
**VILNIUS UNIVERSITY**

**FACULTY OF MATHEMATICS AND INFORMATICS**

**DATA SCIENCE STUDY PROGRAMME**

Master's thesis

# Estimation of Rare Event Prevalence in Online Platforms with Imperfect ML and Human Moderation

## Retų reiškinių paplitimo įvertinimas interneto platformose naudojant netobulą mašininio mokymosi ir žmogaus moderavimo sistemą

Vytautas Dominykas Leipus

Supervisor : Dr. Rasa Giniūnaitė

Scientific advisor : Jevgenij Gamper

**Vilnius**
**2026**

## Acknowledgements

I want to thank my supervisor for guidance and feedback during the early stages of this thesis, and my technical advisor for technical advice during the initial development of the project. Any remaining errors are my own, and not a reflection of the guidance provided.

## Summary

Estimating prevalence under rare-event conditions is a persistent challenge in large-scale content moderation and monitoring systems, where observations are often collected through non-random, model-assisted sampling pipelines. This thesis investigates how design choices across such pipelines, spanning sampling strategies, classifier performance, and decision thresholds, jointly affect prevalence estimation accuracy. Using a simulation-based framework, the estimation process is decomposed into modular components and their contributions to bias and variance are systematically analyzed.

Multiple sampling schemes, including simple random sampling and classifier-assisted approaches with human moderation, are evaluated across controlled parameter grids that reflect realistic operational constraints. Rather than optimizing a single configuration, the study focuses on characterizing trade-offs and interaction effects between components. Performance is assessed using bias, variance, and error decomposition metrics, enabling comparison of how uncertainty propagates through the pipeline.

Results indicate that targeted sampling can substantially reduce variance but may introduce non-negligible bias when classifier error rates or decision thresholds are misaligned with true prevalence levels. More conservative designs produce stable but less efficient estimates. Overall, the findings demonstrate that no single parameter choice is universally optimal. Instead, reliable prevalence estimation depends on coherent coordination between sampling and modeling decisions. The proposed framework offers a structured approach for analyzing such systems and provides practical guidance for designing robust prevalence estimation pipelines under rare-event conditions.



**Keywords: Prevalence estimation, Rare events, Bias, Human-in-the-loop moderation**

# Santrauka

Paplitimo (prevalence) įvertinimas retų įvykių sąlygomis išlieka reikšmingu iššūkiu didelio masto turinio moderavimo ir stebėsenos sistemose, kur stebėjimai dažnai renkami naudojant neatsitiktines, modeliais paremtas atrankos procedūras. Šiame darbe nagrinėjama, kaip sprendimai, priimami skirtinguose tokios sistemos etapuose, apimančiuose atrankos strategijas, klasifikatorių veikimo charakteristikas ir sprendimų slenksčius, bendrai veikia paplitimo įverčių tikslumą. Taikant simuliacijomis paremtą metodologinę sistemą, įvertinimo procesas suskaidomas į modulinius komponentus ir sistemiškai analizuojamas jų indėlis į poslinkį (bias) ir dispersiją.

Vertinamos kelios imties atrankos schemos, įskaitant paprastą atsitiktinę atranką ir klasifikatoriais paremtas strategijas su žmogaus moderacija, naudojant kontroliuojamus parametrų rinkinius, atspindinčius realistiškus ribotumus. Užuot siekiant optimizuoti vieną konfigūraciją, darbe dėmesys skiriamas kompromisų ir sąveikos efektų tarp sistemos komponentų analizei. Veikimas vertinamas remiantis poslinkio, dispersijos ir paklaidos dekompozicijos metrikomis, leidžiančiomis įvertinti, kaip neapibrėžtumas sklinda per visą sistemą.

Rezultatai rodo, kad tikslinga atranka gali reikšmingai sumažinti dispersiją, tačiau netinkamai suderinti klasifikatoriaus klaidų rodikliai ar sprendimų slenksčiai gali lemti reikšmingą poslinkį. Konservatyvesni sprendimai užtikrina stabilesnius, bet mažiau efektyvius įverčius. Apskritai nustatyta, kad universaliai optimalaus parametrų pasirinkimo nėra, o patikimas paplitimo įvertinimas priklauso nuo nuoseklaus atrankos ir modeliavimo sprendimų suderinimo. Siūloma metodologinė sistema suteikia struktūruotą pagrindą tokių sistemų analizei ir praktines gaires patikimų paplitimo įvertinimo procesų kūrimui retų įvykių sąlygomis.



**Raktiniai žodžiai: Paplitimo įvertinimas, Reti įvykiai, Žmogaus dalyvavimu grįsta moderacija**

# List of Figures

# List of Tables

# Contents

# Introduction

Online platforms must detect and estimate the prevalence of harmful or fraudulent items, such as scams, counterfeit products, or misinformation. This task is difficult due to the low prevalence of such items and the imperfect accuracy of both machine learning systems and human moderators. Despite their rarity, unwanted events can be costly both to users and platforms, not only in monetary terms but also through reputational damage, loss of trust, and reduced participation. At the same time, false-positive decisions in human-in-the-loop moderation systems may unjustly penalize legitimate users or sellers, introducing additional social and economic costs. Estimating the true rate of harmful content or products therefore requires correcting for detection errors and understanding the impact of sampling design. This project uses a simulation approach to explore how these factors influence the accuracy and reliability of prevalence estimates. This project is inspired by the methodology outlined by Yi Liu in the blog post Estimating Prevalence of Rare Events [23], which presents a practical framework for prevalence estimation under imperfect detection conditions.

In practice, such detection and moderation pipelines are deployed at scale by major online platforms, including social media and professional networking services such as Facebook and LinkedIn, where automated classifiers are used to triage content or accounts for further human review. Due to the rarity of harmful events and the selective nature of these pipelines, observed samples are often highly non-representative of the underlying population. As a result, accurate prevalence estimation requires explicit correction for detection errors and careful accounting of uncertainty introduced at each stage of the moderation process.

Building on this framework, the present study focuses on the interaction between machine learning-based detection, sampling strategies, and moderator accuracy, with particular attention to the resulting bias-variance trade-offs in prevalence estimation. Rather than treating detection and correction as fixed components, the simulation systematically varies classifier operating characteristics, sampling designs, and moderator performance to examine how each stage contributes to estimation bias, variance. This allows for a decomposition of uncertainty across the pipeline and highlights regimes in which improvements in one component yield diminishing returns due to downstream correction effects.

# 1 Literature review

This literature review focuses on the components of the detection and labeling pipeline that directly shape the prevalence estimation problem. First, it surveys machine-learning driven risk scoring and sampling decisions, emphasizing how prediction models interact with sampling rules to generate selective labels. Second, it examines research on selective sampling and measurement bias, including classical probability sampling and modern score-based selection mechanisms, to establish how unequal inclusion probabilities affect observed data. Third, it reviews work on human moderators as noisy labelers, highlighting models of reviewer error and their implications for downstream inference. Finally, the review covers estimation methods designed to correct sampling and measurement errors, as well as related studies in platform integrity and fraud detection that motivate the application context. These topics are included because together they define the statistical and operational environment in which prevalence must be estimated, and they reveal where existing approaches fall short when sampling bias and human labeling noise occur simultaneously.

## 1.1 Risk Scoring and Machine Learning in Enforcement Pipelines

This section reviews literature on the use of machine-learning-based risk scoring within enforcement and moderation pipelines, focusing on how feature-based models are used to estimate the likelihood of policy violations and to guide downstream decisions. The reviewed work examines how model outputs are used as risk scores, how decision thresholds are chosen, and how these scores are subsequently used to prioritize items for review or to define biased sampling schemes. The literature also highlights the implications of model error for downstream evaluation and prevalence estimation, providing the conceptual grounding for the sampling and estimation approaches discussed in later sections.

Rather than producing binary accept/reject decisions, many modern machine-learning systems are designed to output continuous risk scores that estimate the likelihood of an undesirable event. This approach allows systems to express uncertainty and relative risk more flexibly than hard classifications. The framework studied in this thesis follows this paradigm by using model-generated risk scores as inputs to downstream selection and decision processes, rather than as final determinations. In practice, such scores may represent probabilities, calibrated likelihoods, or risk rankings derived from model outputs. This use of outputs is especially common in domains such as credit scoring, fraud detection, content moderation, and platform governance, where decisions often involve trade-offs between false positives and false negatives rather than absolute correctness [10, 21].

In high-stakes or other regulated settings, there is a strong preference for interpretable and stable models over highly complex or opaque alternatives. Interpretable models enable practitioners to reason about feature influence, diagnose failures, and provide explanations to affected users or stakeholders. Stability is particularly important when risk scores influence downstream actions, as small changes in input data or retraining procedures should not result in disproportionate changes in system behavior. Prior research has shown that simpler models often perform competitively with more complex methods while offering substantially improved transparency and robustness [3, 32].

Beyond raw predictive accuracy, calibration and monotonic ranking properties are also critical objectives in risk-scoring systems. A well-calibrated model ensures that predicted scores correspond meaningfully to empirical outcome frequencies, which is essential when scores are interpreted as probabilities or thresholds are applied consistently across populations. Additionally, maintaining a monotonic relationship between scores and true risk enables reliable prioritization even if exact probability estimates are imperfect. In many applied contexts the ability to correctly rank cases by relative risk is more valuable than minimizing classification error alone [9, 25].

Importantly, risk scores are typically not final decisions, but rather inputs into broader decision-making pipelines. Human review, policy rules, resource constraints, and contextual signals may all influence the final outcome. For example, moderation systems may use risk scores to prioritize content for review, or trigger additional verification steps. This separation between scoring and decision-making allows platforms to adjust policies and thresholds without retraining underlying models, and helps align technical outputs with evolving legal, ethical, or organizational requirements [17].

Finally, score-based prioritization has important implications for the composition of observed data. When systems preferentially act on high-risk cases, the resulting feedback loop can bias the data available for future model training or evaluation. This phenomenon, sometimes described as selective labeling or intervention bias, can distort estimates of model performance and obscure the true distribution of risk in the population. Understanding these dynamics is crucial for designing monitoring strategies, updating models responsibly, and avoiding unintended reinforcement of existing biases [15, 20].

## 1.2 Sampling and Selective Observation

Selective sampling arises whenever observations are collected through mechanisms that give certain items a higher probability of being reviewed or labeled than others. In many modern detection systems, this occurs because operational or computational constraints require platforms to inspect only a subset of all items, leading to inclusion probabilities that vary systematically across the population. Such unequal sampling creates a divergence between the distribution of observed labels and the underlying population distribution, introducing bias if not properly accounted for. This section reviews the theoretical foundations relevant to this problem, beginning with classical probability and non-probability sampling strategies and inverse-probability weighting, then examining how machine-learning models induce score-dependent sampling in practical systems, and finally discussing how selective measurement generates Missing-Not-At-Random (MNAR) patterns that compromise naive prevalence estimation. Together, these topics establish the statistical context in which unbiased estimation must operate when only a selectively labeled subset of data is available.

### 1.2.1 Probability sampling in rare event estimation

Classical sampling theory provides the foundational framework for understanding how information obtained from a subset of a population can be used to infer population-level quantities. Central to this framework is the distinction between probability sampling and non-probability sampling.

In probability sampling designs, each unit in the population has a known and typically positive inclusion probability, making it possible to derive unbiased estimators of totals, means, and prevalences. In contrast, non-probability or informative sampling designs lack known inclusion probabilities, either because units are selected according to operational rules, expert judgments, or algorithmic procedures. When inclusion probabilities are unknown or depend on unobserved characteristics, naive estimators, such as the raw sample mean, become biased and cannot be interpreted as representative of the full population.

A key insight from probability sampling is that unequal inclusion probabilities do not, by themselves, prevent valid inference. Rather, they require estimators that explicitly account for the sampling design. The most prominent example is the Horvitz–Thompson estimator [13], which adjusts each sampled unit by the inverse of its inclusion probability. If unit $i$ is sampled with probability $\pi_i$, then its contribution is weighted by $1/\pi_i$. This inverse probability weighting principle ensures that units sampled with greater frequency are appropriately down-weighted, while units that are rarely sampled are correspondingly up-weighted. Under correct specification of the inclusion probabilities, the Horvitz-Thompson estimator is unbiased for a wide range of estimates, including population means and prevalence rates. Related developments, such as the Hansen-Hurwitz estimator [11] and model-assisted strategies described in classical survey sampling texts [7, 33], further formalize conditions under which unequal-probability sampling can still yield reliable inference.

The relevance of these ideas becomes particularly clear in settings where rare events are of primary interest. In fraud detection, spam filtering, and similar domains, platforms often intentionally over-sample suspicious or high-risk items to ensure adequate coverage of the positive class. Although practically justified, this practice produces sample prevalence values that sometimes differ by orders of magnitude from the true population prevalence. Without correction, such samples systematically overstate the frequency of harmful outcomes. Classical sampling theory thus provides essential conceptual tools: it demonstrates how selection processes distort observed data and how inverse probability weighting can, in principle, undo such distortions.

However, traditional sampling designs typically rely on predefined, design-based inclusion probabilities, whereas modern detection systems increasingly employ algorithmically determined inclusion mechanisms. Instead of assigning units fixed probabilities prior to data collection, platforms rely on machine-learning models to score items and route only the highest-risk cases to downstream processing. The resulting observation process is no longer a classical probability sample, but a score-dependent or informative sample whose inclusion probabilities depend on unit-level features and the behavior of the predictive model. This shift motivates the transition to the next subsection, which examines how classical principles of importance weighting manifest and become more complex when sampling decisions are driven by machine-learning risk scores rather than designed sampling plans.

### 1.2.2 Non-probability sampling in rare event estimation

Non-probability sampling refers to data collection processes in which inclusion in the sample is not governed by a randomized design and the inclusion probabilities of individual units are unknown, uncontrolled, or only partially specified. In such settings, units may be selected based on

convenience, expert judgment, heuristic rules, or some observed characteristics correlated with the outcome of interest. Because the sampling mechanism depends on unit-level attributes rather than a predefined design, the resulting sample is typically informative, meaning that the distribution of observed outcomes differs systematically from that of the underlying population. As a consequence, standard estimators that assume representative samples are generally biased, and additional modeling assumptions or correction techniques are required to support valid inference.

Non-probability sampling is widely used in applied settings where efficiency and risk prioritization outweigh the need for randomization. Classical examples include case–control designs in epidemiology, where samples are intentionally enriched for positive outcomes to study rare events [4, 28]. Similar principles underlie targeted audit and inspection programs, such as tax enforcement or regulatory compliance checks, in which units are selected based on suspicion indicators rather than random assignment [1]. In modern detection systems, non-probability sampling commonly arises through selective labeling pipelines, where outcomes are observed only after a human or algorithmic decision routes an item for further review [16, 20]. Across these domains, samples are systematically enriched for high-risk cases, resulting in observed data that are informative but not representative of the underlying population.

