting the shortest crashing scenarios. *Expert Systems with Applications*, 198: 116722, 2022.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *Advances in Neural Information Processing Systems*, 34:21163–21174, 2021.

Disi Ji, Robert L Logan, Padhraic Smyth, and Mark Steyvers. Active Bayesian assessment of black-box classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7935–7944, 2021.

Philip N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press, Cambridge, MA, 1983.

Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for debugging machine learning. In *NeurIPS MLSys Workshop*, volume 3, 2018.

Daniel Kang, Nikos Arechiga, Sudeep Pillai, Peter D Bailis, and Matei Zaharia. Finding label and model errors in perception data with learned observation assertions. In *Proceedings of the 2022 international conference on management of data*, pages 496–505, 2022.

Gopinath Kathiresan. Adaptive test optimization: Using reinforcement learning to improve software testing strategies. *Well Testing Journal*, 33(S2):715–732, 2024.

Christopher J Kelly, Alan Karthikesalingam, Mustafa Suleyman, Greg Corrado, and Dominic King. Key challenges for delivering clinical impact with artificial intelligence. *BMC medicine*, 17:1–9, 2019.

Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. Adaptive rest api testing with reinforcement learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 446–458. IEEE, 2023.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. In *International Conference on Machine Learning*, pages 5753–5763. PMLR, 2021.

Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Khoa Lam, Benjamin Lange, Borhane Blili-Hamelin, Jovana Davidovic, Shea Brown, and Ali Hasan. A framework for assurance audits of algorithmic systems. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 1078–1092, 2024.

Cindy Kaiying Lin and Steven J. Jackson. From bias to repair: Error as a site of collaboration and negotiation in applied data science work. *Proc. ACM Hum.-Comput. Interact.*, April 2023. doi: 10.1145/3579607.

Raoni Lourenço, Juliana Freire, and Dennis Shasha. Debugging machine learning pipelines. In *Proceedings of the 3rd International workshop on data management for end-to-end machine learning*, pages 1–10, 2019.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. Mode: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 175–186, 2018.

Shira Mitchell, Eric Potash, Solon Barocas, Alexander D'Amour, and Kristian Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual review of statistics and its application*, 8 (1):141–163, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Jakob Mökander, Maria Axente, Federico Casolari, and Luciano Floridi. Conformity assessments and post-market monitoring: a guide to the role of auditing in the proposed european ai regulation. *Minds and Machines*, 32(2):241–268, 2022.

Jakob Mökander, Jonas Schuett, Hannah Rose Kirk, and Luciano Floridi. Auditing large language models: a three-layered approach. *AI and Ethics*, pages 1–31, 2023.

Duc Nguyen, DungNB, Ha Q. Nguyen, Julia Elliott, NguyenThanhNhan, and Phil Culliton. Vinbigdata chest x-ray abnormalities detection. `https://kaggle.com/competitions/vinbigdata-chest-xray-abnormalities-detection`, 2020. Kaggle.

Mahsan Nourani, Chiradeep Roy, Jeremy E Block, Donald R Honeycutt, Tahrima Rahman, Eric Ragan, and Vibhav Gogate. Anchoring bias affects mental model formation and user reliance in explainable AI systems. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, pages 340–350, 2021.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33 (3):291–297, 1997.

Inioluwa Deborah Raji and Joy Buolamwini. Actionable auditing: Investigating the impact of publicly naming biased performance results of commercial ai products. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 429–435, 2019.

Charvi Rastogi, Marco Tulio Ribeiro, Nicholas King, Harsha Nori, and Saleema Amershi. Supporting human-ai collaboration in auditing LLMs with LLMs. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pages 913–926, 2023.

Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE transactions on visualization and computer graphics*, 23(1):61–70, 2016.

Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of nlp models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, 2022.

Yao Rong, Tobias Leemann, Thai-Trang Nguyen, Lisa Fiedler, Peizhu Qian, Vaibhav Unhelkar, Tina Seidel, Gjergji Kasneci, and Enkelejda Kasneci. Towards human-centered explainable ai: A survey of user studies for model explanations. *IEEE transactions on pattern analysis and machine intelligence*, 46(4):2104–2122, 2023.

William B Rouse and Nancy M Morris. On looking into the black box: Prospects and limits in the search for mental models. *Psychological Bulletin*, 100(3):349–363, 1986.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

Burr Settles. Active learning literature survey. 2009.

Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. *Advances in neural information processing systems*, 20, 2007.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, and Eric Horvitz. Understanding failures of deep networks via robust feature extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12853–12862, 2021.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Wannita Takerngsaksiri, Rujikorn Charakorn, Chakkrit Tantithamthavorn, and Yuan-Fang Li. Pytester: Deep reinforcement learning for text-to-testcase generation. *Journal of Systems and Software*, 224: 112381, 2025.

Boris van Breugel, Nabeel Seedat, Fergus Imrie, and Mihaela van der Schaar. Can you rely on your model evaluation? improving model evaluation with synthetic test data. *Advances in Neural Information Processing Systems*, 36, 2024.

Yunlong Wang, Priyadarshini Venkatesh, and Brian Y Lim. Interpretable directed diversity: Leveraging model explanations for iterative crowd ideation. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–28, 2022.

James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.

Chhavi Yadav, Michal Moshkovitz, and Kamalika Chaudhuri. Xaudit: A learning-theoretic look at auditing with explanations. *Transactions on Machine Learning Research*, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Chih-Kuan Yeh, Been Kim, Sercan Arik, Chun-Liang Li, Tomas Pfister, and Pradeep Ravikumar. On completeness-aware concept-based explanations in deep neural networks. *Advances in neural information processing systems*, 33:20554–20565, 2020.

Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018.

Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2020.

A Extended Related Work

A.1 Methods for Auditing ML Models

This section expands on Section 2. Model audits are important not only for uncovering and fixing bugs in ML applications, but also for revealing hidden structures of collaboration and advocating for a reshaping of existing practices in ML science [Lin and Jackson, 2023].

Facilitating Manual Audits of ML Models. Besides interfaces, another strategy involves assisting humans in generating test cases by expanding the auditing dataset through generative models [Ribeiro and Lundberg, 2022, Rastogi et al., 2023, van Breugel et al., 2024]. For example, Ribeiro and Lundberg [2022] propose a testing loop where a large language model (LLM) generates test cases, and human auditors review the top failing ones, adding valid tests to relevant topics or sub-topics to construct a comprehensive test dataset.

Automated Audits of ML Models. Examples of such methods include algorithms and frameworks for identifying problematic model behaviors in specific tasks, such as improving model accuracy through selective input sampling [Ma et al., 2018], rather than directly assisting human auditors. Kang et al. [2018] propose using a set of assertion rules to detect errors in machine learning outputs for object detection in videos. For example, a car should not disappear and then reappear in consecutive video frames. These assertions can be “soft" and represented as probabilistic. This probabilistic modeling approach is further extended to identify errors in human-provided labels and machine learning model outputs within the autonomous vehicle domain [Kang et al., 2022], where ensuring the accuracy of annotations is critical for safety and performance. Kang et al. [2022] propose a system that learns priors to distinguish likely from unlikely values. The resulting probabilistic model is then used to rank data point labels based on their likelihood of being erroneous. Unlike sequential test selection for performance estimation [Kossen et al., 2021, Ji et al., 2021] or bandit-based error discovery [Lakkaraju et al., 2017], RLAuditor learns its acquisition policy and conditions it on model explanations.

Role of Model Explanations. Through a user study, Balayn et al. [2022] demonstrate the utility of diverse explanations in facilitating manual audits of ML models. In this work, we select explanation techniques (SHAP and GradCAM) that are well-established and widely validated in the literature to ensure a reasonable level of supporting human-AI collaboration [Chandrasekaran et al., 2018, Chromik et al., 2021, Wang et al., 2022]; see Table 3 in [Rong et al., 2023].

A.2 Layered Audit Frameworks

Prior layered model audit frameworks such as Lam et al. [2024], Mökander et al. [2023] focus on high-level, organization-wide audits, often assuming the resources of large institutions and aligning with regulations such as NYC’s Local Law 144 [Lam et al., 2024]. In contrast, there has been comparatively less focus on application-level audits. However, these are increasingly important, as many ML model practitioners lack the capacity for large-scale, organization-level audits. They need practical, lightweight tools to conduct audits routinely within their specific applications.

Another line of audit frameworks focuses on defining metrics and evaluations that capture key aspects of fairness and interpretability. For example, Mitchell et al. [2021] define fairness using equal decision measures or causal reasoning, while [Doshi-Velez and Kim, 2017, Alvarez-Melis and Jaakkola, 2018] introduce metrics such as robustness of interpretability to assess model explanations. However, some measures, such as human-grounded evaluation of explanations, require manual effort and cannot be fully captured by automated metrics [Doshi-Velez and Kim, 2017]. These frameworks emphasize post hoc evaluations at the dataset or model level. For instance, Mitchell et al. [2021] evaluate social bias across the dataset, Alvarez-Melis and Jaakkola [2018] assess the robustness of all generated explanations and Doshi-Velez and Kim [2017] emphasize human understanding of a model by viewing some explanations. They fail to provide individual samples that human auditors may want to examine more closely.