## 1.3 Prevalence Estimation in Rare-Event Settings

Rare-event settings pose several challenges for statistical estimation because outcomes of interest occur infrequently but carry disproportionate legal, financial, or societal consequences. In settings such as fraud detection, content moderation, or regulatory enforcement, accurate prevalence estimates are essential for platform governance, policy evaluation, and resource allocation, yet are difficult to obtain due to extreme class imbalance and asymmetric cost of errors. Observed enforcement outcomes, such as investigated cases or flagged items, reflect selective intervention rather than the true underlying event rate, making naive estimators unreliable. Moreover, strong predictive performance at the individual level does not necessarily translate into accurate population-level inference, particularly when decisions are driven by risk-based prioritization. This section examines the defining characteristics of rare-event domains and outlines why prevalence estimation in such settings requires approaches that explicitly account for selection mechanisms and downstream biases, while also foreshadowing the pipeline perspective developed in subsequent sections.

Many moderation and fraud-detection problems fall within rare-event domains, characterized by extreme class imbalance and asymmetric costs of error. Harmful events may constitute a very small fraction of all observed items, yet carry disproportionately high consequences when missed. Furthermore, false positives may impose significant costs on legitimate users or content creators. These characteristics complicate both modeling and evaluation, as standard classification metrics and estimators may perform poorly or become misleading under severe imbalance.

A common challenge in such settings arises from the use of naive prevalence estimators based on observed enforcement outcomes, such as the proportion of reviewed items found to violate policy. These estimators implicitly assume that reviewed cases are representative of the broader population. In practice, however, enforcement outcomes are conditional on prior filtering, risk scoring, and

human moderation decisions. Because low-risk items are often excluded from review altogether, observed enforcement data reflect a highly selected subset of cases, rendering unadjusted prevalence estimates biased and generally uninformative about population-level rates [20, 22].

This issue highlights a fundamental distinction between predictive performance and population-level inference. Models optimized for detecting rare harmful events may achieve strong ranking or classification performance on observed data while still providing little insight into the true prevalence of those events in the underlying population. Predictive accuracy on selectively labeled samples does not imply valid inference about unobserved cases, particularly when selection mechanisms depend on model outputs or heuristic rules. Conflating these objectives risks overstating what can be learned from operational metrics alone [17].

Finally, prevalence estimation in rare-event settings is shaped by multiple, interacting sources of bias, which are further explored through a pipeline perspective. These include human moderator judgment and inconsistency, non-probability or rule-based sampling strategies, and systematic avoidance of low-risk items. Each stage of the pipeline influences which cases are observed, labeled, and counted, making prevalence an emergent property of the system rather than a directly measurable quantity. Recognizing these biases upfront provides essential context for the methodological choices and inferential limits discussed in subsequent sections [24].

## 1.4 Moderation and (noisy) Label Generation

In many enforcement and monitoring pipelines, the labels used for analysis are not direct observations of ground truth but the result of human moderation processes operating under policy, time, and resource constraints. Human moderators act as noisy and heterogeneous measurement instruments: their decisions may vary across individuals, over time, and across ambiguous cases, leading to systematic disagreement and asymmetric error patterns. Furthermore, labels are generated only for items that have already passed through a prior upstream selection mechanism, making label availability conditional on prior sampling decisions rather than uniformly observed across the population. These features imply that observed labels reflect both moderation noise and selective observation, with important consequences for downstream inference. This section examines how human labeling processes shape the data-generating mechanism and discusses why explicitly modeling moderation noise is essential for valid prevalence estimation.

Recent empirical work examining human moderation in ML-assisted decision pipelines highlights consistent patterns of noise, discretion, and interaction effects that complicate downstream inference. Lai et al. [19] study how human decision-makers engage with model predictions in high-stakes classification tasks, demonstrating that moderators do not simply accept model outputs but selectively override, reinterpret, or defer to them depending on confidence, context, and perceived risk. This interaction introduces systematic variation in error rates that cannot be attributed to either the model or the human in isolation. Similarly, Wang et al. [34] document how human reviewers adapt their labeling behavior in response to model guidance and interface cues, leading to heterogeneous and dynamically shifting decision thresholds across moderators. These findings indicate that moderation noise is shaped not only by individual subjectivity but also by the structure of the

human–AI interaction itself. More recently, Koshy et al. [18] analyze a large-scale moderation system and show that enforcement outcomes are influenced by policy ambiguity, reviewer discretion, and temporal changes in platform priorities, resulting in non-stationary and asymmetric labeling errors. Across these studies, a common theme emerges: labels generated through human moderation are conditional on both prior algorithmic selection and the form of model assistance provided, producing observed data that reflect a coupled human–machine decision process rather than independent noisy measurement. This body of work motivates treating moderation as an imperfect, context-dependent labeling mechanism whose interaction with selective sampling must be explicitly accounted for in prevalence estimation.

## 1.5 Estimation Under Selective and Noisy Observation

This section synthesizes the implications of selective sampling and noisy labeling for statistical inference. When observations are both selectively generated and imperfectly measured, standard identification assumptions underlying prevalence estimation no longer hold, and naive estimators can be severely biased. In such settings, inference must be understood as conditional on the observation process and the assumptions used to model it. Rather than focusing solely on point estimation, this section reviews how reweighting, partial identification, and sensitivity analysis provide complementary tools for reasoning about prevalence under selection. Emphasis is placed on robustness and variance considerations, as well as on the fundamental limits of what can be inferred without randomization or strong structural assumptions.

In many applied machine-learning systems, model estimation is performed on data that are selectively observed and imperfectly labeled, rather than on a random sample of the underlying population. In moderation and risk-scoring pipelines, labels are often generated only for a subset of cases, typically those deemed sufficiently risky to warrant further inspection or intervention. As a result, observed outcomes are conditional on prior system decisions, and unobserved cases may differ systematically from those included in the training or evaluation data. This form of selective labeling complicates both estimation and inference, as standard assumptions about data availability and representativeness no longer hold (Lakkaraju et al., 2017; Kleinberg et al., 2018).

A key implication of such selective observation is the breakdown of standard identification assumptions under missing-not-at-random (MNAR) mechanisms. Because low-risk items are frequently omitted from the labeled dataset, missingness depends directly on latent risk and model predictions rather than occurring randomly. In this setting, naive estimators that treat observed labels as representative can produce biased estimates of error rates, calibration, or treatment effects. The focus on specific event detections, such as confirmed policy violations or flagged incidents, induces a structural dependence between selection and outcome that violates the assumptions underlying conventional supervised learning and statistical evaluation frameworks [22].

To address inference under selection, several methodological approaches have been proposed, each relying on different assumptions. Reweighting methods attempt to correct selection bias by modeling the selection process explicitly and adjusting contributions accordingly, though their validity depends on correctly specified selection models. Partial identification and bounding approaches

relax point-identification requirements by deriving ranges of plausible parameter values consistent with observed data and weak assumptions. Sensitivity analysis explores how estimates vary under alternative assumptions about unobserved cases, allowing researchers to assess robustness without committing to a single, unverifiable model of missingness [24, 31].

Given these limitations, it is worthwhile to focus on variance decomposition and robustness analysis rather than precise point identification. Instead of reporting single estimates as definitive, researchers may quantify uncertainty arising from both sampling variability and structural ambiguity due to selective observation. This perspective aligns with practical decision-making needs in moderation systems, where understanding the stability of conclusions across plausible data-generating processes is often more informative than optimizing a single metric under fragile assumptions [14].

Finally, there are limits to what can be inferred from selectively observed and noisy data without strong assumptions or randomization. Without access to outcomes for low-risk or unreviewed cases, certain quantities, such as true prevalence rates or global error metrics, may be unidentifiable. Acknowledging these limits is an important step toward responsible modeling and evaluation. It motivates careful system design choices, transparent reporting of assumptions, and, where feasible, the incorporation of randomized audits or controlled data collection strategies to improve identifiability and long-term system reliability [15, 27].

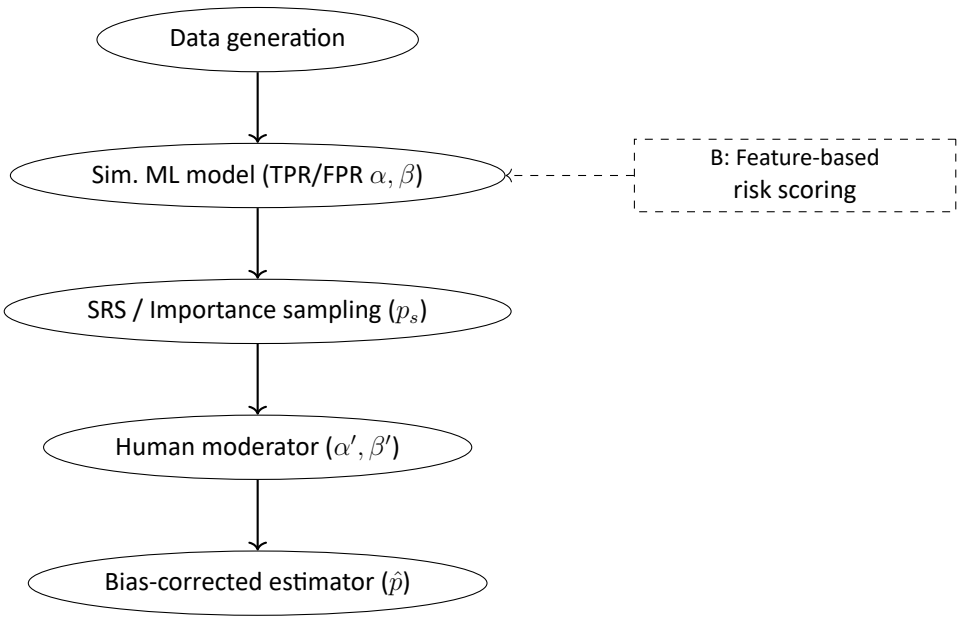
# 2 Problem setup and notation

This section formalizes the problem of estimating true prevalence under biased sampling and imperfect classification, introducing the notation and assumptions that will be used throughout the remainder of the thesis.

## 2.1 Conceptual overview of the estimation pipeline

1 figure. provides a schematic overview of the prevalence estimation pipeline used throughout this thesis. The process begins with data generation, where a synthetic population with a specified true prevalence is constructed. A simulated machine learning model then assigns predictions or risk scores to each item, characterized by fixed true positive and false positive rates $(\alpha, \beta)$. Based on these outputs, items are selected for review either through simple random sampling or importance sampling with inclusion probabilities $p_s$. The sampled items are subsequently evaluated by a human moderator, whose decisions are subject to misclassification with rates $(\alpha', \beta')$. Finally, a bias-corrected estimator combines the observed moderator labels and sampling weights to produce an estimate of the underlying population prevalence $\hat{p}$.

Approach B follows the same pipeline structure but differs in the construction of the machine learning stage: instead of directly simulating predictions with fixed error rates, feature-based risk scores are generated and thresholded to achieve target operating characteristics. All downstream sampling, moderation, and correction steps remain unchanged.




***1 figure.*** *Pipeline flowchart.*

## 2.2 Notation and assumptions

Let

- $N$ be the total number of listings in the marketplace.
- $p$ be the **true** proportion of counterfeit listings (the quantity we aim to estimate).
- $\mathbf{x}_i$ be the feature vector describing listing $i$.
- $y_i \in \{0, 1\}$ be the **true** label for listing $i$, where $y_i = 1$ indicates a counterfeit listing.

A machine learning model $M$ predicts $\hat{y}_i = M(\mathbf{x}_i)$. Suppose the ML model has:

- **True Positive Rate (TPR)** $\alpha = P(\hat{y} = 1 \mid y = 1)$,
- **False Positive Rate (FPR)** $\beta = P(\hat{y} = 1 \mid y = 0)$.

A human moderator $H$ also makes errors, with:

- **Moderator TPR** $\alpha_H = P(\hat{y}_H = 1 \mid y = 1)$,
- **Moderator FPR** $\beta_H = P(\hat{y}_H = 1 \mid y = 0)$.

Because the overall prevalence $p$ is small, simple random sampling typically yields very few positive cases, resulting in high estimator variance. To improve efficiency, many online platforms therefore employ targeted or risk-based sampling strategies that oversample items deemed likely to be positive. While such designs substantially increase the effective sample size of rare events, they induce selection bias that must be explicitly accounted for. In addition, both machine learning predictions and human moderation decisions are subject to misclassification. Consequently, valid prevalence estimation requires correction for both selective sampling and labeling errors. The construction of the resulting bias-corrected estimator is described in the 3 section.

# 3 Methodology

This section presents a step-by-step simulation that generates a synthetic population, simulates the outputs of both the machine learning model and human moderator decisions, and applies an estimator to approximate the prevalence *p*

## 3.1 Simulation steps

**Step 1: Data Generation.**

1. Generate $N$ feature vectors $\{\mathbf{x}_i\}_{i=1}^{N}$ from some distribution (e.g., multivariate normal).
2. Assign true labels $y_i \sim$ Bernoulli$(\sigma(\mathbf{w}^\top \mathbf{x}_i))$, where $\sigma(\cdot)$ is the logistic function, and **w** is calibrated so that the marginal proportion of $y = 1$ is near the desired $p$.

**Step 2: ML Model Generation.** We have an ML model with controlled True Positive Rate $\alpha$ and False Positive Rate $\beta$. The approach is described as:

**Approach A: Direct Simulation of Predictions.**

1. For each listing $i$ with true label $y_i$:
   - If $y_i = 1$ (counterfeit), set $\hat{y}_i = 1$ with probability $\alpha$, else 0.
   - If $y_i = 0$ (genuine), set $\hat{y}_i = 1$ with probability $\beta$, else 0.
2. This yields an exact TPR $= \alpha$ and FPR $= \beta$, but ignores $\mathbf{x}_i$.

**Approach B: Feature-Based Risk Scoring and Thresholding.**

1. For each listing $i$ with feature vector $\mathbf{x}_i$, compute a risk score

$$r_i = g(\mathbf{x}_i),$$

   where $g(\cdot)$ is a predefined scoring function.

2. Map the score $r_i$ to discrete risk points.
3. Choose a threshold $\tau$ and flag the listing if

$$\hat{y}_i = \begin{cases} 1, & r_i \geq \tau, \\ 0, & r_i < \tau. \end{cases}$$

4. This yields TPR and FPR that depend on both the score quality and the choice of $\tau$, and explicitly uses $\mathbf{x}_i$.

**Step 3: Importance Sampling.** Given the ML model outputs (either $\hat{y}_i$ or a continuous $\hat{p}_i$):

- Define a sampling probability $p_s(\hat{y}_i)$ or $p_s(\hat{p}_i)$ that *oversamples* items likely to be counterfeit.
- For each listing $i$, sample it with probability $p_s(\hat{y}_i)$ (or $\hat{p}_i$) so that you form a labeled set $S$ reviewed by the moderator.

**Step 4: Human Moderator Confusion Matrix.** After sampling, each selected listing $i$ is presented to the moderator. The moderator's label $\hat{y}_{H,i}$ is generated as:

- If $y_i = 1$: $\hat{y}_{H,i} = 1$ with probability $\alpha_H$, else 0.
- If $y_i = 0$: $\hat{y}_{H,i} = 1$ with probability $\beta_H$, else 0.

This simulates human misclassification. Optional complexities might include moderator fatigue or dependence on the ML score, but a constant $(\alpha_H, \beta_H)$ is sufficient for a first experiment.

**Step 5: Estimation.** The objective is to estimate the true population prevalence $p$ while accounting for both selective sampling and moderator misclassification. Let $p_s(i)$ denote the inclusion probability of item $i$, and define the corresponding design weight as $w_i = 1/p_s(i)$ for sampled items. To correct for systematic errors in human moderation, observed moderator labels are adjusted using the moderator confusion rates $(\alpha_H, \beta_H)$. A design-based, bias-corrected estimator of prevalence can then be written as:

$$\hat{p} \;=\; \frac{1}{N} \sum_{i \in S} w_i \left( \frac{\hat{y}_{H,i} - \beta_H}{\alpha_H - \beta_H} \right),$$

where $S$ denotes the set of sampled items and $\hat{y}_{H,i}$ is the moderator-assigned label. This estimator can be viewed as an inverse-probability-weighted estimator with an additional correction for label misclassification. The structure of this estimator follows classical design-based estimation principles [13] combined with standard corrections for outcome misclassification [22].

## 3.2 Variance Decomposition of the Prevalence Estimator

The uncertainty of the prevalence estimator arises from multiple stochastic components in the moderation pipeline, including population sampling, machine learning misclassification, and human moderator errors. To better understand the contribution of each component, we decompose the variance of the estimator into interpretable sources using a simulation-based variance decomposition framework.

Let $p \in (0,1)$ denote the true prevalence and $\hat{p}$ the final bias-corrected estimator. The estimator depends on several stochastic mechanisms and can be written abstractly as

$$\hat{p} = g(\mathcal{S}, \hat{y}, \hat{y}_H),$$

where $\mathcal{S}$ denotes the sampled set, $\hat{y}$ the machine learning predictions, and $\hat{y}_H$ the human moderator labels.

The total variance of the estimator is defined as

$$\mathsf{Var}(\hat{p}) = \mathbb{E}\big[(\hat{p} - \mathbb{E}[\hat{p}])^2\big] ,$$

where the expectation is taken over all sources of randomness in the pipeline.

To attribute uncertainty to individual stages, we apply the law of total variance [6]. Let $\mathcal{S}$ denote the sampling mechanism, $\mathcal{M}$ the machine learning prediction process, and $\mathcal{H}$ the human moderation process. Then,

$$\mathsf{Var}(\hat{p}) = \mathbb{E}_{\mathcal{S}}[\mathsf{Var}(\hat{p} \mid \mathcal{S})] + \mathsf{Var}_{\mathcal{S}}(\mathbb{E}[\hat{p} \mid \mathcal{S}]) .$$

Building on this identity, we decompose the total variance into additive components:

$$\mathsf{Var}(\hat{p}) = V_{\mathsf{sampling}} + V_{\mathsf{ML}} + V_{\mathsf{human}} + V_{\mathsf{interaction}},$$

where

$$V_{\mathsf{sampling}} = \mathsf{Var}_{\mathcal{S}}(\mathbb{E}[\hat{p} \mid \mathcal{S}]) ,$$

$$V_{\mathsf{ML}} = \mathbb{E}_{\mathcal{S}}[\mathsf{Var}_{\mathcal{M}}(\hat{p} \mid \mathcal{S})] ,$$

$$V_{\mathsf{human}} = \mathbb{E}_{\mathcal{S},\mathcal{M}}[\mathsf{Var}_{\mathcal{H}}(\hat{p} \mid \mathcal{S}, \mathcal{M})] .$$

The interaction term is defined residually as

$$V_{\mathsf{interaction}} = \mathsf{Var}(\hat{p}) - (V_{\mathsf{sampling}} + V_{\mathsf{ML}} + V_{\mathsf{human}}) ,$$

and captures non-additive effects arising from dependencies between the individual stages of the pipeline. This form of component-wise attribution is conceptually related to variance component decompositions used in experimental design and hierarchical modeling [8].

Closed-form expressions for the above variance components are unavailable due to nonlinear sampling schemes and bias correction. We therefore estimate each component using Monte Carlo simulation [30]. For a fixed configuration, we repeatedly resample one stochastic component at a time while holding the others fixed, and estimate conditional variances empirically as

$$\widehat{\mathsf{Var}}(X) = \frac{1}{R-1} \sum_{r=1}^{R} \big(\hat{p}^{(r)} - \bar{p}\big)^2 , \qquad \bar{p} = \frac{1}{R} \sum_{r=1}^{R} \hat{p}^{(r)} .$$

This decomposition enables attribution of estimator uncertainty to individual stages of the moderation pipeline and facilitates evaluation of how improvements in machine learning accuracy, moderator reliability, or sampling design affect total estimation variance.

As a practical illustration, variance decomposition is implemented by estimating the total variance of the estimator through repeated simulation and attributing variance contributions to individual stochastic components, an approach commonly used in variance-based sensitivity analysis and recent stochastic model decomposition studies [5, 29].

# 4 Simulation Implementation

The simulation framework was implemented in Python using standard scientific computing libraries for numerical computation, data manipulation, and visualization. The pipeline is modular-like and designed to reflect the whole process of estimating the prevalence of rare harmful events in the presence of imperfect machine learning predictions and noisy human moderation. Detailed parameter selection and configuration of simulations can be found in Appendix B.

## 4.1 A and A1 - sampling approaches description

Approach *A* (simple random sampling) draws listings uniformly at random from the full population, independent of any model-based risk signal. Each listing therefore has the same inclusion probability, and the resulting sample directly reflects the true class distribution. This approach is straightforward to implement and unbiased by construction, but becomes inefficient when the target prevalence is low, as only a small number of positive listings are observed.

Approach *A1* (risk-weighted sampling) modifies the sampling step by assigning higher inclusion probabilities to listings with elevated predicted risk, as indicated by an upstream scoring model. In practice, this is implemented by stratifying or weighting listings according to their risk score and oversampling high-risk strata. While this substantially increases the number of positives observed for a fixed labeling budget and reduces estimator variance, it introduces a sampling bias that must be explicitly corrected in the prevalence estimator. Importantly, the final prevalence estimate depends on both the sampling design and the error characteristics of the risk model, which are accounted for in the estimation stage described in Section X.

## 4.2 B - Realistic features and logistic thresholding

To guide selective data collection mentioned in approach B, we employ a logistic regression model to assign each unit a continuous risk score reflecting the probability of a positive outcome. Let $X_i \in \mathbb{R}^5$ denote the feature vector for unit $i$, and let $y_i \in \{0,1\}$ denote the true latent label. The data-generating process assumes

$$\Pr(y_i = 1 \mid X_i) = \sigma(b_0 + r_i),$$

where $\sigma(\cdot)$ is the logistic link function and $r_i$ is a linear risk index constructed from the features. The risk index $r_i$ is constructed as a weighted combination of the five features listed in Table 1 table., with higher values corresponding to greater likelihood of a positive outcome. The intercept $b_0$ is obtained via a numerical root-finding procedure to ensure that $\frac{1}{N}\sum_i \Pr(y_i = 1 \mid X_i)$ matches the desired prevalence level.

A logistic regression model is fitted to the generated data, yielding estimated scores $\hat{s}_i = \Pr(\hat{y}_i = 1 \mid X_i)$. To align the classifier with a desired operating point, a decision threshold $t$ is selected by minimizing the squared deviation between the empirical true positive rate and false positive rate

and their respective targets $(\alpha, \beta)$. Specifically, the threshold is chosen as

$$t = \arg\min_{u} \left\{ \big(\mathsf{TPR}(u) - \alpha\big)^2 + \big(\mathsf{FPR}(u) - \beta\big)^2 \right\},$$

where $\mathsf{TPR}(u)$ and $\mathsf{FPR}(u)$ denote the true and false positive rates induced by threshold $u$. This calibration procedure corresponds to standard operating-point selection in statistical classification, where decision thresholds are tuned to achieve desired trade-offs between true and false positive rates [2, 12]

Rather than using this threshold deterministically, the resulting risk scores are used to construct a probabilistic sampling design. Each unit $i$ is assigned an inclusion probability

$$\pi_i \propto \hat{s}_i^{\gamma},$$

where $\gamma > 0$ controls the aggressiveness of risk-based targeting. The probabilities are rescaled to achieve a desired average sampling fraction and truncated to lie within fixed bounds to avoid degenerate inclusion probabilities. Sampling is then carried out independently across units using Bernoulli draws with probabilities $\pi_i$. Such score-dependent inclusion probabilities define a model-assisted sampling design, in which auxiliary risk information is used to guide unequal-probability sampling in order to improve efficiency while retaining known inclusion probabilities for design-based correction [7, 33].

This design preserves randomization while preferentially sampling high-risk units, enabling variance reduction relative to simple random sampling while retaining known inclusion probabilities for subsequent design-based correction.

## 4.3 Simulated features

The feature set in 1 table. is designed to reflect commonly used signals in real-world content moderation and fraud detection systems across online marketplaces and social media platforms. Access to detailed, individual-level platform data is typically restricted due to privacy concerns, commercial sensitivity, and proprietary risk-scoring practices. Consequently, we construct a synthetic feature space that captures the qualitative structure of signals frequently employed in practice, rather than attempting to replicate any specific platform's internal models. The selected features represent generic behavioral and content related indicators, such as account maturity, posting intensity, link usage, textual templating, and content duplication, that are widely applicable across domains. This abstraction allows the simulation framework to remain broadly relevant while avoiding reliance on sensitive or unavailable real-world data.

***1 table.*** *Simulated marketplace-style features used in the prevalence simulation pipeline.*

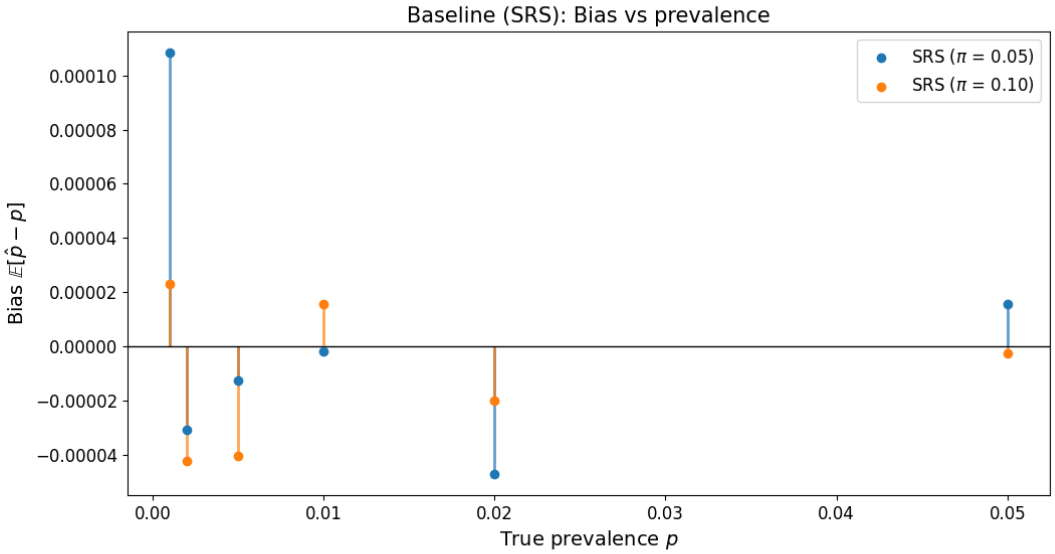
| Feature | Meaning in marketplace/social-media |
|---|---|
| $f_1$: `account_age_norm` | How "old" the seller/poster account is |
| $f_2$: `activity_burstiness` | How many items listed/posted recently |
| $f_3$: `link_intensity` | Fraction of descriptions with external links / link density |
| $f_4$: `text_effort_low` | Very short or templated item descriptions |
| $f_5$: `duplication_score` | Item looks like many other listings (same text/images) |

# 5 Results

The empirical results are illustrative case studies, demonstrating the behavior and limitations of the proposed framework, rather than an exhaustive evaluation of all model configurations. Due to many possible variations of parameter configurations of the model, no finite set of empirical results can be fully representative of its behavior. The results presented here therefore focus on reasonable configurations that highlight the interpretative and computational properties of the framework.

## 5.1 Baseline model estimated bias

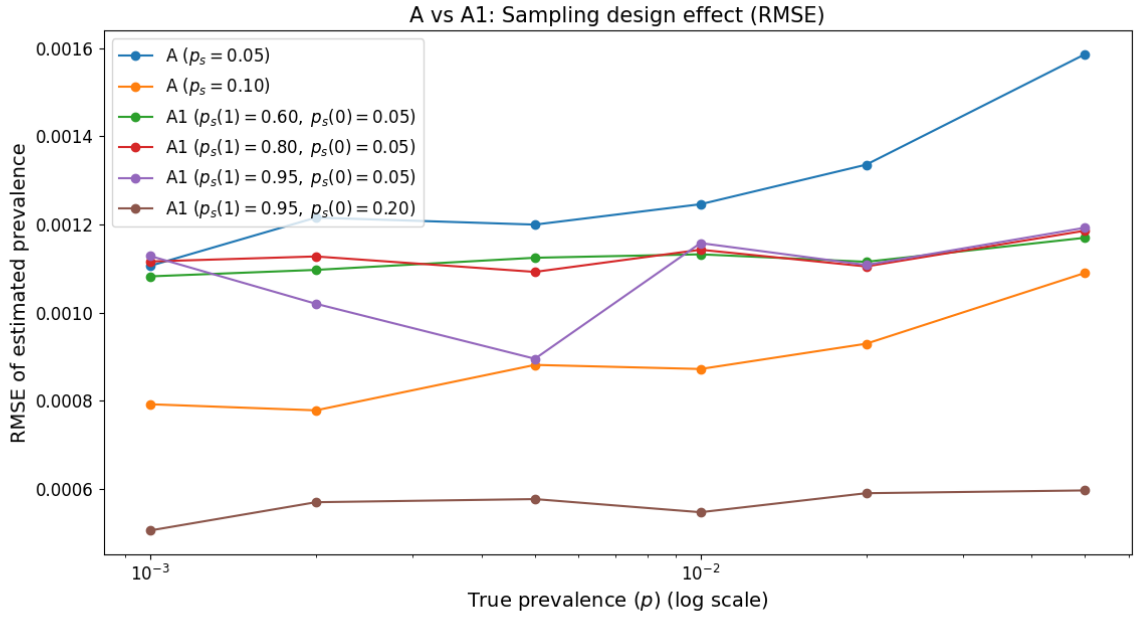
Results presented in 2 figure. show the estimated bias for the baseline model parameter configuration (ref config!) under simple random sampling across a range of true prevalence levels and sampling fractions. Under simple random sampling (SRS), the baseline estimator shows evidence of negligible bias throughout the considered prevalence range. For both sampling fractions (π=0.05 and π=0.10), the empirical bias remains close to zero, with no systematic monotonic relationship with the true prevalence *p*. The observed fluctuations around zero are small in magnitude and comparable across sampling fractions, suggesting that they primarily reflect Monte Carlo simulation variability rather than structural bias. As expected under SRS, increasing the sampling fraction reduces bias variability but does not change the bias pattern, reinforcing that the estimator is approximately unbiased in the absence of selective sampling mechanisms. This baseline therefore provides a suitable reference point for assessing bias introduced by more informative or adaptive sampling designs in subsequent sections.