However, our focus is on identifying error samples and presenting them to users to support the auditing process. This procedure is not mathematically modeled in prior works. The definition of

"error" in our framework is general and can be determined by users or based on existing fairness or interpretability criteria.

A.3 Reinforcement Learning (RL)

RL is a general paradigm for solving sequential decision-making problems, including those involving uncertainty, through trial and error. In its most general setup, an agent learns to make optimal decisions by interacting with its environment, receiving rewards for its actions [Sutton, 2018]. Through the history of trial and error, it refines a policy to maximize cumulative rewards. RL has been widely used in complex domains such as robotics, game playing, autonomous driving, and finance, where explicit programming of optimal behavior is impractical [Silver et al., 2016, Arulkumaran et al., 2017]. In this work, we use RL to select the optimal next action (test case) for audits, while the development of new RL algorithms is not the main technical focus.

Test Case Generation for Software using RL. Prior work [Takerngsaksiri et al., 2025, Durmaz and Tümer, 2022, Kim et al., 2023, Kathiresan, 2024] has used RL to generate software test cases, which are modeled as sequences of actions such as input tokens, API calls, or UI interactions. These input sequences are designed to uncover bugs or test for malicious behavior in the system. For example, Takerngsaksiri et al. [2025] use deep RL within large language models to generate executable, high-coverage test code, while Durmaz and Tümer [2022] employ RL to generate minimal crashing input sequences. In these works, states represent the current program or input context, actions correspond to next steps, and rewards are based on outcomes such as code coverage or syntax validity.

In contrast, our work focuses on auditing ML models. The sequence of actions generated by our algorithm is not to trigger a system failure, but to construct a comprehensive understanding of the model's erroneous behavior: Unlike software debugging, where the goal is often to generate a failing test case, our method aims to select a representative set of test samples that reveal different facets of the model's failures to human auditors. Our state formulation simulates the accumulation of knowledge about model behavior, reflecting how humans build understanding through observations. This is a more challenging and cognitively aligned task.

B Derivation of the Bayesian Belief Update

We derive the auditor's belief update rule in Eq. 2 from the definition of conditional probability. Since $\mathcal{H}_m = \mathcal{H}_{m-1} \cup \{(x_m, \hat{y}_m, e_m)\}$, conditioning on $\mathcal{H}_m$ is equivalent to conditioning on both the history $\mathcal{H}_{m-1}$ and the new observation. Applying the definition of conditional probability:

$$P(\theta_{\mathcal{M}} \mid \hat{y}_m, e_m, x_m, \mathcal{H}_{m-1}) = \frac{P(\theta_{\mathcal{M}},\, \hat{y}_m, e_m \mid x_m, \mathcal{H}_{m-1})}{P(\hat{y}_m, e_m \mid x_m, \mathcal{H}_{m-1})}, \tag{4}$$

where we condition on x_m and $\mathcal{H}_{m-1}$ throughout, as these are known before the new output is observed. Factoring the joint in the numerator via the chain rule:

$$P(\theta_{\mathcal{M}},\, \hat{y}_m, e_m \mid x_m, \mathcal{H}_{m-1}) = P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m, \mathcal{H}_{m-1}) \cdot P(\theta_{\mathcal{M}} \mid x_m, \mathcal{H}_{m-1}). \tag{5}$$

We make two simplifying observations:

1. $P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m, \mathcal{H}_{m-1}) = P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m)$: given the behavioral pattern $\theta_{\mathcal{M}}$ and input x_m, the past history adds no information about the current output.
2. $P(\theta_{\mathcal{M}} \mid x_m, \mathcal{H}_{m-1}) = P(\theta_{\mathcal{M}} \mid \mathcal{H}_{m-1})$: the choice of the next input does not change our belief about $\theta_{\mathcal{M}}$ before seeing its output.

Substituting back into Eq. 4 yields:

$$P(\theta_{\mathcal{M}} \mid \mathcal{H}_m) = \frac{P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m)\, P(\theta_{\mathcal{M}} \mid \mathcal{H}_{m-1})}{P(\hat{y}_m, e_m \mid x_m, \mathcal{H}_{m-1})},$$

which is the belief update rule in Eq. 2.

C Proof of Proposition 3: Online Bayesian Update

We show that the state update in Eq. 3 is an instance of the general Bayesian update in Eq. 2 under Gaussian assumptions. Under the operationalization in Section 4, the posterior $P(\theta_{\mathcal{M}} \mid \mathcal{H}_m)$ decomposes into independent updates per class. For predicted class i:

Prior. The prior $P(\theta_{\mathcal{M}} \mid \mathcal{H}_{m-1})$ from Eq. 2, restricted to class i, becomes:

$$\mu_i \sim \mathcal{N}\left(\phi_{\text{class}}(i),\ \frac{\Sigma}{n_i}\right).$$

The prior mean $\phi_{\text{class}}(i)$ is the current state (the best estimate so far). The prior covariance Σ/n_i reflects that after n_i observations, uncertainty has been reduced by a factor of n_i.

Likelihood. The likelihood $P(\hat{y}_m, e_m \mid \theta_{\mathcal{M}}, x_m)$ from Eq. 2, for a sample predicted as class i, becomes:

$$\hat{y} \sim \mathcal{N}(\mu_i,\ \Sigma).$$

Given the true mean μ_i, each observation has covariance Σ.

Posterior. The posterior $P(\theta_{\mathcal{M}} \mid \mathcal{H}_m)$ from Eq. 2, restricted to class i, is obtained by multiplying the prior and likelihood. Since both are Gaussian, the posterior is also Gaussian (conjugacy). It can be interpreted as a weighted average: the prior contributes n_i observations worth of evidence pointing at $\phi_{\text{class}}(i)$, and the new data contributes one observation pointing at $\hat{y}$. Balancing by total evidence:

$$\mu_i \mid \hat{y} \sim \mathcal{N}\left(\frac{n_i \cdot \phi_{\text{class}}(i) + \hat{y}}{n_i + 1},\ \frac{\Sigma}{n_i + 1}\right).$$

The posterior mean is:

$$\frac{n_i \cdot \phi_{\text{class}}(i) + \hat{y}}{n_i + 1} = \phi'_{\text{class}}(i),$$

which is exactly the state update in Eq. 3. The posterior precision (inverse covariance) increases from n_i to $n_i + 1$, reflecting growing confidence with each observation. □

D Proof of Theorem 4: Convergence Rate

Under Assumption 2, for each predicted class i and logit dimension j, the entry $\phi_{\text{class}}(i, j)$ is the sample mean of n_i independent Gaussian random variables with mean $\mu_i(j)$ and variance σ^2 (where σ^2 is the corresponding diagonal entry of Σ). Since Gaussian variables are sub-Gaussian, Hoeffding's inequality applies:

$$P(|\phi_{\text{class}}(i, j) - \mu_i(j)| > t) \leq 2\exp\left(-\frac{n_i\, t^2}{2\sigma^2}\right).$$

Setting $t = \sigma\sqrt{2\log(2N^2/\delta)/m_{\min}}$ and applying a union bound over all $N \times N$ entries of the logit matrix:

$$P\left(\|\phi_{\text{class}} - \theta^{(\mu)}_{\mathcal{M}}\|_\infty > \sigma\sqrt{\frac{2\log(2N^2/\delta)}{m_{\min}}}\right) \leq N^2 \cdot 2\exp\left(-\frac{m_{\min} \cdot t^2}{2\sigma^2}\right) = \delta.$$

The argument for $\phi_{\text{explanation}}$ is identical, with the union bound taken over $N \times d$ entries. □

E State Estimation Error under the Learned Policy

To quantify the gap between Assumption 2 and the policy's error-seeking selection, we measure the state error $\|\phi_{\text{class}} - \theta^{(\mu)}_{\mathcal{M}}\|$ on `SynTab` after $K = 50$ queries, mixing a fraction of uniformly random queries into the greedy policy (Table 2). A fraction of 0.25 approximates the proportion of random steps taken during training. The bound of Theorem 4 evaluates to 0.080 at $\delta = 0.05$, $N = 5$, and $m_{\min} = K/N = 10$, the number of observations the least-observed class would receive if the budget were spread evenly.

Table 2: State error and class coverage on `SynTab` as a function of the fraction of uniform queries.