***2 figure.*** *Bias vs Prevalence under simple random sampling*

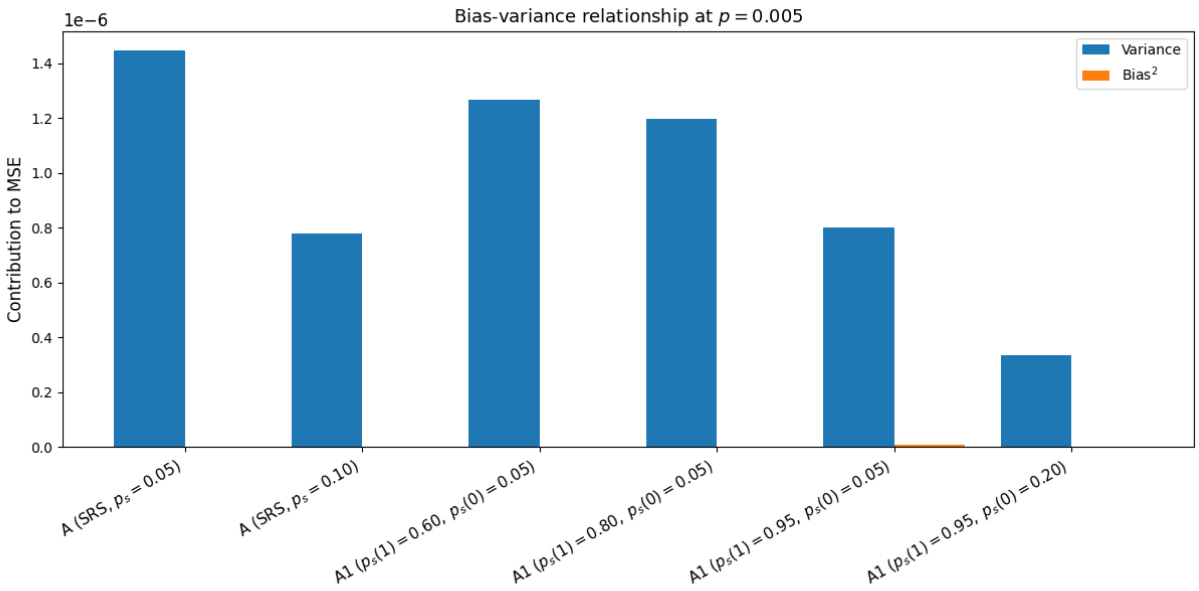
## 5.2 Bias-variance relationship by sampling design

3 figure. compares the root mean squared error (RMSE) of the prevalence estimator under the baseline design *A* and a set of informative sampling designs denoted by *A1* across true prevalence levels. Under the baseline design, RMSE generally increases as prevalence rises, reflecting the combined effect of estimator variance and finite-sample variability, with lower sampling fractions exhibiting consistently higher RMSE. In contrast, the informative sampling designs reduce RMSE across all prevalence levels for *A* specification with lower sampling rate ($ps$=0.05), however, do not show smaller RMSE for the higher sampling rate variant. The magnitude of RMSE reduction depends on the degree of selectivity in the design: higher positive selection probabilities $p_s(1)$ and increased negative sampling rates $p_s(0)$ are associated with markedly lower RMSE, particularly in the rare-event regime. Notably, one *A1* configuration ($p_s(1)$=0.95, $p_s(0)$=0.2) achieves lower RMSE than the baseline specification even when the latter uses a higher overall sampling rate, highlighting the efficiency gains attainable through targeted sampling. These results partly illustrate the trade-off introduced by informative sampling designs, where reductions in variance, and hence RMSE, can be achieved at the cost of potential bias, a phenomenon examined in more detail in the subsequent bias analysis.




***3 figure.*** *Sampling design effect on RMSE*

4 figure. illustrates the decomposition of mean squared error into variance and squared bias components at prevalence set to half a percent. Across all designs, MSE is overwhelmingly driven by variance, with the squared bias contribution remaining negligible even under highly selective sampling schemes. Consequently, the efficiency gains of informative designs in this setting arise primarily from variance reduction rather than from a meaningful bias–variance trade-off.



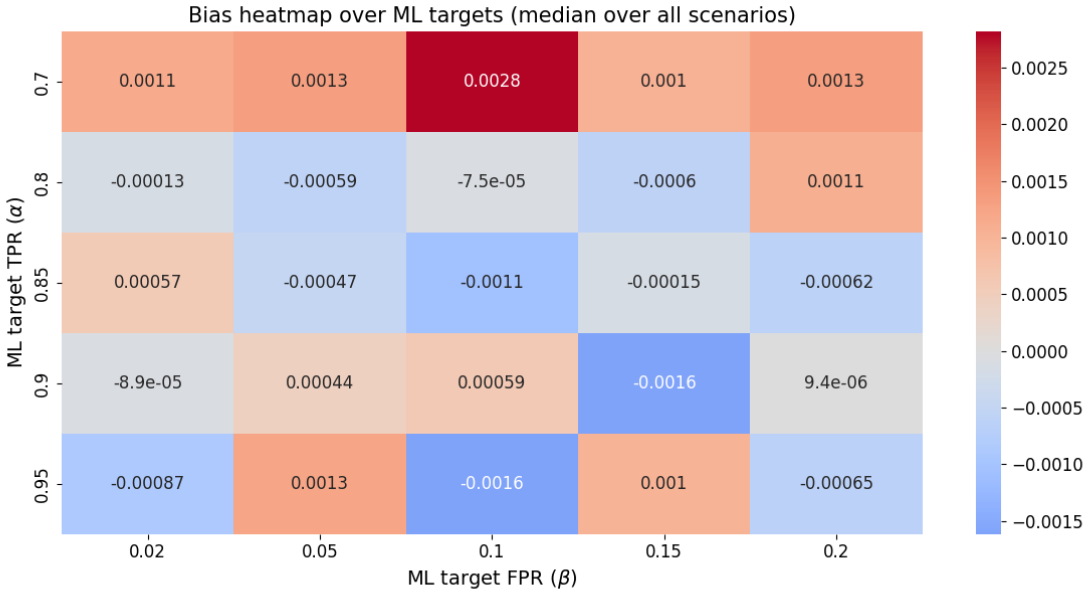

***4 figure.*** *Bias and Variance relationship*

## 5.3 Results for Approach B (Feature-based targeting)

This section presents results for Approach B, which incorporates feature-based risk scoring and probabilistic targeting into the prevalence estimation pipeline. Unlike earlier analyses based on fixed error rates, this approach generates machine learning predictions from a realistic feature model, allowing ML target configurations to be studied in a more applied setting. The results focus on two aspects: systematic bias patterns across ML target choices and the resulting estimation uncertainty. Bias behavior is examined using heatmaps to identify structural dependencies, while uncertainty is assessed through confidence interval width under varying moderator accuracy.

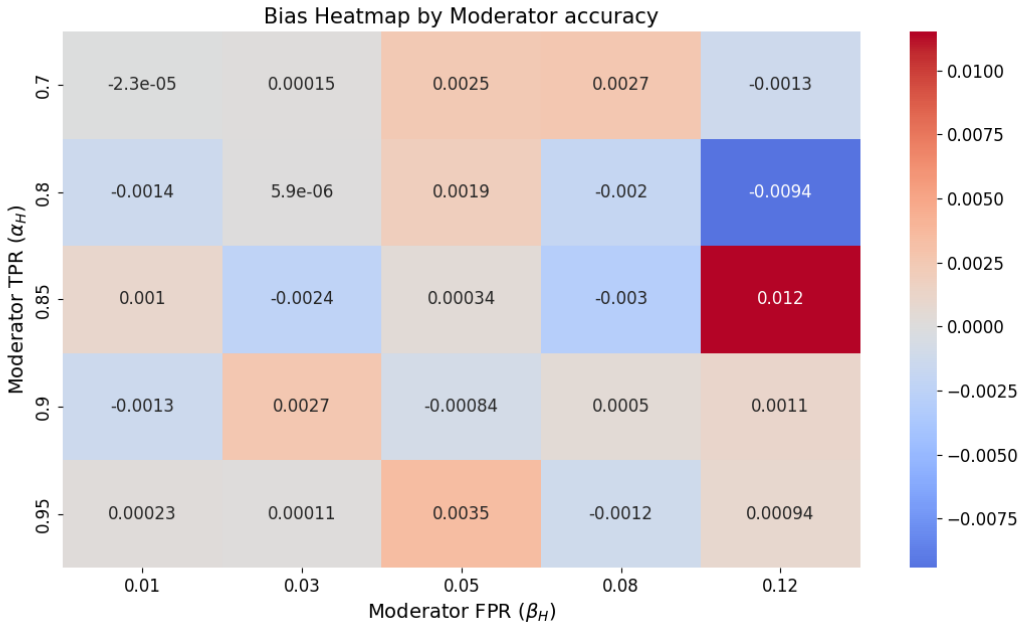
### 5.3.1 Bias patterns across configurations

5 figure. presents the median bias across all scenarios as a function of ML target rate. Overall, bias remains small in magnitude and exhibits no monotonic dependence on either target TPR or FPR. This suggests that, when averaged over sampling designs and moderator configurations, ML target choices do not systematically dominate estimation bias. Instead, bias appears to arise from interactions with downstream components, with ML effects largely cancelling out in aggregate. Bias values fluctuate around zero, with both small positive and negative deviations, given their limited magnitude, part of the residual variation across ML target configurations may be attributable to finite Monte Carlo variability rather than systematic directional effects.




***5 figure.*** *Bias Heatmap over ML target rates*

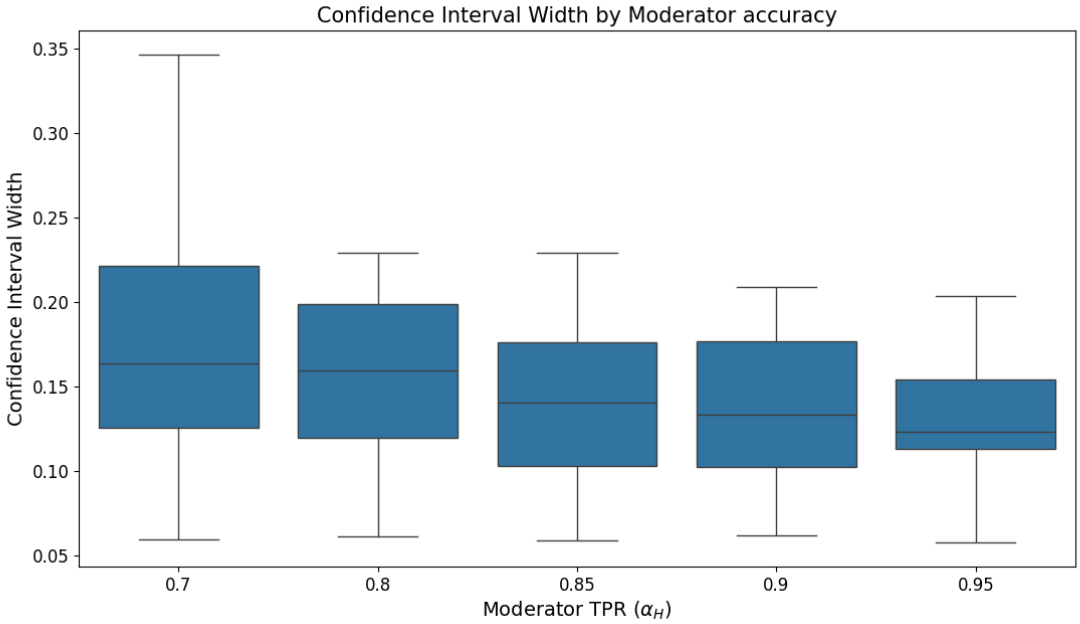
6 figure. shows bias as a function of moderator operating characteristics, with ML targets and sampling design held fixed. Bias remains generally small and fluctuates around zero, indicating no systematic overestimation or underestimation induced by the correction procedure. Larger deviations arise only under aggressive moderator regimes, reflecting amplification of correction errors. As in machine learning variation case, part of the residual variation may be attributable to finite Monte Carlo noise.




***6 figure.*** *Bias Heatmap over Moderator accuracy*

### 5.3.2 Estimation uncertainty and confidence interval width

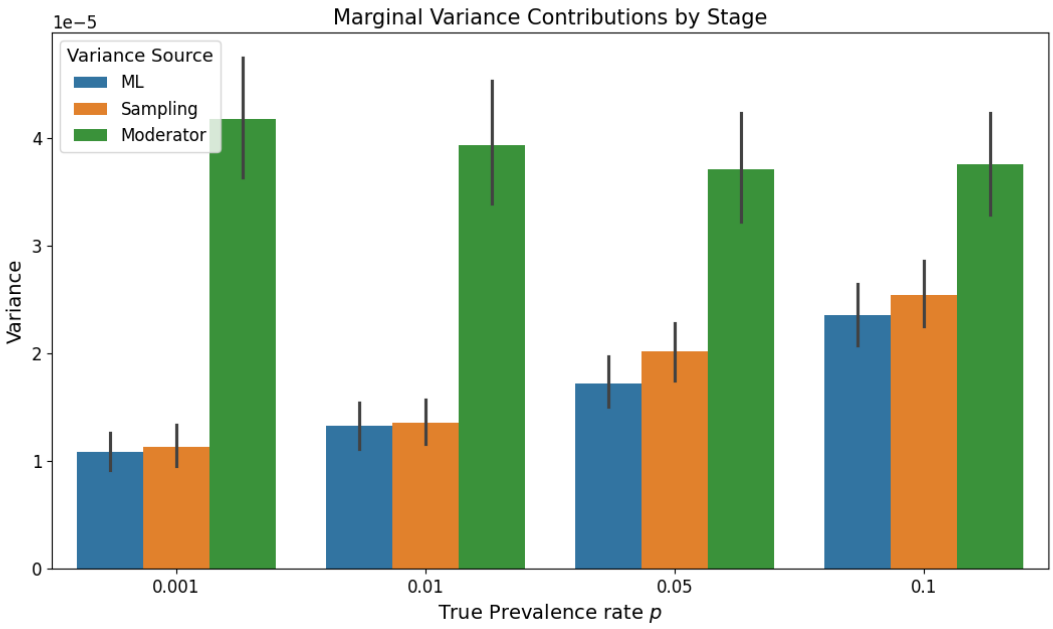
In 7 figure. we can see the distribution of confidence interval widths across Monte Carlo replications as a function of moderator true positive rate $\alpha_H$. As moderator accuracy increases, the median confidence interval width decreases monotonically, indicating a systematic reduction in estimator variance. This pattern reflects the role of moderator misclassification as a major source of uncertainty: lower values of $\alpha_H$ amplify variance through the bias-correction step, leading to wider and more dispersed confidence intervals. In contrast, higher moderator accuracy substantially stabilizes the estimator, producing both narrower intervals and reduced variability across replications. Occasional extreme interval widths observed at lower $\alpha_H$ values are attributable to finite-sample variability amplified by misclassification correction, rather than systematic estimation failure. Overall, the results show that moderator accuracy plays a central role in determining the reliability of prevalence estimates and their associated uncertainty.