Uniform fraction	State error	Classes observed (of 5)
0.00 (greedy policy)	0.099	3.80
0.25 (training exploration)	0.071	4.91
1.00 (uniform)	0.040	5.00

F Implementation Details for RLAuditor

F.1 RLAuditor: Formalizing Efficient Auditing as an MDP

We introduce **RLAuditor**: an automated approach to solving the *Efficient Application Auditing* problem. Typically, auditors begin by evaluating model predictions and explanations on a few exploratory instances to develop an understanding of the model's performance and reasoning. They then exploit this understanding to more strategically select subsequent instances where model behavior is likely to be informative. As their familiarity with the model grows, they become increasingly effective at identifying its errors. Throughout this process, auditors must continuously balance exploration (i.e., learning more about the model's reasoning) and exploitation (i.e., using that knowledge to efficiently detect model errors). While this manual process may not scale well, it provides key insights for designing automated auditors. First, the sequential nature of selecting instances for model evaluation motivates formulating the auditing task as a *sequential decision-making problem*. Specifically, we model this process as a Markov Decision Process (MDP), which naturally captures the dynamics between the agent (the auditor) and the environment (the model's behavior), as well as the dependence on past observations. Second, the inherent *exploration-exploitation trade-off* in auditing motivates the use of *reinforcement learning* to address this challenge. Given an MDP formulation of the auditing process, we then learn a *policy for sequentially selecting the elements of set C'* (see Definition 1) using RL. Figure 7 depicts the process of learning this policy, using RL and human feedback. With model explanations, the utilization of features can be automatically assessed, further enhancing the auditing process. When deploying RLAuditor, the gray elements are excluded, reflecting the automated auditing process.

In practice, the expert is familiar with both the application and the model being audited, enabling them to identify errors. When working with RLAuditor, the expert's role goes beyond simple labeling. They provide feedback on whether selected samples are erroneous and can offer more advanced guidance, such as highlighting errors that RLAuditor may have overlooked. This feedback can be incorporated as a reward signal to refine and improve RLAuditor's performance. Moreover, in practice, the expert is responsible for composing comprehensive reports that summarize the different types of errors identified in the model.

F.2 Implementation Details for Figure 1

Figure 1 illustrates the intuition behind modeling the auditing process as a sequential task, where knowledge about the model accumulates over time and the selection of subsequent samples is guided toward more informative errors. To validate this intuition in the implemented algorithm RLAuditor, we plot the training curve of RLAuditor on `SynTab`. The state learning error is measured using the mean squared error:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N * N} \sum_{i,j} \left(\phi_{\text{class}}^{\text{confi}}(i,j) - \phi_{\text{class}}^{\text{confu}}(i,j) \right)^2,$$

where $\phi_{\text{class}}^{\text{confi}}$ represents the learned state, and $\phi_{\text{class}}^{\text{confu}}$ denotes the ground-truth state, which requires the ground-truth labels (defined in Section 5.2). Note that it is impracticable to obtain the ground-truth state for $\phi_{\text{explanation}}$, so we omit its error estimation in s for this plot. We also plot the reward curve over the training steps. The figure shows a clear trend: the estimation of s improves over time, and the reward increases as the model knowledge aids error discovery. Both curves saturate after about 300 steps. This figure also shows that the selection of error samples is a sequential decision-making process that relies on s.

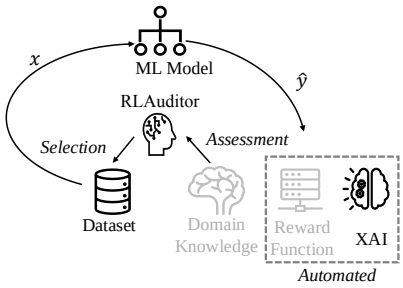


Figure 7: Auditing with RLAuditor.

F.3 Training Algorithm for RLAuditor

Algorithm for RLAuditor can be found in Algorithm 1. Our RLAuditor is adopted from Deep Q-Learning (DQN) [Mnih et al., 2015]. DQN combines Q-learning with deep neural networks (DNNs) to enable agents to learn value-based policies directly from high-dimensional input. The algorithm trains an agent through techniques such as experience replay and a target network. To stabilize training, Algorithm 1 uses a separate target network for value estimation.

Algorithm 1 RLAuditor Training Algorithm

Initialize replay buffer (memory) $\mathcal{B}$
Initialize Q with weights θ
Initialize target $\hat{Q}$ with weights $\hat{\theta} = \theta$
Set the number of episodes E
Set the maximum number of steps per episode H
for episode $= 1, 2, \ldots, E$ **do**
  $k \leftarrow 1$
  Select a random initial state s_1
  while Goal state not reached **and** $k \leq H$ **do**
    With probability ϵ select a random action a_k, otherwise select $a_k = \arg\max_a Q(s_k, a; \theta)$.
    Execute action a_k and observe reward r_k.
    Set $s_{k+1} = T(s_k, a_k)$, and store transition (s_k, a_k, r_k, s_{k+1}) in $\mathcal{B}$.
    Sample a random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{B}$.
    Set $z_j = r_j$ if the episode terminates at step $j+1$, otherwise $z_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \hat{\theta})$.
    Perform a gradient descent step on $(z_j - Q(s_j, a_j; \theta))^2$ with respect to θ.
    Every 1000 steps, update $\hat{Q} = Q$
    $k \leftarrow k + 1$
  end while
end for

F.4 Implementation Details

Q **Architecture.** The Q-network integrates state and action representations to estimate Q-values. It consists of two sub-networks. One maps input states to a 128-dimensional latent space using fully connected layers with ReLU activations. Actions are processed through another network with a linear layer, and ReLU, producing 64-dimensional action features. The model expands state features to match the number of actions, concatenates them with action features, and passes them through an estimation network comprising multiple fully connected layers and ReLU activations. The final output layer predicts a single Q-value per state-action pair. This architecture enables effective learning in environments with structured action spaces, making it suitable for complex decision-making tasks.

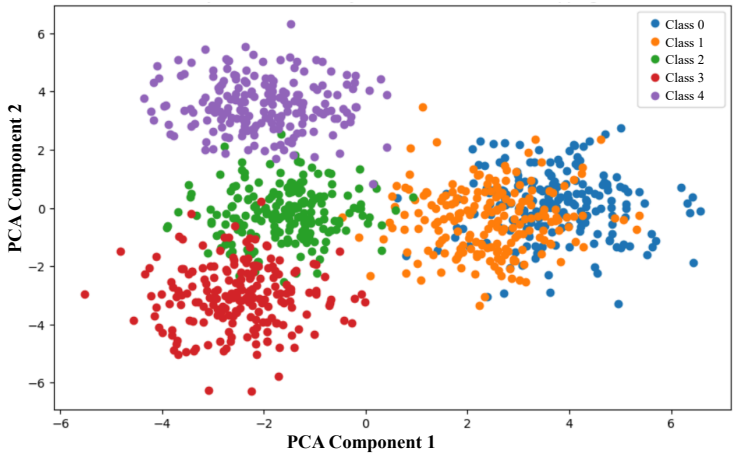


Figure 8: PCA analysis of classes in SynTab.

Training Hyperparameters. The training setup includes a batch size of 128 for processing training samples, and a replay buffer capable of holding up to 1M past experiences. The training begins after 50,000 steps, with updates occurring every four steps. The target network is updated every 1,000 steps. The learning rate is set at 0.001, with a smoothing factor of 0.95 applied for target estimation. Moreover, an exploration factor of 0.01 is used to balance exploration and exploitation during action selection.

G Implementation Details for Experiments

G.1 Datasets

G.1.1 SynTab

We generate a tabular dataset consisting of five classes, with each class containing 1000 samples. The dataset includes ten features, and each class is sampled from a Gaussian distribution. To introduce Level 1 errors, we make two classes share a similar distribution by assigning them close values for the mean μ and standard deviation σ in the Gaussian distribution. This can be observed in Figure 8, where class 0 and 1 are overlapping. For Level 2 errors, two out of the ten features are redundant, meaning they contain no class information. Using these two features for classification results in a Level 2 error. We randomly split 60% for $\mathcal{D}_{train}$, and the rest for $\mathcal{D}_{test}$. $\mathcal{M}$ is a trained logistic regression model, and the feature embedding weights before classification, are used as the explanation e. Please note that $\mathcal{M}$ is trained on a larger, separate dataset rather than $\mathcal{D}_{\text{train}}$, where $\mathcal{D}_{\text{train}}$ refers specifically to the training set used for the RL-based algorithm for simplicity. This applies to all other datasets used in the experiments.

G.1.2 SynImg

To further evaluate the performance of the proposed method on a more complex and larger dataset, we generate a challenging image dataset consisting of 24k images, with 12k for $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ each. This dataset is inspired by Yeh et al. [2020]. Each image contains up to 15 shapes, with only five of them being relevant to the ground truth class.