***7 figure.*** *Confidence Interval Width By Moderator Accuracy*

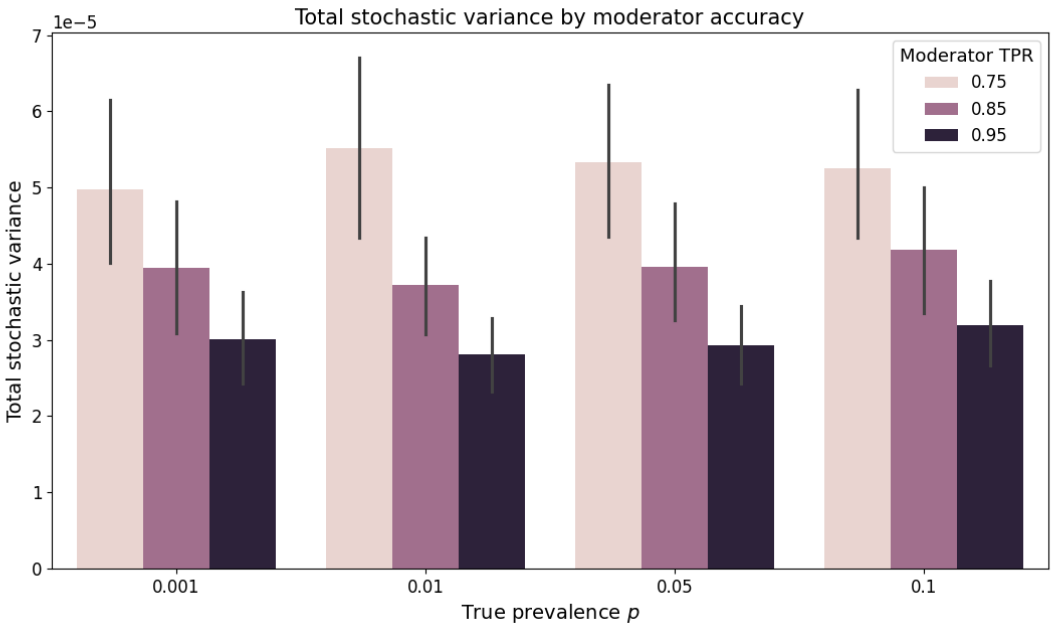
## 5.4 Prevalence estimation variance decomposition

Pattern shown in 8 figure. represents decomposition for stochastic variance into contributions from the ML scoring stage, sampling stage, and correction stage. Across all prevalence levels, variance is dominated by the correction stage, with sampling contributing more than ML stochasticity. This indicates that uncertainty in prevalence estimation is driven primarily by reweighting and correction mechanisms rather than classifier noise, explaining the limited sensitivity of confidence interval widths to ML target performance. For interpretability, multiplicative interaction terms between stages were not presented in the figure.




***8 figure.*** *Marginal Stochastic Variance Contributions by stage*

9 figure. shows that total stochastic variance remains relatively stable across a wide range of true prevalence values, while being strongly driven by moderator accuracy. Improvements in moderator sensitivity substantially reduce variance at all prevalence levels, whereas changes in prevalence have a comparatively minor effect. Within the considered range of prevalence rates, total stochastic variance is more sensitive to moderator accuracy than to changes in true prevalence.




***9 figure.*** *Total Stochastic variance by Moderator Accuracy*

# 6 Discussion

This section discusses the ramifications of the simulation results for prevalence estimation under selective sampling and imperfect classification, with particular emphasis on how sampling design, model accuracy, and moderation interact in rare-event settings.

## 6.1 Limitations and considerations

While the simulation framework provides useful insights into the behavior of prevalence estimation under imperfect classification and moderation, several limitations should be acknowledged.

The simulation uses fixed true positive and false positive rates (TPR/FPR) for both the ML model and human moderator. In real-world settings, these rates may vary over time, across content types, or even among different moderators. The assumption of stationarity may limit applicability in more dynamic environments.

An important practical implication of these results concerns the interaction between selective sampling and fixed review budgets. While targeted sampling can greatly reduce estimator variance in rare-event settings, it does not remove constraint imposed by limited human review capacity. When the true prevalence is extremely low, even aggressive targeting may result in only a small number of reviewed positive cases if the review budget is fixed. Consequently, the effective sample size available for prevalence estimation may remain limited, regardless of the size of the underlying population. This indicates that selective sampling simply reallocates review effort rather than increasing it, and that its benefits are bounded by the available moderation budget. In large-scale platforms, where population sizes far exceed feasible review volumes, the review budget can become the dominant bottleneck for inference rather than the sampling design alone.

A related consideration arises in regimes combining both very small populations and extremely low prevalence. In such settings, highly selective sampling criteria may result in no units being sampled at all, particularly when inclusion probabilities are strongly concentrated on high-risk scores. While this represents an extreme case, it highlights an additional practical limitation of aggressive targeting strategies: under certain conditions, selectivity itself can preclude data collection. This reinforces the need to balance targeting strength against coverage, especially in low-volume or early-stage systems.

## 6.2 Future work

The prevalence estimation framework studied in this thesis is inherently modular, consisting of distinct components for sampling, classification, moderation, and estimation. Each component can be adapted or extended to serve different operational goals, such as improving precision, reducing review costs, or accommodating alternative data constraints. This modular structure naturally motivates several directions for future work, in which individual components of the pipeline could be modified or optimized without altering the overall framework.

Incorporating optimality seeking complication, such as costs per reviewed sample or moderator intervention, would add an important dimension to the framework. Modeling utility functions, for example, maximizing estimation accuracy subject to fixed review budgets, would allow the pipeline to be evaluated under realistic cost constraints. Such extensions would help assess the applicability of the framework in real-world, dynamic settings where cost efficiency is a central concern.

The experimental design in this thesis focuses primarily on parameter regimes that are feasible and representative of moderate to well-performing systems. This choice was made to emphasize interpretable and practically relevant behavior rather than extreme or edge cases. Nevertheless, more extensive stress testing, such as exploring highly uninformative classifiers, extreme error rates, or adversarial parameter combinations could provide additional insight into the dynamic interactions between components of the estimation pipeline. Examining such edge cases may help identify regime boundaries where the estimator becomes unstable or where specific components dominate overall performance, offering a better understanding of pipeline behavior under adverse conditions.

An additional limitation of analysis with simulated features is the assumption that the true prevalence is known by construction. In real-world applications, the true prevalence of harmful or low-quality content is rarely observable and must itself be inferred or approximated. A valuable direction for future work would therefore be to apply the proposed framework to real datasets where prevalence is known or can be reliably estimated for specific events or quality attributes. Comparing bias and uncertainty in such empirical settings to the simulated results would provide an important validation of whether the favorable bias properties observed in simulation persist under realistic data-generating conditions.

# 7 Conclusions

The simulation results indicate that prevalence estimation under imperfect detection is governed by interactions across the full detection, sampling, correction pipeline rather than by any single component in isolation. Across a wide range of operating regimes, estimation bias remains generally small and fluctuates around zero, with both over- and underestimation occurring depending on the configuration. No monotonic dependence of bias on machine learning target performance is observed, suggesting that classifier tuning alone does not dominate estimation accuracy once downstream correction is applied. In contrast, uncertainty measured by stochastic variance and confidence interval width is more strongly influenced by moderator accuracy and correction behavior, with evidence of stabilizing variance that limits gains from upstream improvements. These findings reflect a bias–variance trade-off: aggressive detection or moderation can reduce variance through increased effective sample size but may introduce bias, while more conservative regimes stabilize bias at the cost of higher uncertainty. Overall, the results emphasize the importance of analyzing prevalence estimation as an integrated system and motivate careful design choices that balance accuracy, reliability, and practical constraints.

# References and sources


[1] J. Andreoni, B. Erard, J. Feinstein. “Tax Compliance.” In: *Journal of Economic Literature* 36.2 (1998), pages 818–860.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. ISBN: 978-0-387-31073-2.

[3] L. Breiman. “Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author).” In: *Statistical Science* 16.3 (2001), pages 199–231. https://doi.org/10.1214/ss/1009213726. URL: https://doi.org/10.1214/ss/1009213726.

[4] N. E. Breslow, N. E. Day. *Statistical Methods in Cancer Research, Volume I: The Analysis of Case-Control Studies*. Lyon: International Agency for Research on Cancer, 1980.

[5] Á. Carmona-Cabrero, R. Muñoz-Carpena, W. S. Oh, R. Muneepeerakul. “Decomposing Variance Decomposition for Stochastic Models: Application to a Proof-Of-Concept Human Migration Agent-Based Model.” In: *Journal of Artificial Societies and Social Simulation* 27.1 (2024), page 16. https://doi.org/10.18564/jasss.5174. URL: https://www.jasss.org/27/1/16.html.

[6] G. Casella, R. L. Berger. *Statistical Inference*. 2nd. Duxbury Press, 2002.

[7] W. G. Cochran. *Sampling Techniques*. 3rd. New York: John Wiley & Sons, 1977. ISBN: 978-0-471-16240-7.

[8] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, D. B. Rubin. *Bayesian Data Analysis*. 3rd. Chapman and Hall/CRC, 2013.

[9] C. Guo, G. Pleiss, Y. Sun, K. Q. Weinberger. “On Calibration of Modern Neural Networks.” In: *Proceedings of the 34th International Conference on Machine Learning*. Edited by D. Precup, Y. W. Teh. Volume 70. Proceedings of Machine Learning Research. PMLR, 2017, pages 1321–1330. URL: https://proceedings.mlr.press/v70/guo17a.html.

[10] D. J. Hand, W. E. Henley. “Statistical Classification Methods in Consumer Credit Scoring: A Review.” In: *Journal of the Royal Statistical Society Series A: Statistics in Society* 160.3 (2007), pages 523–541. ISSN: 0964-1998. https://doi.org/10.1111/j.1467-985X.1997.00078.x. URL: https://doi.org/10.1111/j.1467-985X.1997.00078.x.

[11] M. H. Hansen, W. N. Hurwitz. “On the Theory of Sampling from Finite Populations.” In: *The Annals of Mathematical Statistics* 14.4 (1943), pages 333–362. https://doi.org/10.1214/aoms/1177731356.

[12] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd edition. New York: Springer, 2009. ISBN: 978-0-387-84857-0.

[13] D. G. Horvitz, D. J. Thompson. “A Generalization of Sampling Without Replacement from a Finite Universe.” In: *Journal of the American Statistical Association* 47.260 (1952), pages 663–685. https://doi.org/10.1080/01621459.1952.10483446.

[14] G. W. Imbens, D. B. Rubin. *Causal Inference for Statistics, Social, and Biomedical Sciences*. Cambridge, UK: Cambridge University Press, 2015. ISBN: 9780521885881.

[15] H. Jiang, B. Kim, M. Y. Guan, M. Gupta. *To Trust Or Not To Trust A Classifier*. 2018. URL: https://arxiv.org/abs/1805.11783.

[16] J. Kleinberg, H. Lakkaraju, J. Leskovec, J. Ludwig, S. Mullainathan. "Human Decisions and Machine Predictions." In: *The Quarterly Journal of Economics* 133.1 (2018), pages 237–293. https://doi.org/10.1093/qje/qjx032.

[17] J. Kleinberg, J. Ludwig, S. Mullainathan, Z. Obermeyer. "Prediction Policy Problems." In: *American Economic Review* 105.5 (2015), pages 491–95. https://doi.org/10.1257/aer.p20151023. URL: https://www.aeaweb.org/articles?id=10.1257/aer.p20151023.

[18] V. Koshy, F. Choi, Y.-S. Chiang, H. Sundaram, E. Chandrasekharan, K. Karahalios. "Venire: A Machine Learning-Guided Panel Review System for Community Content Moderation." In: (2024).

[19] V. Lai, C. Tan, S. L. Smith. "Human-in-the-Loop Machine Learning: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 34.10 (2022), pages 4734–4750. https://doi.org/10.1109/TKDE.2021.3065567.

[20] H. Lakkaraju, J. Kleinberg, J. Leskovec, J. Ludwig, S. Mullainathan. "The Selective Labels Problem: Evaluating Algorithmic Predictions in the Presence of Unobservables." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pages 275–284. ISBN: 9781450348874. https://doi.org/10.1145/3097983.3098066. URL: https://doi.org/10.1145/3097983.3098066.

[21] S. Lessmann, B. Baesens, H.-V. Seow, L. C. Thomas. "Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research." In: *European Journal of Operational Research* 247.1 (2015), pages 124–136. ISSN: 0377-2217. https://doi.org/https://doi.org/10.1016/j.ejor.2015.05.030. URL: https://www.sciencedirect.com/science/article/pii/S0377221715004208.

[22] R. J. A. Little, D. B. Rubin. *Statistical Analysis with Missing Data*. 3rd edition. Hoboken, NJ: Wiley, 2019. ISBN: 9781119482260.

[23] Y. Liu. *Estimating Prevalence of Rare Events*. 2019. URL: https://www.unofficialgoogledatascience.com/2019/08/estimating-prevalence-of-rare-events.html.

[24] C. F. Manski. *Partial Identification of Probability Distributions*. New York, NY: Springer, 2003. ISBN: 9780387009872.

[25] A. Niculescu-Mizil, R. Caruana. "Predicting Good Probabilities with Supervised Learning." In: *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*. Bonn, Germany: ACM, 2005, pages 625–632. https://doi.org/10.1145/1102351.1102430.

[26] OpenAI. *ChatGPT*. 2025. URL: https://www.openai.com/.

[27] J. Pearl. *Causality: Models, Reasoning, and Inference*. 2nd edition. Cambridge, UK: Cambridge University Press, 2009. ISBN: 9780521895606.

[28] R. L. Prentice, R. Pyke. “Logistic Disease Incidence Models and Case-Control Studies.” In: *Biometrika* 66.3 (1979), pages 403–411. https://doi.org/10.1093/biomet/66.3.403.

[29] O. T. Räisä, A. Honkela. “A Bias–Variance Decomposition for Ensembles over Multiple Synthetic Datasets.” In: *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Volume 258. Proceedings of Machine Learning Research. Journal of Machine Learning Research, 2025, pages –. URL: https://arxiv.org/abs/2402.03985.

[30] C. P. Robert, G. Casella. *Monte Carlo Statistical Methods*. 2nd. Springer, 2004.

[31] P. R. Rosenbaum. *Observational Studies*. New York, NY: Springer, 2002. ISBN: 9780387950051.

[32] C. Rudin. *Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead*. 2019. URL: https://arxiv.org/abs/1811.10154.

[33] C.-E. Särndal, B. Swensson, J. Wretman. *Model Assisted Survey Sampling*. New York: Springer, 1992. ISBN: 978-0-387-40620-6.

[34] X. Wang, U. Gadiraju, F. Hopfgartner, S.-H. Wu. “Human-in-the-Loop Machine Learning: A Systematic Review.” In: *IEEE Intelligent Systems* 36.4 (2021), pages 70–78. https://doi.org/10.1109/MIS.2021.3085760.

# A Use of AI-assisted tools

An AI-based language model (ChatGPT, OpenAI) was used as a supplementary tool during the preparation of the thesis. It was used to identify relevant literature for the review, improving language clarity and academic style, and troubleshooting minor programming and LaTeX issues [26].

# B Simulation Parameters and Configuration

The tables in this appendix section summarize the parameter values and grids used in the simulation experiments. They are provided to ensure transparency and reproducibility, while the main text focuses on interpreting aggregate patterns rather than individual parameter settings.