Figure 9 lists the shapes that used in SynImg. We use the first five shapes (q_i) in Figure 9 to determine the five concepts (c_i) in the dataset. Concretely, we define concepts using specific formulas. The first concept calculates a binary outcome based on the formula:

$$c_1 = ((1 - q_1 \cdot q_3) + q_4) > 0.$$

The second concept computes the sum of q_2 and the product of q_3 and q_4:

$$c_2 = q_2 + (q_3 \cdot q_4).$$

The third concept sums two products:

$$c_3 = (q_4 \cdot q_5) + (q_2 \cdot q_3).$$

The fourth concept uses the `bitwise XOR` operation between q_1 and q_2:

$$c_4 = q_1 \oplus q_2.$$

Finally, the fifth concept adds q_2 and q_5:

$$c_5 = q_2 + q_5.$$

After obtaining the concepts, we encode different combinations of concepts into ten classes, i.e., each sample is defined by a combination of the five concepts. Specifically, we use the following algorithm: (1) It takes a list of 5-digit binary numbers representing concept combinations, and turns each binary number into a regular number (decimal). (2) It assigns a label by dividing that number by ten and taking the remainder. That way, all labels are between 0 and 9. Figure 10 shows examples from the dataset, where the it demonstrates the corresponding concept and class.

On this dataset, we train a five-layer CNN as the model $\mathcal{M}$ and use GradCAM [Selvaraju et al., 2017] for the explanation e. Figure 11 show more examples. It is straightforward to obtain annotations for Level 1 error by comparing the predictions with the ground truth labels. As for the Level 2, since the positions of the five relevant shapes are known during dataset generation, we can assess the correctness of the explanation e, i.e., whether the shapes corresponding to the ground-truth class are correctly highlighted by e in the important areas based on a threshold.

G.1.3 VinDr-CXR

To verify the performance of our methods in real-world applications, we use a medical image dataset named VinDr-CXR [Nguyen et al., 2020]. It consists of more than 100k raw X-ray images in DICOM format, annotated for the presence of 14 types of thoracic abnormalities, with each finding localized using a bounding box. We extract 8513 samples for $\mathcal{D}_{\text{train}}$ and 8654 for $\mathcal{D}_{\text{test}}$, ensuring each sample is assigned an abnormality label along with its corresponding bounding box. A ResNet-50 is used as $\mathcal{M}$. Similarly to `SynImg`, we identify Level 2 errors using Grad-CAM [Selvaraju et al., 2017] as the explanation e. A Level 2 error is determined based on whether the top-K attribution scores in e achieve an Intersection over Union (IoU) with the bounding box exceeding a preset threshold.

In the real-world deployment of ML models in medical applications, Level 1 and Level 2 audits are critical for both patients and medical professionals. Level 1 is essential for patient safety, as incorrect predictions can lead to misdiagnoses. Level 2, on the other hand, is important for doctors, who must assess the accuracy of the model's decisions and determine whether to trust them. We use this dataset to demonstrate the practical application of our proposed RLAuditor.

G.1.4 e-SNLI

The e-SNLI (Explainable Stanford Natural Language Inference) dataset [Camburu et al., 2018] builds on the original SNLI dataset [Bowman et al., 2015] to help make AI models easier to understand. A sample in the SNLI dataset contains a pair of sentences that is labeled as entailment (one follows from the other), contradiction (they disagree), or neutral (no clear connection). e-SNLI adds human-written explanations for why the pair has such a relationship and also highlights the words that are important for the decision (also annotated by humans). This makes AI systems more transparent and reliable. There are 9842 samples in $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ each.

On this dataset, we finetune a pretrained BERT model [Devlin et al., 2018] as $\mathcal{M}$, and use SHAP [Lundberg and Lee, 2017] as e. To obtain Level 1 error labels, we annotate the samples that are misclassified. For Level 2 error annotation, we first normalize the SHAP values within each sample and retain only the tokens with importance scores above 0.5. Next, we create a separate set of tokens from the human-highlighted words provided in the dataset. If the IoU between these two sets is below a predefined threshold, the sample is labeled as a Level 2 error.

G.1.5 MATH

The MATH dataset [Hendrycks et al., 2021] contains competition mathematics problems with step-by-step solutions across seven subject categories: Prealgebra, Algebra, Geometry, Number

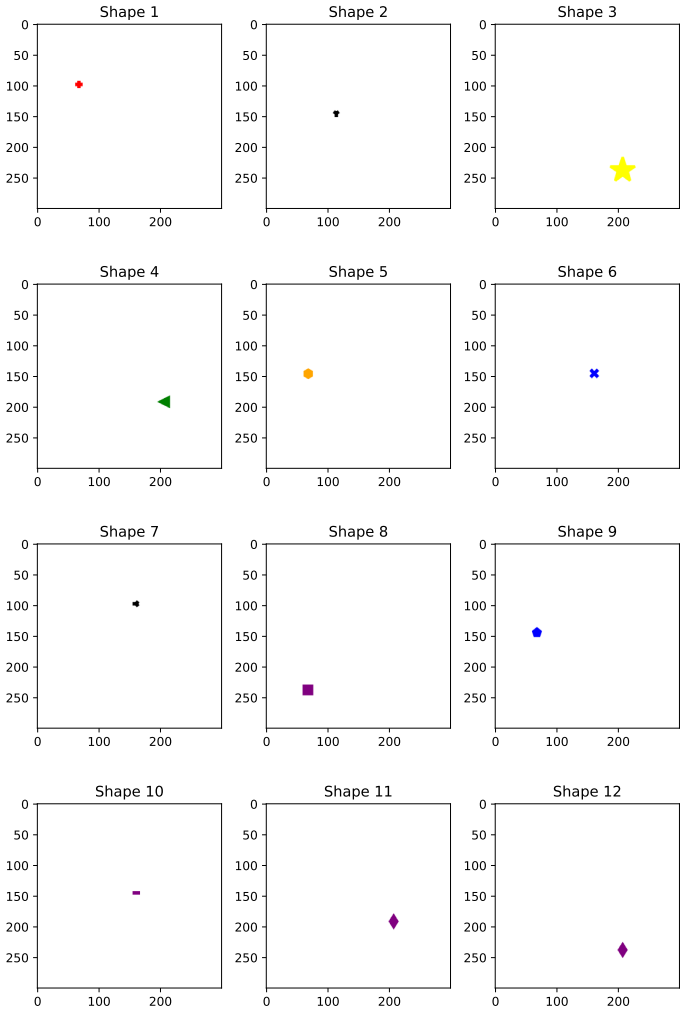


Figure 9: Shapes used in `SynImg`. The first five shapes contain class-discriminative information.

Theory, Counting & Probability, Intermediate Algebra, and Precalculus. We use this dataset to audit the mathematical reasoning of a LLM. The audited model $\mathcal{M}$ is DeepSeek-R1-Distill-Qwen-7B [DeepSeek-AI, 2025], a reasoning-focused LLM that produces step-by-step solutions.

We construct an auditing dataset of 1706 math problems balanced across the seven categories (approximately 250 per category, restricted to difficulty levels 1–3). For each problem, the LLM generates a step-by-step solution, from which we extract the final answer and parse the individual reasoning steps. We split the data into 70% as $\mathcal{D}_{train}$ for and 30% as $\mathcal{D}_{test}$. The LLM's chain-of-thought reasoning naturally serves as the explanation e. A Level 1 error corresponds to an incorrect final answer, while a Level 2 error corresponds to a factually incorrect reasoning step (even if the final answer happens to be correct). To detect Level 2 errors automatically, we employ Qwen2.5-Math-PRM-7B [Yang et al., 2024], a process reward model (PRM) that assigns a correctness score to each individual reasoning step. A step is flagged as incorrect when its PRM score falls below a threshold of 0.3.

The explanation embedding $\phi_{\text{explanation}}$ is computed using a sentence transformer (all-MiniLM-L6-v2) applied to the full reasoning chain produced by the LLM. The prediction features ϕ_{class} are derived from the PRM confidence scores aggregated per category.

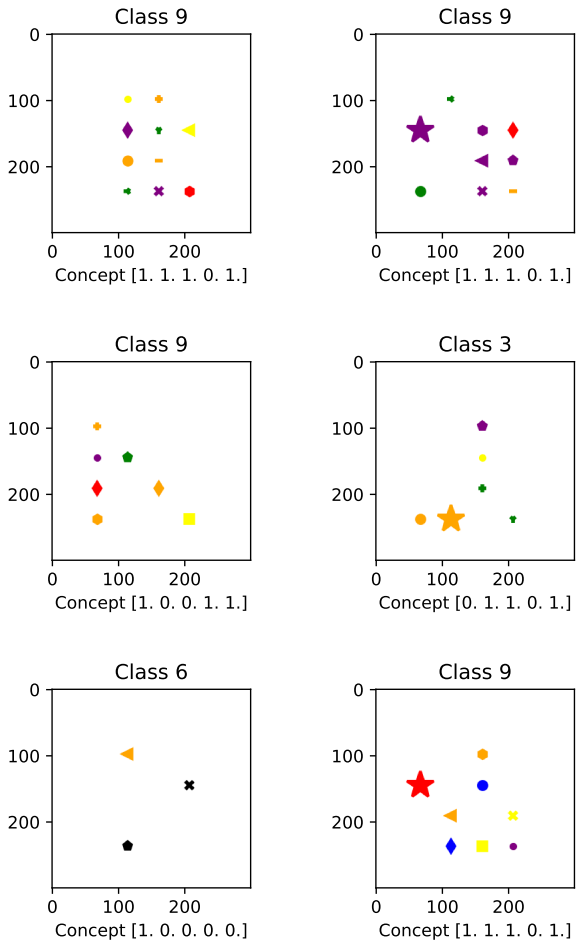
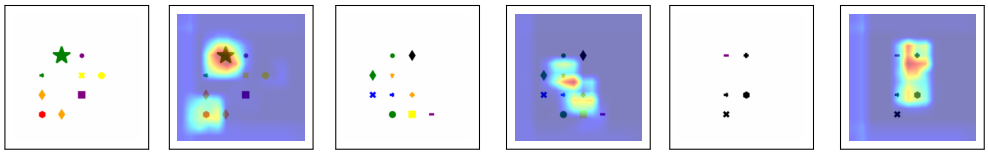


Figure 10: Examples from `SynImg`. Class and the concept labels are listed.