## B.1 Approach A, A1 parameters

***A.1*** *Global simulation parameters used across all Monte Carlo experiments*

| Component | Symbol | Value |
|---|---|---|
| Population size | $N$ | $1{,}000{,}000$ |
| Monte Carlo replications | $R$ | $200$ |
| Random seed offset | – | $123$ |
| Confidence level | – | $95\%$ |

***A.2*** *True prevalence values used in simulation experiments*

| Parameter | Values |
|---|---|
| True prevalence | $p \in \{0.001, 0.002, 0.005, 0.01, 0.02, 0.05\}$ |

***A.3*** *Machine learning and moderator error-rate parameters*

| Component | Parameter | Value |
|---|---|---|
| ML true positive rate | $\alpha$ | $0.85$ |
| ML false positive rate | $\beta$ | $0.10$ |
| Moderator true positive rate | $\alpha_H$ | $0.90$ |
| Moderator false positive rate | $\beta_H$ | $0.05$ |

***A.4*** *Sampling designs and inclusion probabilities used in experiments*

| Approach | Design type | Inclusion probabilities | Description |
|---|---|---|---|
| A | SRS | $p_s = 0.05$ | Low-rate simple random sampling |
| A | SRS | $p_s = 0.10$ | Higher-rate simple random sampling |
| A1 | Targeted | $p_s(1) = 0.60, p_s(0) = 0.05$ | Moderate targeting |
| A1 | Targeted | $p_s(1) = 0.80, p_s(0) = 0.05$ | Strong targeting |
| A1 | Targeted | $p_s(1) = 0.95, p_s(0) = 0.05$ | Very strong targeting |
| A1 | Targeted | $p_s(1) = 0.95, p_s(0) = 0.20$ | Strong targeting with higher baseline |

***A.5** Parameter settings used for bias–variance decomposition figures*

| Parameter | Symbol | Value |
|---|---|---|
| Fixed prevalence | $p$ | $0.005$ (primary), $0.01$ (robustness) |
| Error rates | $(\alpha,\beta,\alpha_H,\beta_H)$ | As in Table A.3 |
| Sampling designs | – | As in Table A.4 |

## B.2 Approach B parameters

***A.6** Global simulation parameters used across all experiments*

| Parameter | Symbol / Name | Value |
|---|---|---|
| Population size | $N$ | $10^5$ to $10^6$ |
| Random seed offset | – | $0$ (incremented by scenario) |
| Risk score exponent | $\gamma$ | $1.5$ |
| Sampling rate target | $\bar{p}_s$ | $0.10$ |
| Sampling probability bounds | $[p_s^{\min}, p_s^{\max}]$ | $[0.01, 0.80]$ |
| Confidence level | – | $95\%$ |

***A.7** Feature-based risk model used in Approach B*

| Feature | Description | Risk weight |
|---|---|---|
| $f_1$ | Account age (normalized) | $2.0 \times (1 - f_1)$ |
| $f_2$ | Activity burstiness | $2.5 \times f_2$ |
| $f_3$ | Link intensity | $2.0 \times f_3$ |
| $f_4$ | Low-effort text indicator | $1.5 \times f_4$ |
| $f_5$ | Content duplication score | $2.0 \times f_5$ |

***A.8** Parameter grid used in main simulation experiments*

| Component | Parameter | Values explored |
|---|---|---|
| True prevalence | $p$ | $\{0.001, 0.01, 0.05, 0.10\}$ |
| ML target TPR | $\alpha$ | $\{0.70, 0.85\}$ |
| ML target FPR | $\beta$ | $\{0.10, 0.20\}$ |
| Moderator TPR | $\alpha_H$ | $\{0.85, 0.95\}$ |
| Moderator FPR | $\beta_H$ | $\{0.05, 0.10\}$ |

***A.9*** *Parameter settings used for bias heatmap analyses*

| Parameter | Symbol | Values |
|---|---|---|
| True prevalence | $p$ | $\{0.001, 0.01\}$ |
| ML target TPR | $\alpha$ | $\{0.70, 0.80, 0.85, 0.90, 0.95\}$ |
| ML target FPR | $\beta$ | $\{0.02, 0.05, 0.10, 0.15, 0.20\}$ |
| Moderator TPR | $\alpha_H$ | $0.90$ |
| Moderator FPR | $\beta_H$ | $0.05$ |

## B.3 Variance decomposition section parameters

***A.10*** *Global simulation settings*

| Component | Symbol | Value |
|---|---|---|
| Population size | $N$ | $50{,}000$ (variance analysis), $100{,}000$ (main results) |
| Monte Carlo replications | $R$ | $30$ (variance estimation) |
| Random seed offset | – | $0$ (incremented by scenario) |
| Confidence level | – | $95\%$ |
| Feature dimensions | – | $3$ discrete features ($\{0, \ldots, 6\}$) (not used, only labels are important) |

***A.11*** *True prevalence values used in simulation experiments*

| Parameter | Values |
|---|---|
| True prevalence | $p \in \{0.001, 0.01, 0.05, 0.10\}$ |

***A.12*** *Machine learning model error-rate configurations*

| Parameter | Symbol | Values |
|---|---|---|
| ML true positive rate | $\alpha$ | $\{0.60, 0.70, 0.85, 0.90\}$ |
| ML false positive rate | $\beta$ | $\{0.01, 0.05, 0.10, 0.20\}$ |

***A.13*** *Moderator misclassification parameters*

| Parameter | Symbol | Values |
|---|---|---|
| Moderator true positive rate | $\alpha_H$ | $\{0.75, 0.85, 0.95\}$ |
| Moderator false positive rate | $\beta_H$ | $\{0.05, 0.10, 0.20\}$ |

***A.14*** *Sampling design and inclusion probabilities*

| Approach | Design | Inclusion probabilities | Description |
|---|---|---|---|
| A | Uniform | $p_s = 0.10$ | Simple random sampling |
| A1 | Targeted | $p_s(1) = 0.80$, $p_s(0) = 0.10$ | Risk-based oversampling |

***A.15*** *Parameter settings used for stochastic variance decomposition*

| Component | Parameter | Value |
|---|---|---|
| Fixed prevalence | $p$ | $\{0.001, 0.01, 0.05, 0.10\}$ |
| Variance repetitions | $B$ | $30$ (total), $15$ (stage-wise) |
| Variance components | – | ML, sampling, moderator |
| Interaction term | – | Residual: $V_{\text{total}} - \sum V_{\text{stage}}$ |

# C Project codes

All full codes corresponding to the simulations reported in this thesis are publicly available at: https://github.com/domlei/Estimation-of-Rare-Event-Prevalence-in-Online-Platforms-with-Imperfect-ML-and-Human-Moderation.git.

## C.1 Code for A, A1 approaches generation and plotting

```
import numpy as np
import polars as pl
import pandas as pd
import itertools
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams.update({
    "font.size": 12,
    "axes.titlesize": 15,
    "axes.labelsize": 14,
    "legend.fontsize": 12,
    "legend.title_fontsize": 13,
    "xtick.labelsize": 12,
    "ytick.labelsize": 12,
})

def generate_data_polars(N, p, seed=None):
    """
    Generating population data with true prevalence
    Only includes true label y.
    """
    rng = np.random.default_rng(seed)

    y = (rng.random(N) < p).astype(int)

    return pl.DataFrame({
```

```
        "id": np.arange(N),
        "y": y
    })

# A (SRS): constant sampling probability
def define_sampling_polars_A(df, ps_const=0.1):
    return df.with_columns([
        pl.lit(ps_const).alias("ps")
    ])

def sample_data_polars(df, seed=None):
    np.random.seed(seed)
    rand = pl.Series('rand', np.random.rand(df.height))
    df = df.with_columns([rand])
    return df.with_columns([(pl.col('rand') < pl.col('ps')).alias('is_sampled')])

def simulate_ml_predictions_polars(y: pl.Series, alpha, beta, seed=None):
    """
    Simulating binary ML predictions y_hat with:
      P(y_hat=1 | y=1) = alpha  (TPR)
      P(y_hat=1 | y=0) = beta   (FPR)
    """
    rng = np.random.default_rng(seed)

    y_np = y.to_numpy()
    u = rng.random(len(y_np))

    y_hat = np.where(
        y_np == 1,
        (u < alpha).astype(int),
        (u < beta).astype(int)
    )

    return pl.Series("y_hat", y_hat)

# Simulating inclusion for moderation based on selected rate
def simulate_moderator_polars(df, alpha_H, beta_H, seed=None):
    np.random.seed(seed)
    y = df['y'].to_numpy()
    is_sampled = df['is_sampled'].to_numpy()
    mod_label_vals = np.full(df.height, np.nan)
    sampled_y = y[is_sampled]
    mod_sampled = np.where(
```

```
        sampled_y == 1,
        (np.random.rand(len(sampled_y)) < alpha_H),
        (np.random.rand(len(sampled_y)) < beta_H)
    ).astype(int)
    mod_label_vals[is_sampled] = mod_sampled
    return df.with_columns(pl.Series("mod_label", mod_label_vals))


# A1: defining sampling probability
def define_sampling_polars_A1(df, ps_pos=0.8, ps_neg=0.1):
    return df.with_columns([
        pl.when(pl.col("y_hat") == 1).then(ps_pos).otherwise(ps_neg).alias("ps")
    ])

def estimate_prevalence_polars(df, alpha_H, beta_H):
    sampled_df = df.filter(pl.col('is_sampled'))
    weights = 1 / sampled_df['ps']
    adjusted = (sampled_df['mod_label'] - beta_H) / (alpha_H - beta_H)
    p_hat = (adjusted * weights).sum() / df.height
    ci_width = 1.96 * adjusted.std() * weights.mean() / np.sqrt(max(sampled_df.height, 1))
    return p_hat, (p_hat - ci_width, p_hat + ci_width)

# Monte Carlo runner for A vs A1 (proper bias/var/mse across R runs)
def run_A_vs_A1_mc(
    N=1_000_000,
    p_grid=(0.001, 0.002, 0.005, 0.01, 0.02, 0.05),
    alpha=0.85, beta=0.10,
    alpha_H=0.90, beta_H=0.05,

    # baseline A design grid
    a_grid=(0.05, 0.10),

    # A1 design grid remains the same
    a1_grid=((0.60, 0.05), (0.80, 0.05), (0.95, 0.05), (0.95, 0.20)),
    R=200,
    seed_offset=0
):
    records = []

    for p in p_grid:

        # Approach A (SRS)
        for ps_const in a_grid:
```

```
    p_hats_A = []
    for r in range(R):
        seed = seed_offset + 10_000 * r + int(1e6 * p)

        df = generate_data_polars(N=N, p=p, seed=seed)
        y_hat = simulate_ml_predictions_polars(df["y"], alpha, beta,
        seed=seed + 100)
        df = df.with_columns([y_hat])

        df = define_sampling_polars_A(df, ps_const=ps_const)
        df = sample_data_polars(df, seed=seed + 200)
        df = simulate_moderator_polars(df, alpha_H, beta_H,
        seed=seed + 300)

        p_hat, _ = estimate_prevalence_polars(df, alpha_H, beta_H)
        p_hats_A.append(p_hat)

    p_hats_A = np.array(p_hats_A)
    err_A = p_hats_A - p

    records.append({
        "approach": "A",
        "p_true": p,

        # design parameters
        "ps_const": ps_const,
        "ps_pos": None,
        "ps_neg": None,

        # metrics
        "bias": float(err_A.mean()),
        "var": float(p_hats_A.var(ddof=1)),
        "mse": float((err_A**2).mean()),
        "rmse": float(np.sqrt((err_A**2).mean())),
        "mean_p_hat": float(p_hats_A.mean())
    })

# Approach A1 (unequal-probabilities)
for (ps_pos, ps_neg) in a1_grid:
    p_hats_A1 = []
    for r in range(R):
        seed = seed_offset + 10_000 * r + int(1e6 * p) + 777
```

```
                df = generate_data_polars(N=N, p=p, seed=seed)
                y_hat = simulate_ml_predictions_polars(df["y"], alpha, beta,
                seed=seed + 100)
                df = df.with_columns([y_hat])

                df = define_sampling_polars_A1(df, ps_pos=ps_pos, ps_neg=ps_neg)
                df = sample_data_polars(df, seed=seed + 200)
                df = simulate_moderator_polars(df, alpha_H, beta_H,
                seed=seed + 300)

                p_hat, _ = estimate_prevalence_polars(df, alpha_H, beta_H)
                p_hats_A1.append(p_hat)

            p_hats_A1 = np.array(p_hats_A1)
            err_A1 = p_hats_A1 - p

            records.append({
                "approach": "A1",
                "p_true": p,

                # design parameters
                "ps_const": None,
                "ps_pos": ps_pos,
                "ps_neg": ps_neg,

                # metrics
                "bias": float(err_A1.mean()),
                "var": float(p_hats_A1.var(ddof=1)),
                "mse": float((err_A1**2).mean()),
                "rmse": float(np.sqrt((err_A1**2).mean())),
                "mean_p_hat": float(p_hats_A1.mean())
            })

    return pl.DataFrame(records)


# Plotting functions

def plot_rmse_vs_p(df: pl.DataFrame):
    pdf = df.to_pandas()

    # Creating presentable labels
    def policy_label(row):
```

```
        if row["approach"] == "A":
            return rf"A ($p_s={row['ps_const']:.2f}$)"
        return rf"A1 ($p_s(1)={row['ps_pos']:.2f},\; p_s(0)={row['ps_neg']:.2f}$)"

    pdf["label"] = pdf.apply(policy_label, axis=1)

    plt.figure()
    for label, sub in pdf.groupby("label"):
        sub = sub.sort_values("p_true")
        plt.plot(sub["p_true"], sub["rmse"], marker="o", label=label)

    plt.xscale("log")
    plt.xlabel(r"True prevalence ($p$) (log scale)")
    plt.ylabel("RMSE of estimated prevalence")
    plt.title("A vs A1: Sampling design effect (RMSE)")
    plt.legend()
    plt.tight_layout()
    plt.show()


df_mc = run_A_vs_A1_mc(
    N=1_000_000,
    p_grid=(0.001, 0.002, 0.005, 0.01, 0.02, 0.05),
    alpha=0.85, beta=0.10,
    alpha_H=0.90, beta_H=0.05,
    a_grid=(0.05, 0.10),
    a1_grid=((0.60, 0.05), (0.80, 0.05), (0.95, 0.05), (0.95, 0.20)),
    R=200,
    seed_offset=123
)
print(df_mc)

plot_rmse_vs_p(df_mc)

def plot_bias2_vs_var_at_p(df_mc, p0=0.005):
    """
    At one prevalence p0, compare Bias^2 vs Variance
    across sampling policies (A and A1 variants).
    columns: approach, p_true, ps_const, ps_pos, ps_neg, bias, var
    """
    pdf = df_mc.to_pandas()
    sub = pdf[np.isclose(pdf["p_true"], p0)].copy()
```

```
    # Creating presentable labels
    def label_row(r):
        if r["approach"] == "A":
            return rf"A (SRS, $p_s={r['ps_const']:.2f}$)"
        return rf"A1 ($p_s(1)={r['ps_pos']:.2f},\; p_s(0)={r['ps_neg']:.2f}$)"

    sub["label"] = sub.apply(label_row, axis=1)

    # Metrics
    sub["bias2"] = sub["bias"] ** 2
    sub = sub.sort_values(["approach", "ps_pos", "ps_neg"], na_position="first")

    labels = sub["label"].tolist()
    x = np.arange(len(labels))
    width = 0.40

    plt.figure(figsize=(8, 5))

    plt.bar(x - width/2, sub["var"].to_numpy(), width, label="Variance")
    plt.bar(x + width/2, sub["bias2"].to_numpy(), width, label=r"Bias$^2$")

    plt.xticks(x, labels, rotation=30, ha="right", fontsize=11)
    plt.ylabel("Contribution to MSE", fontsize=12)
    plt.title(rf"Bias-variance relationship at $p={p0:g}$", fontsize=13)

    plt.legend(fontsize=10, title_fontsize=11)
    plt.yticks(fontsize=10)

    plt.tight_layout()
    plt.show()


# Deploy:
plot_bias2_vs_var_at_p(df_mc, p0=0.005)
#plot_bias2_vs_var_at_p(df_mc, p0=0.01)


def plot_lollipop_bias(
    df_mc,
    approach="A",
    title="Bias vs true prevalence (lollipop plot)",
    log_x=False,
    color_map=None
```

```
):
    """
    Lollipop (pole) plot of bias vs prevalence, with colors separating sampling regimes.

    Parameters
    ----------
    df_mc : DataFrame (polars or pandas)
        Columns: approach, p_true, bias
        If approach == "A", should also contain ps_const for regime coloring.
    approach : str
        Which approach to plot (default: "A" for SRS)
    title : str
        Plot title
    log_x : bool
        Whether to use log scale for x-axis (default: False)
    color_map : dict or None
        Optional mapping {ps_const_value: matplotlib_color}. If None, uses defaults.
    """
    # Convert to pandas if needed
    if hasattr(df_mc, "to_pandas"):
        pdf = df_mc.to_pandas()
    else:
        pdf = df_mc.copy()

    # Filter to the requested approach only
    sub = pdf[pdf["approach"] == approach].copy()
    sub = sub.sort_values("p_true")

    plt.figure(figsize=(6, 3.5))

    # Default colors for baseline regimes (override via color_map if you want)
    if color_map is None:
        color_map = {
            0.05: "tab:blue",
            0.10: "tab:orange",
        }

    # If ps_const exists, color by sampling regime; otherwise fall back to single-
color plot
    if "ps_const" in sub.columns and sub["ps_const"].notna().any():
        for ps_val, g in sub.groupby("ps_const"):
            g = g.sort_values("p_true")
            c = color_map.get(ps_val, "gray")
```

```
            plt.vlines(
                g["p_true"],
                0,
                g["bias"],
                color=c,
                alpha=0.7,
                linewidth=2
            )
            plt.scatter(
                g["p_true"],
                g["bias"],
                color=c,
                label=fr"SRS ($\pi$ = {ps_val:.2f})",
                zorder=3
            )
        plt.legend(frameon=True)
    else:
        # Generic single-series lollipop plot
        plt.vlines(sub["p_true"], 0, sub["bias"], alpha=0.7, linewidth=2)
        plt.scatter(sub["p_true"], sub["bias"], zorder=3)

    plt.axhline(0, color="black", linewidth=1)

    if log_x:
        plt.xscale("log")
        plt.xlabel(r"True prevalence $p$ (log scale)")
    else:
        plt.xlabel(r"True prevalence $p$")

    plt.ylabel(r"Bias $\mathbb{E}[\hat p - p]$")
    plt.title(title)

    plt.tight_layout()
    plt.show()


plot_lollipop_bias(
    df_mc,
    approach="A",
    title="Baseline (SRS): Bias vs prevalence"
)
```