(a) (b) (c)

Figure 11: `SynImg`: Image x and its explanation e.

G.2 Audited ML Model

Table 3 lists the accuracies of the model $\mathcal{M}$ used in audits. Please note that the model is trained with a separate training dataset, which is different from the $\mathcal{D}_{train}$ used for training the RLAuditor. The size of the training set is 2k, 36k, 40.7k and 55k for `SynTab`, `SynImg`, VinDr-CXR, and e-SNLI, respectively (which is the original training set from the dataset). The validation set is in fact the $\mathcal{D}_{train}$ for training the RLAuditor, while the test set is $\mathcal{D}_{test}$. For MATH, the audited model is

Dataset	Val	Test
SynTab	0.87	0.86
SynImg	0.83	0.82
VinDr-CXR	0.72	0.70
e-SNLI	0.89	0.88
MATH	0.59	0.59

Table 3: Accuracies of the trained model $\mathcal{M}$ used in experiments for audits.

DeepSeek-R1-Distill-Qwen-7B (used in inference only, without further training). The reported accuracy is the answer-correctness rate on the auditing set.

Table 4: Audited model, prediction task, and error definitions for each dataset.

Dataset	Audited model $\mathcal{M}$	Prediction task (Y)	Level 1 error	Level 2 error
SynTab	logistic regression	5 classes from 10 tabular features	misclassification	reliance on the 2 class-uninformative features
SynImg	5-layer CNN	10 classes of synthetic images defined by 5 shape concepts	misclassification	Grad-CAM misses the class-relevant shapes
VinDr-CXR	ResNet-50	15 classes (14 chest X-ray abnormalities + no finding)	misclassification	Grad-CAM IoU with the finding's bounding box below threshold
e-SNLI	BERT	3 NLI labels (entailment / contradiction / neutral)	misclassification	SHAP-top tokens' IoU with human-highlighted words below threshold
MATH	DeepSeek-R1-Distill-Qwen-7B	solves competition problems (7 subject categories)	incorrect final answer	a reasoning step scored incorrect by the PRM

G.3 Baselines

Active Learning. Our paper includes baselines based on active learning methods. Specifically, we adopt the Expected Gradient Length (EGL) [Settles et al., 2007] and compute it as follows:

$$x_{\text{EGL}} = \arg\max_x \sum_i f_\omega(i \mid x) \, \|\nabla \, \ell_\omega(\mathcal{L} \cup \langle x, i \rangle)\|, \tag{6}$$

where $f_\omega(\cdot)$ denotes the trained user model in our case with parameters ω, $f_\omega(i \mid x)$ is the probability it assigns to label i, and the sum runs over its candidate labels. To include e in the input, we use the explanation e as the weighted mask in the same manner as proposed to construct the feature embeddings $\psi(x_k, e_k)$ for the state s in Section 3.2. ℓ_ω is the objective function for the model training, which is the cross-entropy loss, and $\mathcal{L}$ is the labeled training set. Let $\nabla \, \ell_\omega(\mathcal{L})$ be the gradient of the objective function with respect to ω. The Euclidean norm of this gradient, $\|\nabla \, \ell_\omega(\mathcal{L})\|$, should be nearly zero since the model converged in the last round of training [Settles et al., 2007]. Therefore, x_{EGL} can be simplified as:

$$x_{\text{EGL}} = \arg\max_x \sum_i f_\omega(i \mid x) \, \|\nabla \, \ell_\omega(\langle x, i \rangle)\|. \tag{7}$$

We run EGL on all samples in the test set $\mathbf{D}_{test}$ and select the top-K samples with the highest expected gradient length. Note that if the oracle label of a sample is available, it is used in the EGL computation. However, in practical auditing scenarios, oracle labels are typically unavailable, so we instead rely on the predicted labels.

DNN. We utilize the deep neural network from the DQN algorithm (i.e., the target network) as described in Algorithm 1. Specifically, the DNN is trained on the same dataset $\mathcal{D}_{\text{train}}$, assigning a score to each state-action pair analogous to a reward signal. It is optimized using the cross entropy loss with the reward $R(s, a)$ from the training set as the target. To ensure a fair comparison with the RL-based agent, we adopt the same training hyperparameters, setting the learning rate to 0.001 and the number of epochs to 200. After training, the DNN is evaluated on the $\mathcal{D}_{test}$.

	SynTab		SynImg		VinDr-CXR		e-SNLI		MATH		Housing Prices	
	Level 1 & 2	Level 2	Level 1 & 2	Level 2	Level 1 & 2	Level 2	Level 1 & 2	Level 2	Level 1 & 2	Level 2	Level 1 & 2	Level 2
Random	11.4 ± 2.3	3.5 ± 1.8	238.8 ± 10.9	188.8 ± 11.0	360.4 ± 13.9	61.8 ± 5.6	14.0 ± 2.4	8.6 ± 2.2	13.5 ± 0.2	5.6 ± 0.6	27.5 ± 15.3	1.6 ± 0.5
DNN	21.5 ± 2.3	5.0 ± 0.3	386.0 ± 3.7	0.0	438.5 ± 5.5	0.0	23.6 ± 2.5	13.2 ± 1.4	18.0 ± 1.1	2.6 ± 1.0	33.1 ± 3.3	11.0 ± 1.5
AL	28.0 ± 0	1.0 ± 0	320.0 ± 0	193.0 ± 0	490.0 ± 0	67.0 ± 0	28.0 ± 0	12.0 ± 0	16.8 ± 0.8	2.2 ± 1.2	—	—
W/o learned states	28.9 ± 1.5	6.4 ± 1.5	431.9 ± 11.9	328.8 ± 18.4	876.7 ± 6.4	152.3 ± 8.8	24.8 ± 0.7	16.0 ± 0.6	19.0 ± 0.7	10.9 ± 0.5	35.6 ± 4.1	15.3 ± 1.7
W/o expl	26.7 ± 2.2	5.4 ± 1.1	388.3 ± 4.2	210.2 ± 9.0	873.7 ± 7.6	160.2 ± 19.5	23.4 ± 0.6	14.6 ± 0.5	18.2 ± 0.7	10.2 ± 1.0	33.8 ± 2.7	15.2 ± 2.1
Ours	29.6 ± 1.3	9.0 ± 0.8	458.3 ± 3.2	336.8 ± 2.7	885.5 ± 7.2	163.1 ± 6.8	27.3 ± 0.9	17.0 ± 0.9	20.0 ± 0.5	13.0 ± 1.1	37.2 ± 4.7	17.0 ± 1.6

Table 5: Comparison of auditing results for two levels of errors across five datasets, plus the Housing Prices regression task. Each dataset's K is set to the same number for all methods. The number of identified errors is listed (mean $\pm$ std over 5 independent runs).

H More Experimental Results

In this section, we present additional quantitative results.

H.1 Full Results of Comparison

First, we report the complete results, including the mean and standard deviation on five independent runs, as shown in Table 5. The results for the first four datasets in Table 5 complement those presented in Figure 3 of the main paper, and the MATH column reports the results for auditing the LLM's mathematical reasoning.

We also include the two additional baselines compared to Figure 3:

- W/o learned states: This baseline does not utilize the states learned from the training set (assuming that the training and test sets share a similar data distribution), but initializes the state to zeros at the start of testing.
- W/o expl: This baseline does not include $\phi_{\text{explanation}}$ in the state s, although it still contains the labels for Level 2 errors during training.

It is worth noting that when the explanation features are not in the states, the detection of Level 2 errors is worse compared to our method. Moreover, if learned states are not used during testing, performance declines but the RL algorithm can still learn knowledge from the observed samples.

H.2 Extension to Regression Tasks

Section 5 mainly discusses RLAuditor performance on classification tasks. However, our algorithm can be extended to regression tasks with two main adaptations in: (1) constructing s and (2) designing the reward $R(s, a)$. In the original formulation, s is defined based on class information, which is not available in regression tasks. To address this, we manually partition the samples into N bins according to their predicted values. Once N is fixed, each sample can be assigned to a bin i, viewing it as a class. To obtain ϕ_{bin}, we get the feature embeddings before the regression prediction, and thus

$$\phi_{\text{bin}}(i, l) = \frac{1}{|\mathcal{U}_i|} \sum_{x_k \in \mathcal{U}_i} \psi_l(x_k),$$

where $\psi_l(x_k)$ denotes the feature embedding of dimension l for the sample x_k. On the other hand, the information for $\phi_{\text{explanation}}$ is constructed from the explanations e as

$$\phi_{\text{explanation}}(i) = \frac{1}{|\mathcal{U}_i|} \sum_{x_k \in \mathcal{U}_i} e_k,$$

where e_k denotes the explanation for sample x_k. In this setting, we use e_k directly as features rather than as a mask. This is because masking could distort the regression prediction, and explanations in regression tasks do not have spatial structure. Therefore, they can be averaged across each dimension.

To design the reward for regression tasks, we first compute the absolute prediction error for each sample as

$$\text{error}_k = |\hat{y}_k - y_k|,$$

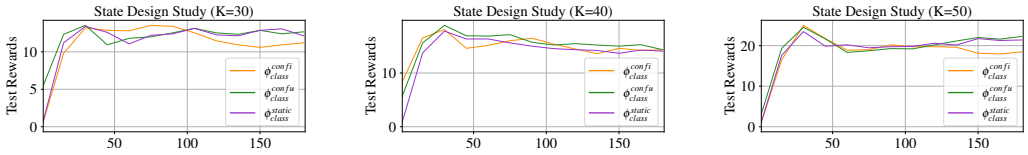
where $\hat{y}_k$ is the model prediction for sample x_k and y_k is its ground-truth target value. To emphasize larger errors, we introduce an exponent parameter $\alpha > 1$ and define the reward as

$$R_k = (\text{error}_k)^{\alpha}.$$

In this formulation, α controls the degree to which large errors are penalized. $\alpha > 1$ amplifies the contribution of larger errors. We set $\alpha = 2.5$. This design encourages the algorithm to prioritize samples where the model performs poorly, focusing learning on selecting samples with high uncertainty and predictive error.

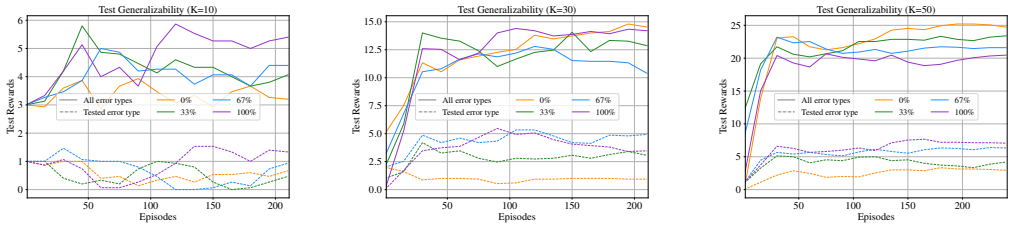
Dataset and Audited ML Model. We use California House Price dataset [Pace and Barry, 1997]. It includes property attributes, geographic information, and sale prices, and is used to analyze factors affecting home values and to build models for predicting future house prices. The dataset contains 20,640 samples, which are split into 12,384 for training, 4,128 for validation, and 4,128 for testing. The training set is used to fit a Random Forest regression model, which is the model to be audited. The model is configured with 200 estimators, no maximum depth restriction, and parallel processing to improve efficiency. The trained model achieves a test RMSE of 0.509 and an R^2 score of 0.811, indicating strong predictive performance on unseen data. When constructing the states for RLAuditor, the data is divided into bins based on quantiles, meaning each bin contains roughly the same number of values rather than equal numerical ranges. This ensures that the distribution is represented evenly across all bins, even if the data is skewed. We set N to 500. In practice, we found that a smaller N results in overly coarse categorization, which can reduce performance. We compute SHAP values to provide explanations. The level 1 error is estimated using R as described earlier. For the level 2 error, we label the top-10 most extreme feature importance scores for each feature as incorrect.

During training RLAuditor, the validation set is used, and the performance of the trained algorithm is evaluated on the test set. Table 5 lists the results of detected number errors within $K = 50$ steps. Note that AL requires gradient computations, which are not applicable to Random Forest models. Thus, this comparison is omitted from the table. Our algorithm continues to outperform the other methods, demonstrating that our algorithm also generalizes to regression tasks.



(a) $K = 30$ (b) $K = 40$ (c) $K = 50$

Figure 12: Comparison of test rewards when training with different state designs. The average test rewards over 15 episodes are shown. Three values of K (number of selection attempts) are evaluated to the effect of the state design.



(a) $K = 10$ (b) $K = 30$ (c) $K = 50$

Figure 13: Comparison of test rewards when training with $p\%$ ($p \in \{0, 33, 67, 100\}$) annotations of error in class B. The average test rewards over 15 episodes are shown. Three values of K are used.

H.3 Extended Figures

Figure 12 provides extended figures for Figure 4 in Section 5.2. We evaluate the performance under various values of K, and we can see that $\phi_{\text{class}}^{\text{confu}}$ consistently achieves the best results across all three settings. However, $\phi_{\text{class}}^{\text{static}}$ performs comparably when $K = 30$ or 50. Despite their strong performance, both settings have limitations in real-world applications. In contrast, while $\phi_{\text{class}}^{\text{confi}}$ is inferior due to limited access to additional knowledge, its performance only marginally decreases, demonstrating the robustness and quality of this design.

In Figure 13, more results of learning with limited supervision on class B are shown. We use $K = 10, 30$ and 50. From the result, we observe that with all the samples in B labeled ($p = 100$) during training, the number of the identified error for B (represented by the purple dashed line) is the highest among $K = 10$ and 50. As fewer in B are labeled, generally fewer errors are discovered. For example, in Figure 13c, the number of identified errors decreases as fewer samples are labeled. However, the best overall number of identified errors depends on the value of K. For instance, when $K = 50$, the setting with no labeled errors in B results in the highest total test reward, outperforming the setting where all samples in B are labeled, as shown in Figure 13c. This is because the algorithm finds it easier to learn when there are fewer error types.

Our RLAuditor on `SynTab` fails in scenarios when there is a nuance in the prediction distribution between two classes with fewer samples for the agent to learn from (e.g., class 4 and class 3, as shown in Figure 8), this can present a challenge for the agent. One potential failure case that our current method cannot detect involves using incorrect features that are not among the top-k most important features. Since the algorithm focuses on the top-k features, it may ignore these "non-dominant" features, which could lead to problems in safety-critical applications where certain features are strictly "forbidden." However, the reward function can be adjusted to address the concern.

H.4 Training Efficiency

In this section, we discuss the efficiency of the training of our RLAuditor. First, we elaborate the convergence of the training. Convergence means that the agent achieves a stable performance after training several epochs. From the learning curves in Figure 13 and Figure 12, we see that our model achieves a relatively stable score after 200 epochs. To evaluate its convergence, we report the performance of our trained agent by (1) averaging results over multiple consecutive episodes (15 episodes) and (2) plotting the reward curves in our experimental results. Furthermore, we use experience replay and a target network, which stabilize training, and an epsilon-greedy strategy for exploration. The converged training curve suggests that our reward design is sufficient to effectively guide learning.

Once the agent is trained, the inference process is highly efficient, as it only requires a simple feedforward pass through a DNN, given that we are using DQN. Specifically, we measured the training time on the SynTab dataset, where our method took approximately $3.8\times$ longer to train ($519s$ vs. $136s$) than the supervised learning (SL) method. However, the inference times for both methods are comparable, as both require a feedforward pass through the NN to compute the sample score (or label). Moreover, training efficiency is reflected in limited supervision, as analyzed in Section 5.3. With limited labeled data for error type B, our method still achieves reasonable performance in detecting errors.

H.5 Coverage of Error Types

An auditor that only rediscovers the most common error pattern is of limited use. We therefore report three complementary studies on `SynTab`: (1) Can the auditor find an error type it was never trained to look for? This is the limited-supervision study in Section 5.3, where class B's errors are removed from training. (2) Will the auditor spend budget on an error type that is scarce at audit time? Here the policy is trained normally, but only few such errors exist in the auditing pool (Table 6). (3) Does the auditor cover the full range of error classes rather than collapsing to a single pattern? (Table 7)

Setting. The audited model has an accuracy of 86%. The training pool contains 1000 samples, of which 89 are errors of a designated rare type R (other error types are untouched), and the policy is trained normally. In the test pool we keep only 10 type-R errors and delete the rest. We compare against LOA [Kang et al., 2022], XAudit [Yadav et al., 2024], and random selection, with $K = 50$, averaged over 5 seeds.