## C.2 Code for approach - B generation and plotting

```
import numpy as np
import polars as pl
import pandas as pd
import itertools
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.special import expit as sigmoid
from sklearn.linear_model import LogisticRegression

mpl.rcParams.update({
    "font.size": 12,
    "axes.titlesize": 15,
    "axes.labelsize": 14,
    "legend.fontsize": 12,
    "legend.title_fontsize": 13,
    "xtick.labelsize": 12,
    "ytick.labelsize": 12,
})

# Defined sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Risk-points definition
def compute_risk_points(f1, f2, f3, f4, f5):
    """
    f1..f5 are arrays in [0,1] representing:
    f1: account_age_norm (0=new,1=old)
    f2: activity_burstiness
    f3: link_intensity
    f4: text_effort_low
    f5: duplication_score
    """
    rp  = 2.0 * (1 - f1)  # newer → more risk
    rp += 2.5 * f2        # bursty → more risk
    rp += 2.0 * f3        # many links → more risk
    rp += 1.5 * f4        # low-effort text → more risk
    rp += 2.0 * f5        # duplication → more risk
    return rp
```

```
# Solver for b0 so mean(prob)  target_prevalence
def solve_b0_for_target_prevalence_from_risk(risk_centered, target_prev,
                                             tol=1e-6, max_iter=50):
    """
    risk_centered: numpy array of risk_points - mean(risk_points)
    target_prev: desired average prevalence (e.g. 0.015 for 1.5%)
    """
    def prevalence_given_b0(b0):
        logits = b0 + risk_centered
        return sigmoid(logits).mean()

    low, high = -15.0, 5.0  # wide bracket in log-odds space

    for _ in range(max_iter):
        mid = 0.5 * (low + high)
        prev_mid = prevalence_given_b0(mid)

        if abs(prev_mid - target_prev) < tol:
            return mid

        if prev_mid < target_prev:
            low = mid
        else:
            high = mid

    return 0.5 * (low + high)


def generate_data_and_ml_B_polars(N, target_prev, alpha_target, beta_target, seed=None):
    """
    Approach B:
    - Generate features realistic for marketplace + social media
    - Generate true y via logistic model with intercept b0 tuned to target_prev
    - Fit logistic regression and threshold scores
    to approximate (alpha_target, beta_target)
    Returns:
      df_polars, metrics_dict
    """
    rng = np.random.default_rng(seed)

    # 1) Generate shared features
    # f1: account_age_norm (0=new,1=old)
    account_age_days = rng.exponential(scale=180, size=N)
```

```
f1 = account_age_days / (account_age_days + 365)

# f2: activity_burstiness
actions_last_24h = rng.poisson(lam=3, size=N)
f2 = np.clip(actions_last_24h / 30, 0, 1)

# f3: link_intensity
base_link_prop = rng.beta(1, 5, size=N)
heavy_linkers = rng.random(N) < 0.1
f3 = np.where(
    heavy_linkers,
    np.clip(base_link_prop + 0.6, 0, 1),
    base_link_prop,
)

# f4: text_effort_low
f4 = rng.beta(2, 4, size=N)

# f5: duplication_score
template_users = rng.random(N) < 0.1
f5 = np.where(
    template_users,
    rng.beta(5, 1, size=N),
    rng.beta(1, 5, size=N),
)

X = np.column_stack([f1, f2, f3, f4, f5])

# 2) True risk model → p_true → y
risk_points = compute_risk_points(f1, f2, f3, f4, f5)
risk_centered = risk_points - risk_points.mean()

b0 = solve_b0_for_target_prevalence_from_risk(
    risk_centered=risk_centered,
    target_prev=target_prev,
)

logits = b0 + risk_centered
p_true = sigmoid(logits)

y = (rng.random(N) < p_true).astype(int)

# 3) Fitting logistic regression
```

```
clf = LogisticRegression(solver="lbfgs", max_iter=1000)
clf.fit(X, y)
scores = clf.predict_proba(X)[:, 1]

# 4) Threshold search to approximate (alpha_target, beta_target)
s = scores
P = y.sum()
Nn = N - P
if P == 0 or Nn == 0:
    raise ValueError("Need both positives and negatives in y for ML metrics.")

order = np.argsort(-s)
s_desc = s[order]
y_desc = y[order]

tp_cum = np.cumsum(y_desc)
fp_cum = np.cumsum(1 - y_desc)
TPR = tp_cum / P
FPR = fp_cum / Nn

d2 = (TPR - alpha_target) ** 2 + (FPR - beta_target) ** 2
k_best = int(np.argmin(d2))
threshold = s_desc[k_best]

y_hat = (scores >= threshold).astype(int)

# Confusion matrix
TP = int(((y_hat == 1) & (y == 1)).sum())
FP = int(((y_hat == 1) & (y == 0)).sum())
TN = int(((y_hat == 0) & (y == 0)).sum())
FN = int(((y_hat == 0) & (y == 1)).sum())

TPR_final = TP / P
FPR_final = FP / Nn

# Dataframe of features and risk points
df = pl.DataFrame({
    "f1": f1,
    "f2": f2,
    "f3": f3,
    "f4": f4,
    "f5": f5,
    "risk_points": risk_points,
```

```
        "p_true": p_true,
        "y": y,
        "score": scores,
        "y_hat": y_hat,
    })

    metrics = {
        "b0": b0,
        "true_prevalence": float(y.mean()),
        "mean_p_true": float(p_true.mean()),
        "ml_TPR": TPR_final,
        "ml_FPR": FPR_final,
        "ml_threshold": float(threshold),
        "P": int(P),
        "N": int(N),
    }

    return df, metrics

def define_sampling_polars_score(df, target_rate=0.1, gamma=1.5,
                                 ps_min=0.01, ps_max=0.8):
    """
    Approach B-risk: sampling probabilities based on continuous ML scores.

    target_rate: desired average sampling fraction (e.g. 0.1 = 10% of items)
    gamma: how aggressively to focus on high-score items
    """
    scores = df["score"].to_numpy()
    # avoid zeros
    scores = np.clip(scores, 1e-6, 1 - 1e-6)

    # importance weights proportional to score^gamma
    w = scores ** gamma

    # scale so that mean(ps)  target_rate
    c = target_rate * len(scores) / w.sum()
    ps = np.clip(c * w, ps_min, ps_max)

    return df.with_columns(pl.Series("ps", ps))


def sample_data_polars(df, seed=None):
    np.random.seed(seed)
```

```
    rand = pl.Series('rand', np.random.rand(df.height))
    df = df.with_columns([rand])
    return df.with_columns([
        (pl.col('rand') < pl.col('ps')).alias('is_sampled')
    ])

def simulate_moderator_polars(df, alpha_H, beta_H, seed=None):
    np.random.seed(seed)

    y = df['y'].to_numpy()
    is_sampled = df['is_sampled'].to_numpy()

    mod_label_vals = np.full(df.height, np.nan)  # use NaN instead of None

    sampled_y = y[is_sampled]
    mod_sampled = np.where(
        sampled_y == 1,
        (np.random.rand(len(sampled_y)) < alpha_H),
        (np.random.rand(len(sampled_y)) < beta_H)
    ).astype(int)

    mod_label_vals[is_sampled] = mod_sampled

    return df.with_columns(pl.Series("mod_label", mod_label_vals))

def estimate_prevalence_polars(df, alpha_H, beta_H):
    sampled_df = df.filter(pl.col('is_sampled'))
    weights = 1 / sampled_df['ps']
    adjusted = (sampled_df['mod_label'] - beta_H) / (alpha_H - beta_H)
    p_hat = (adjusted * weights).sum() / df.height
    ci_width = 1.96 * adjusted.std() * weights.mean() / np.sqrt(sampled_df.height)
    return p_hat, (p_hat - ci_width, p_hat + ci_width)

results_polars = []

def run_parameter_grid_simulation_B(N=100_000, param_grid=None, seed_offset=0, verbose=False):

    if param_grid is None:
        param_grid = list(itertools.product(
            [0.001, 0.01, 0.05, 0.1],   # target prevalence (p)
            [0.7, 0.85],                # target ML TPR (alpha)
            [0.1, 0.2],                 # target ML FPR (beta)
            [0.85, 0.95],               # Moderator TPR (alpha_H)
```

```
        [0.05, 0.1]                 # Moderator FPR (beta_H)
    ))

results = []

for i, (p, alpha, beta, alpha_H, beta_H) in enumerate(param_grid):
    if verbose:
        print(f"Running combo {i+1}/{len(param_grid)}: "
              f"p_target={p},  _target={alpha},  _target={beta}, "
              f" _H={alpha_H},  _H={beta_H}")

    # Approach B: data + ML predictions
    df, ml_metrics = generate_data_and_ml_B_polars(
        N=N,
        target_prev=p,
        alpha_target=alpha,
        beta_target=beta,
        seed=seed_offset + i,
    )

    # Sampling, moderator, estimator
    df = define_sampling_polars_score(df)
    df = sample_data_polars(df, seed=seed_offset + i + 200)
    df = simulate_moderator_polars(df, alpha_H, beta_H, seed=seed_offset + i + 300)
    p_hat, conf_int = estimate_prevalence_polars(df, alpha_H, beta_H)

    # realized prevalence from y
    realized_prev = float(df["y"].mean())

    # Store results + intermediate metrics
    results.append({
        # targets
        "p_target": p,
        "alpha_target": alpha,
        "beta_target": beta,
        "alpha_H": alpha_H,
        "beta_H": beta_H,

        # world + ML
        "true_prevalence": realized_prev,
        "mean_p_true": ml_metrics["mean_p_true"],
        "b0": ml_metrics["b0"],
        "ml_TPR": ml_metrics["ml_TPR"],
```

```
            "ml_FPR": ml_metrics["ml_FPR"],
            "ml_threshold": ml_metrics["ml_threshold"],

            # estimator
            "p_hat": p_hat,
            "conf_low": conf_int[0],
            "conf_high": conf_int[1],
            "bias_vs_target": p_hat - p,
            "bias_vs_realized": p_hat - realized_prev,
            "ci_width": conf_int[1] - conf_int[0],
        })

    results_df = pl.DataFrame(results)
    return results_df

p_list = [0.001, 0.01]                #
alpha_ml_list = [0.85]                # fix ML (or add 0.70 as another column)
beta_ml_list  = [0.10]                # fix ML FPR

alpha_H_list = [0.70, 0.80, 0.85, 0.90, 0.95]   # moderator TPR
beta_H_list  = [0.01, 0.03, 0.05, 0.08, 0.12]   # moderator FPR

param_grid_biasmap_modq = list(itertools.product(
    p_list,
    alpha_ml_list,
    beta_ml_list,
    alpha_H_list,
    beta_H_list
))

# Deploying simulation
results_df_polars = run_parameter_grid_simulation_B(
N=1_000_00, param_grid=param_grid_biasmap_modq,verbose=True)

print(results_df_polars)


pivot = (
    results_df_polars
    .to_pandas()
    .pivot_table(
        index="alpha_H",
        columns="beta_H",
```

```
        values="bias_vs_realized",
        aggfunc="median"
#        aggfunc="mean"
    )
)

print(pivot)

sns.heatmap(
    pivot,
    annot=True,
    cmap="coolwarm",
    center=0
)

plt.title("Bias Heatmap by Moderator accuracy")
plt.xlabel(r"Moderator FPR ($\beta_H$)")
plt.ylabel(r"Moderator TPR ($\alpha_H$)")
plt.show()

###
# Graph: CI width by moderator TPR
results_pd = results_df_polars.to_pandas()

results_pd['ci_width'] = results_pd['conf_high'] - results_pd['conf_low']

sns.boxplot(data=results_pd, x='alpha_H', y='ci_width')
plt.title("Confidence Interval Width by Moderator accuracy")
plt.ylabel("Confidence Interval Width")
plt.xlabel(r"Moderator TPR ($\alpha_H$)")
plt.show()
###

# Graph: B bias heatmap - median

p_list = [0.001, 0.01]  # rare vs less rare

# Iterable
alpha_H_list = [0.90]   # moderator TPR ( _H)
beta_H_list  = [0.05]   # moderator FPR ( _H)

# ML target grid (axes of heatmap)
alpha_ml_list = [0.70, 0.80, 0.85, 0.90, 0.95]   # ML target TPR ( )
```

```
beta_ml_list  = [0.02, 0.05, 0.10, 0.15, 0.20]   # ML target FPR ( )

param_grid_biasmap = list(itertools.product(
    p_list,
    alpha_ml_list,
    beta_ml_list,
    alpha_H_list,
    beta_H_list
))

results_df_polars = run_parameter_grid_simulation_B(
    N=1_000_000,
    param_grid=param_grid_biasmap,
    verbose=True
)

results_pd = results_df_polars.to_pandas()


pivot_all = (
    results_pd
    .pivot_table(
        index="alpha_target",        # ML target TPR ( )
        columns="beta_target",       # ML target FPR ( )
        values="bias_vs_realized",
        aggfunc="median"
    )
    .sort_index()
    .sort_index(axis=1)
)

plt.figure(figsize=(6.8, 4.2))
sns.heatmap(pivot_all, annot=True, cmap="coolwarm", center=0)
plt.title("Bias heatmap over ML targets (median over all scenarios)")
plt.xlabel(r"ML target FPR ($\beta$)")
plt.ylabel(r"ML target TPR ($\alpha$)")
plt.tight_layout()
plt.show()
```