Rare error types. We define *rare-class recall* as the number of rare-class errors found within K divided by the number of rare-class errors present, and report it alongside total detection in Table 6. RLAuditor is high on both total detection and rare recall. LOA finds 13 errors in total, above random's 7.5, but only 0.2 of the 10 rare errors: it concentrates on common, easily scored errors, a property of static atypicality ranking. XAudit likewise fails to discover the rare errors.

Table 6: Rare error types on SynTab ($K = 50$, 10 rare errors in the pool; mean $\pm$ std over 5 seeds).

Method	Total found	Rare found (of 10)
RLAuditor	**28.0** $\pm$ 2.9	**3.4** $\pm$ 1.4
LOA	13.0 $\pm$ 1.1	0.2 $\pm$ 0.4
XAudit	12.6 $\pm$ 3.2	0.4 $\pm$ 0.5
Random	7.5 $\pm$ 0.3	0.6 $\pm$ 0.5

Diversity metrics. We report three metrics in Table 7:

- *Errors found*: total audit targets found.
- *Error-class coverage*: the fraction of error classes in which a method found at least one error. The denominator excludes classes that contain no errors at all (one of the five in SynTab).
- *Worst-class recall* (% of ceiling): recall on whichever error class the method handled least well, which directly measures whether an entire error class is neglected. We normalize by the best value any method could attain, namely an oracle allocating its K queries to maximize the minimum per-class recall (K/total errors $= 0.225$ here), and report each method's value as a percentage of this ceiling.

RLAuditor does not collapse to a single pattern (90% class coverage) and has the best worst-class recall. XAudit performs worst: it ranks each sample by how strongly the model's attribution falls on the pre-specified spurious features, so it can only select errors arising from that mechanism, and errors not involving those features are invisible to it. Because those features are active in only some classes, its selections concentrate on few classes (low class coverage). The same effect appears in the main comparison (Section 5.1), where XAudit's selections cover 1.8 of 5 classes (class entropy 0.35) versus 3.6 classes (0.79) for RLAuditor.

Table 7: Diversity of detected errors on SynTab

Method	Errors found ↑	Error-class coverage ↑	Worst-class recall (% of ceiling) ↑
RLAuditor	**34.0**	**90%**	**23%**
LOA	15.6	**90%**	9%
XAudit	15.6	45%	5%
Random	12.2	80%	4%

H.6 Robustness across Train/Test Splits

We additionally report results on 5 random train/test splits (Table 8). These results are comparable to those in Figure 3. Together, they show the robustness of our method.

Table 8: RLAuditor results over 5 random train/test splits (mean $\pm$ std; same K as Figure 3).

Dataset	Level 1 & 2	Level 2
SynTab	32.5 $\pm$ 3.6	10.6 $\pm$ 3.4
SynImg	422.4 $\pm$ 13.3	310.2 $\pm$ 10.3
VinDr-CXR	875.8 $\pm$ 15.6	143.2 $\pm$ 10.9
e-SNLI	31.0 $\pm$ 3.2	18.6 $\pm$ 2.1
MATH	18.0 $\pm$ 2.7	10.0 $\pm$ 1.3

I Evaluation with Human Auditors

We developed a user interface to enable the study of interactions between human experts and the proposed RLAuditor. The user study aims to address the following two research questions:

- R1: Can the samples selected by RLAuditor help users better understand the model's behavior and errors?

- R2: What strategies do human experts employ during audits?

In this section, we will introduce implementation details, including the model and the dataset used, followed by the user study details such as the procedure and the participant demographics, and additional analysis of the user study.

I.1 Implementation Details

Dataset and ML model. The Snips dataset serves as a benchmark for intent classification and slot-filling in spoken language understanding [Coucke et al., 2018]. It includes crowdsourced queries covering seven distinct intents, including "PlayMusic," "BookRestaurant," and "GetWeather." The dataset is commonly used to assess natural language understanding models. It consists of 13.1k queries for training and 1.4k for testing. We finetuned a small pre-trained BERT variant model [Bhargava et al., 2021] using the training set as our model to be audited $\mathcal{M}$. The explanation e is computed by the SHAP algorithm. We then use the subset from the test set as $\mathcal{D}_{train}$ to train our RLAuditor. To ensure the study was comprehensible for participants, we limited the audit dataset to 200 samples drawn from the test set. The trained model's accuracy on the test set is 89.5%.

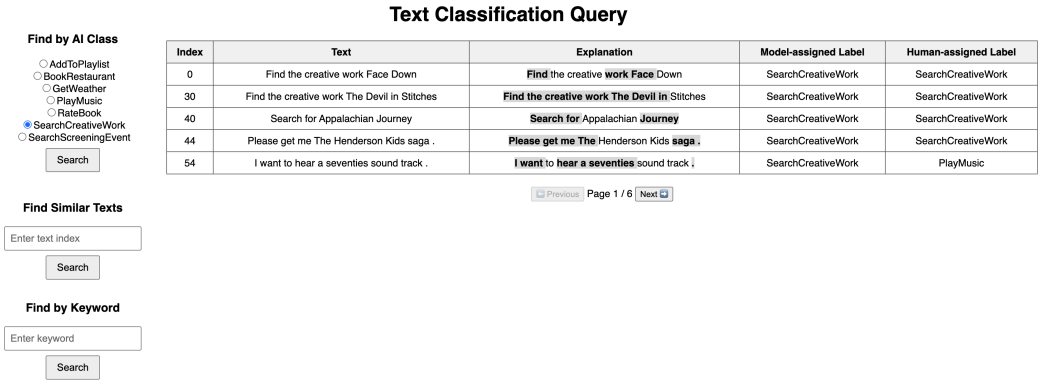


Figure 14: UI for the control group.

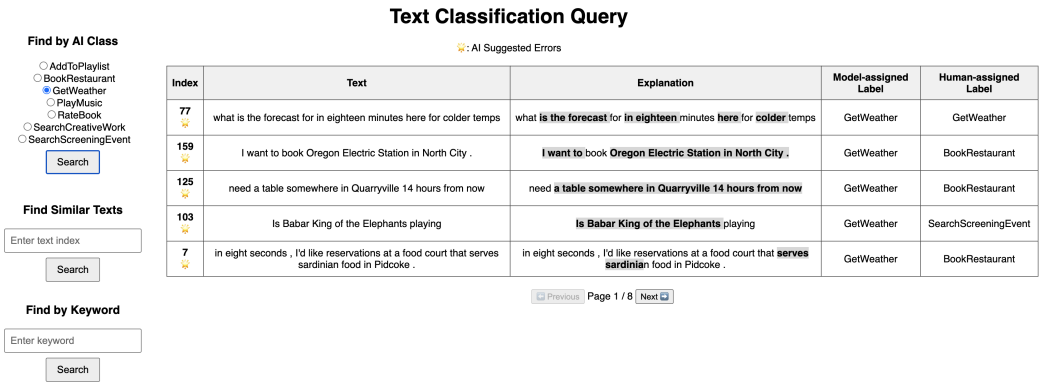


Figure 15: UI for the experimental group.

UI. Figure 14 and Figure 15 demonstrate the UI interface for each group. In this user interface, we provide three filtering techniques to assist users in exploring the data. First, samples can be categorized based on the class predicted by the model $\mathcal{M}$; this option is available in the top-left corner under the label "Find by AI Class". The second feature, "Find Similar Texts", allows users to investigate the model's behavior on inputs similar to a specific entry. By entering the index of a sentence, the system returns a list of texts ranked by cosine similarity between the queried sentence with the other test samples. The third function enables users to search for entries containing a specific keyword, offering a direct way to examine how the model handles certain terms or topics.

For each test sample, the user interface displays the original text, a visual explanation highlighting important words based on SHAP values, the model-assigned label, and the human-annotated (ground truth) label.

The key difference is that the experimental group was presented with samples recommended by our RLAuditor, which were highlighted and positioned at the top of the table.

I.2 User Study Procedure

This user study is approved by the IRB (Institutional Review Board). We recruited 14 participants from the university, all with backgrounds in engineering and general machine learning. Two of these participants conducted a pilot study to assess the clarity of the experiment procedure and evaluate the time constraints involved. The final procedure for the user study is as follows:

- Participants read the task description and receive an explanatory introduction of the UI to get familiar with the UI.
- Participants start to formulate their **report** with the help of the questions proposed.
- Participants answer the **objective questions** of model behaviors.

After the analysis task of the application audits, participants were asked to fill out a questionnaire about their experience interacting with the UI.

The whole session takes 40 minutes and each participant received a $\$10$ gift card as compensation. We conducted a between-group human experiment study. 6 participants in the control group did not receive the AI-suggested samples, while 6 participants in the experimental group did.

I.3 Participant Demographics

All participants were recruited via an institutional mailing list. Each has experience in evaluating and debugging machine learning (ML) models in applied settings and holds at least a bachelor's degree in computer science. The cohort includes PhD students, PhD candidates, and one postdoctoral researcher, making them well-qualified to perform this challenging task.