## C.3 Code for Variance Decomposition for SRS sampling specification

```
import numpy as np
import polars as pl
```

```
import pandas as pd
import itertools
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams.update({
    "font.size": 12,
    "axes.titlesize": 15,
    "axes.labelsize": 14,
    "legend.fontsize": 12,
    "legend.title_fontsize": 13,
    "xtick.labelsize": 12,
    "ytick.labelsize": 12,
})

# Function 1: Generating feature set and true labels
def generate_data_polars(N, p, seed=None):
    np.random.seed(seed)
    y = (np.random.rand(N) < p).astype(int)
    feat_1 = np.random.randint(0, 7, size=N)
    feat_2 = np.random.randint(0, 7, size=N)
    feat_3 = np.random.randint(0, 7, size=N)

    return pl.DataFrame({
        'feat_1': feat_1,
        'feat_2': feat_2,
        'feat_3': feat_3,
        'y': y
    })

# Function 2: Simulating ML predictions with confusion matrix
def simulate_ml_predictions_polars(y, alpha, beta, seed=None):
    np.random.seed(seed)
    y_np = y.to_numpy()
    y_hat = np.zeros_like(y_np)
    y_hat[y_np == 1] = (np.random.rand(np.sum(y_np == 1)) < alpha).astype(int)
    y_hat[y_np == 0] = (np.random.rand(np.sum(y_np == 0)) < beta).astype(int)
    return pl.Series('y_hat', y_hat)

# Function 3: Defining sampling probabilities
def define_sampling_polars(df):
    return df.with_columns([
```

```
        pl.when(pl.col('y_hat') == 1).then(0.8).otherwise(0.1).alias('ps')
    ])

# Function 4: Sampling data based on probabilities
def sample_data_polars(df, seed=None):
    np.random.seed(seed)
    rand = pl.Series('rand', np.random.rand(df.height))
    df = df.with_columns([rand])
    return df.with_columns([
        (pl.col('rand') < pl.col('ps')).alias('is_sampled')
    ])

# Function 5: Simulating human moderator labels
def simulate_moderator_polars(df, alpha_H, beta_H, seed=None):
    np.random.seed(seed)

    y = df['y'].to_numpy()
    is_sampled = df['is_sampled'].to_numpy()

    mod_label_vals = np.full(df.height, np.nan)  # use NaN instead of None

    sampled_y = y[is_sampled]
    mod_sampled = np.where(
        sampled_y == 1,
        (np.random.rand(len(sampled_y)) < alpha_H),
        (np.random.rand(len(sampled_y)) < beta_H)
    ).astype(int)

    mod_label_vals[is_sampled] = mod_sampled

    return df.with_columns(pl.Series("mod_label", mod_label_vals))

# Function 6: Estimating prevalence
def estimate_prevalence_polars(df, alpha_H, beta_H):
    sampled_df = df.filter(pl.col('is_sampled'))
    weights = 1 / sampled_df['ps']
    adjusted = (sampled_df['mod_label'] - beta_H) / (alpha_H - beta_H)
    p_hat = (adjusted * weights).sum() / df.height
    ci_width = 1.96 * adjusted.std() * weights.mean() / np.sqrt(sampled_df.height)
    return p_hat, (p_hat - ci_width, p_hat + ci_width)

# Grid of parameters
param_grid = list(itertools.product(
```

```
    [0.001, 0.01, 0.05, 0.1], # p
    [0.7, 0.85], # alpha
    [0.1, 0.2], # beta
    [0.85, 0.95], # alpha_H
    [0.05, 0.1] # beta_H
))

# Modified parameter grid
param_grid_base = list(itertools.product(
    [0.001, 0.01, 0.05, 0.1],          # p (true prevalence)
    [0.6, 0.7, 0.85, 0.9],             # alpha (model TPR)  (+ add 0.6, 0.9)
    [0.01, 0.05, 0.1, 0.2],            # beta  (model FPR)  (+ add lower-FPR cases)
    [0.75, 0.85, 0.95],                # alpha_H (human TPR) (+ add a weaker-
human case)
    [0.05, 0.1, 0.2]                   # beta_H  (human FPR) (+ add non-ideal human case)
))

results_polars = []

def run_parameter_grid_simulation(N=100_000_0, param_grid=None, seed_offset=0, verbose=False):
    # default grid
    if param_grid is None:
        param_grid = list(itertools.product(
            [0.001, 0.01, 0.05, 0.1],   # p_true
            [0.7, 0.85],                # ML TPR (alpha)
            [0.1, 0.2],                 # ML FPR (beta)
            [0.85, 0.95],               # Moderator TPR (alpha_H)
            [0.05, 0.1]                 # Moderator FPR (beta_H)
        ))

    results = []

    # Main simulation loop
    for i, (p, alpha, beta, alpha_H, beta_H) in enumerate(param_grid):
        if verbose:
            print(f"Running combo {i+1}/{len(param_grid)}: "
                  f"p={p}, ={alpha}, ={beta}, _H={alpha_H}, _H={beta_H}")

        # --- Pipeline steps ---
        df = generate_data_polars(N=N, p=p, seed=seed_offset + i)
        y_hat = simulate_ml_predictions_polars(df["y"], alpha, beta,
        seed=seed_offset + i + 100)
        df = df.with_columns([y_hat])
```

```
        df = define_sampling_polars(df)
        df = sample_data_polars(df, seed=seed_offset + i + 200)
        df = simulate_moderator_polars(df, alpha_H, beta_H,
        seed=seed_offset + i + 300)
        p_hat, conf_int = estimate_prevalence_polars(df, alpha_H, beta_H)

        # --- Store results ---
        results.append({
            "p_true": p,
            "alpha": alpha,
            "beta": beta,
            "alpha_H": alpha_H,
            "beta_H": beta_H,
            "p_hat": p_hat,
            "conf_low": conf_int[0],
            "conf_high": conf_int[1],
            "bias": p_hat - p,
            "ci_width": conf_int[1] - conf_int[0]
        })

    # Convert to Polars DataFrame
    results_df = pl.DataFrame(results)

    return results_df


def estimate_total_variance(params, B=100):
    """
    Estimate total (stochastic) variance by repeating the full simulation B times.
    Each run uses a new seed.
    """
    p_hats = []
    for b in range(B):
        df = generate_data_polars(N=params["N"], p=params["p"], seed=params["seed_offset"])
        y_hat = simulate_ml_predictions_polars(
            df["y"], params["alpha"], params["beta"], seed=params["seed_offset"]
            + b + 100)
        df = df.with_columns([y_hat])
        df = define_sampling_polars(df)
        df = sample_data_polars(df, seed=params["seed_offset"] + b + 200)
        df = simulate_moderator_polars(df, params["alpha_H"], params["beta_H"],
        seed=params["seed_offset"] + b + 300)
        p_hat, _ = estimate_prevalence_polars(df, params["alpha_H"], params["beta_H"])
```

```
        p_hats.append(p_hat)
    return np.var(p_hats, ddof=1)


def estimate_stage_variance(params, stage, B=50):
    p_hats = []

    for b in range(B):
        # Always starting new population for each repetition
        df = generate_data_polars(N=params["N"], p=params["p"], seed=params["seed_offset"])

        # ML stage - resimulate if shaking ML, otherwise fixed
        if stage == "ml":
            y_hat = simulate_ml_predictions_polars(
                df["y"], params["alpha"], params["beta"],
                seed=params["seed_offset"] + 1000 + b)
        else:
            y_hat = simulate_ml_predictions_polars(
                df["y"], params["alpha"], params["beta"],
                seed=params["seed_offset"] + 100)  # fixed baseline
        df = df.with_columns([y_hat])
        df = define_sampling_polars(df)

        # Sampling stage - resimulate if shaking sampling
        if stage == "sampling":
            df = sample_data_polars(df, seed=params["seed_offset"] + 2000 + b)
        else:
            df = sample_data_polars(df, seed=params["seed_offset"] + 200)

        # Moderator stage - resimulate only if shaking moderator
        if stage == "moderator":
            df = simulate_moderator_polars(df, params["alpha_H"], params["beta_H"],
                                           seed=params["seed_offset"] + 3000 + b)
        else:
            df = simulate_moderator_polars(df, params["alpha_H"], params["beta_H"],
                                           seed=params["seed_offset"] + 300)
        # Estimate prevalence
        p_hat, _ = estimate_prevalence_polars(df, params["alpha_H"], params["beta_H"])
        p_hats.append(p_hat)

    # Handle degenerate cases
    p_hats = np.array(p_hats)
    if np.all(np.isnan(p_hats)) or len(np.unique(p_hats)) == 1:
```

```
        return np.nan
    return np.nanvar(p_hats, ddof=1)

def run_parameter_grid_with_variance(N=100_000,
                                     param_grid=None,
                                     B=200,
                                     seed_offset=0,
                                     verbose=False):
    """
    Run the prevalence simulation and estimate stochastic variance components
    (total and stage-wise) for each parameter setting.
    """
    # Reusing existing simulation for main results
    results_df = run_parameter_grid_simulation(N=N, param_grid=param_grid,
    seed_offset=seed_offset, verbose=verbose)
    results_pd = results_df.to_pandas()

    # Storage for variance diagnostics
    variance_records = []

    for i, row in results_pd.iterrows():
        params = dict(
            N=N,
            p=row["p_true"],
            alpha=row["alpha"],
            beta=row["beta"],
            alpha_H=row["alpha_H"],
            beta_H=row["beta_H"],
            seed_offset=seed_offset + i * 10_000
        )

        if verbose:
            print(f"→ Estimating variances for p={params['p']},
             ={params['alpha']},  ={params['beta']}, "
                  f" _H={params['alpha_H']},  _H={params['beta_H']}")

        var_total = estimate_total_variance(params, B=B)
        var_ml = estimate_stage_variance(params, stage="ml", B=B//2)
        # Smaller B for faster run-time
        var_sampling = estimate_stage_variance(params, stage="sampling", B=B//2)
        var_mod = estimate_stage_variance(params, stage="moderator", B=B//2)
        var_inter = var_total - (var_ml + var_sampling + var_mod)
```

```
        variance_records.append({
            "p_true": params["p"],
            "alpha": params["alpha"],
            "beta": params["beta"],
            "alpha_H": params["alpha_H"],
            "beta_H": params["beta_H"],
            "var_stoch_total": var_total,
            "var_stoch_ml": var_ml,
            "var_stoch_sampling": var_sampling,
            "var_stoch_mod": var_mod,
            "var_stoch_inter": var_inter
        })

    variance_df = pl.DataFrame(variance_records)

    # Merge results and variances
    combined = results_df.join(variance_df,
    on=["p_true", "alpha", "beta", "alpha_H", "beta_H"])

    return combined


results_with_variance = run_parameter_grid_with_variance(
    N=50_000,  # smaller N for presentation
    param_grid=param_grid_base,
    B=30,      # repetitions per variance estimate
    verbose=True
)

print(results_with_variance.head())

results_with_variance_calc = results_with_variance.with_columns([
    (pl.col("var_stoch_ml") / pl.col("var_stoch_total")).alias("share_ml"),
    (pl.col("var_stoch_sampling") /
    pl.col("var_stoch_total")).alias("share_sampling"),
    (pl.col("var_stoch_mod") / pl.col("var_stoch_total")).alias("share_mod"),
])

df_calc = results_with_variance_calc.to_pandas()

df_plot = results_with_variance.to_pandas().rename(columns={
    "var_stoch_total": "Total stochastic variance",
    "p_true": "True prevalence $p$",
```

```
    "alpha_H": "Moderator TPR"
})

sns.barplot(
    data=df_plot,
    x="True prevalence $p$",
    y="Total stochastic variance",
    hue="Moderator TPR"
)
plt.title("Total stochastic variance by moderator accuracy")
plt.ylabel("Total stochastic variance")
plt.xlabel("True prevalence $p$")
plt.legend(title="Moderator TPR")
plt.show()



## Outside function env below
results_df_polars = run_parameter_grid_simulation(
N=100_000, param_grid=param_grid_base, verbose=True)

#
print(results_df_polars)
print(len(results_df_polars))

## Graph: CI width by moderator TPR
results_pd = results_df_polars.to_pandas()

results_pd['ci_width'] = results_pd['conf_high'] - results_pd['conf_low']

sns.boxplot(data=results_pd, x='alpha_H', y='ci_width')
plt.title(r"Confidence Interval Width by Moderator TPR ($\alpha_H$)")
plt.ylabel("Confidence Interval Width")
plt.xlabel(r"Moderator TPR ($\alpha_H$)")
plt.show()



results_plot = results_with_variance.select([
    "p_true",
    "alpha",
    "beta",
    "alpha_H",
    "beta_H",
    "var_stoch_ml",
```

```
    "var_stoch_sampling",
    "var_stoch_mod"
])

df_melt = results_plot.to_pandas().melt(
    id_vars=["p_true", "alpha", "beta", "alpha_H", "beta_H"],
    value_vars=["var_stoch_ml", "var_stoch_sampling", "var_stoch_mod"],
    var_name="Stage",
    value_name="Variance"
)

stage_labels = {
    "var_stoch_ml": "ML",
    "var_stoch_sampling": "Sampling",
    "var_stoch_mod": "Moderator"
}

df_melt["Stage"] = df_melt["Stage"].map(stage_labels)

plt.figure(figsize=(8,5))
sns.barplot(
    data=df_melt,
    x="p_true",
    y="Variance",
    hue="Stage"
)
plt.title("Marginal Variance Contributions by Stage")
plt.ylabel("Variance")
plt.xlabel("True Prevalence rate $p$")
plt.legend(title="Variance Source")
plt.show()
```