In the control group, there are two females and four males, with a mean age of 27.8 ± 4.2 years and an average of 5.3 ± 1.2 years of experience in ML. On a 5-point scale, they rated their familiarity with evaluating ML models at an average of 4.2 ± 0.7. The experimental group comprises five males and one female, with a mean age of 27.2 ± 2.2 years and an average of 5.6 ± 2.6 years of ML experience. Their self-reported familiarity score averages 3.6 ± 1.1.

I.4 Guiding Questions for Auditing

For each of the seven intent classes, we use the following questions to guide participants to formulate their audit report:

Misclassification Pattern

- Which other class(es) does the Model most frequently confuse with this class? (Multi-select problem)
- Why do you think these misclassifications occur? (Briefly explain why. With examples if possible.)

Feature Attribution

- Which features (e.g., specific words or phrases) does the Model rely on most when classifying this intent? (Give keywords.)
- Are these features appropriate and meaningful for this intent, or are they misleading/problematic? (Explain your judgment.)

They answer these questions for all seven classes in the dataset.

I.5 Objective Questions

At the end of the study, we also distributed a questionnaire to gather participants' feedback on their experience. The responses were used as follows:

1. How easy or difficult did you find the Model Auditing task? (1-Very Easy, 7-Very Difficult)
2. What aspects of the Model Auditing task did you find challenging, if any?
3. How did you conduct the Model Auditing task? (e.g., your strategy, functionality of the Auditing UI you found useful)
4. What additional information do you think you need for the Model Auditing task? (e.g., information about the dataset, potential features of the Auditing UI)
5. Do you think AI algorithms could assist you in this model auditing task? (Yes, No, or Unsure.)
6. Please share any additional comments or feedback about the task or study.

The second last question is designed for the experimental group.

I.6 Extended Analysis

Analysis of reports. Each reported Level 2 issue was manually checked against the model's behavior on the relevant samples to confirm that it reflected genuine reliance on unintended features, rather than simply being counted as reported. Level 1 statements were scored against the model's confusion matrix. When studying the model features observed by participants, the experimental group tended to agree on more meaningful and relevant features, as illustrated in Figure 17. For instance, for the class *Search Screening Event*, the experimental group referenced both verbs and nouns—such as "find," "want," "see," and "would like", as the control group focused more on nouns indicating screen programs. In another example *Rate Book* shown in Figure 16, both groups selected the words "rate" or "give" as the most important tokens, while the experimental group also pointed out the words "stars" or " points", which are also highly relevant to rating.



Figure 16: Word cloud in the model explanations used by the model for the class "*Rate Book*".

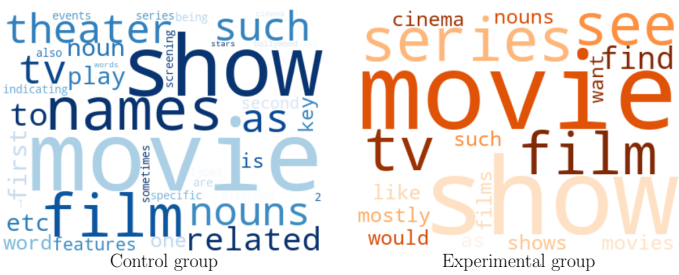


Figure 17: Word cloud in the model explanations used by the model for the class "*Search screening event*".

Analysis of objective questions. For the 1. objective question, the control group gives a score of 3.8 ± 1.7, while the experimental group gives a score of 3.0 ± 1.4. The difference between the groups was not statistically significant, with a p-value is 0.5. However, it shows that the participants with the assistant of RLAuditor rate the auditing task as easier than the control group.

For the second question, participants expressed various challenges. For example, one from the control group noted, *"It was challenging to answer questions for specific categories without having reviewed the rest of the categories."* A holistic and comprehensive view is essential for auditing. However, obtaining such an overview becomes increasingly impractical for humans as the dataset size grows, as another participant from the control group stated *"Checking all the classes for all possible mistakes."* We believe that human-AI collaboration is a promising way to solve this issue. Furthermore, two participants in the experimental group claimed that identifying features used by the model is difficult.

Regarding the 4. question, it is worth noting that one participant in the control group requested a "better filter/ranking mechanism." This suggests that human auditors also appreciate some form of pre-filtering, which can be effectively provided by an algorithm. This feedback further supports the usefulness of our proposed algorithm. One participant in the experimental group suggested including the model's confidence alongside the AI-suggested errors. Moreover, the importance scores for each token in the explanations were also noted as useful. The feature allowing users to view samples by predicted class was utilized by all participants, while the other functions were rarely used. This suggests that for future auditing UI design providing a clear and well-organized view including classification statistics is critical.

In the general feedback on our study (last question), two participants agreed that more samples should be displayed, preferably on a full page. This suggests that human auditors benefit from reviewing a larger number of examples at once to gain a comprehensive understanding of the model's behavior.

J Computational Infrastructure Details

All experiments in this paper are conducted on the device given in Table 9.

Table 9: Computational infrastructure details.

Device Attribute	Value
Computing infrastructure	GPU
GPU model	NVIDIA A40
GPU number	1
CUDA version	12.3

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction precisely describe our three main contributions: (1) an MDP whose state operationalizes a simplified mental model of the audited system, (2) the RLAuditor algorithm that learns to select informative test samples to uncover Level 1 and Level 2 errors, and (3) empirical validation across multiple data modalities and a human subject study. These claims are substantiated by the theoretical results in Section 4 and the empirical results in Sections 5 and 6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations and future work are discussed at the end of the conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The two theoretical results Proposition 3 and Theorem 4 are stated formally in Section 4 with all assumptions explicitly listed. Complete proofs are provided in Appendices C and D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 3.2 fully describes the algorithm, and Appendix F provides the training algorithm (Algorithm 1), Q-network architecture, hyperparameters, and other implementation details. Appendix G describes each dataset, the audited model, the explanation method, the labeling protocol for Level 1 and Level 2 errors, and the data splits. All datasets and pretrained models we use are publicly available.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our source code is available at `https://github.com/yaorong0921/RLAuditor`, including scripts to reproduce the audit pipeline, embedding extraction, RLAuditor training, and baseline comparisons. All datasets used in our experiments are publicly available. References and access details are provided in Appendix G.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main experimental setup is described in Section 5, including the budget K for each dataset, baselines, and evaluation protocol. Appendix F provides full hyperparameters and the training algorithm, and Appendix G provides per-dataset training/test splits, the audited model and explanation method, and baseline configurations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Table 5 reports mean $\pm$ standard deviation over 5 independent runs, and Table 8 addresses different random train/test splits. The user study results in Section 6 report standard deviations along with p-values for the comparison of objective questions between the two groups.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix J reports the compute infrastructure used for all experiments (GPU type, count, and CUDA version).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and our research conforms to it. The user study was approved by the Institutional Review Board (IRB), participants were compensated, and anonymity is preserved in the submission. All datasets used are publicly available and properly cited.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: ML auditing has clear positive societal impact: it helps ensure responsible deployment of ML models in safety-critical applications such as medical diagnosis. Our work supports human-AI collaboration in such audits and is unlikely to have direct negative societal impact, as it is intended to help auditors find errors in ML systems rather than build them. We discuss broader implications in the conclusion.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release any new pretrained models, image generators, or scraped datasets that pose a high risk for misuse. We only use existing publicly available datasets and pretrained models for evaluation.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets and pretrained models are properly cited at first use in the main paper and detailed in Appendix G: VinDr-CXR [Nguyen et al., 2020], e-SNLI [Camburu et al., 2018], MATH [Hendrycks et al., 2021], Snips [Coucke et al., 2018], DeepSeek-R1-Distill-Qwen-7B [DeepSeek-AI, 2025], Qwen2.5-Math-PRM-7B [Yang et al., 2024], BERT [Devlin et al., 2018], and SHAP [Lundberg and Lee, 2017]. We respect the licenses and terms of use of all assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The source code for RLAuditor (training script, environment, baselines, and audit pipeline) is released at `https://github.com/yaorong0921/RLAuditor` with documentation describing how to run each component. We do not release new datasets or pretrained models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

Justification: The user study procedure, recruitment, compensation ($10 per participant), guiding questions, and objective questions used in the study are described in detail in Appendix I. UI screenshots are included to illustrate what participants saw.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [Yes]

Justification: As described in Appendix I, the user study was approved by the Institutional Review Board (IRB). Participants were informed of the procedure prior to participation and consented. The study did not involve any risks beyond those of routine computer-based tasks.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We use LLMs in two distinct ways. First, the audited subject in our MATH experiment is an LLM (DeepSeek-R1-Distill-Qwen-7B), and we use a process reward model (Qwen2.5-Math-PRM-7B) for Level 2 error labeling; both are described in Appendix G. Second, LLMs were also used only for minor grammar and language polishing of the manuscript and did not influence research ideation, experimental design, or scientific content.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.