

# Importance Sampling is All You Need: Predict LLM’s performance on new benchmark by reusing existing benchmark.

Junjie Shi  
CHUNKIT001@e.ntu.edu.sg  
Nanyang Technological University  
Singapore

Wei Ma\*  
weima@smu.edu.sg  
Singapore Management University  
Singapore

Shi Ying  
yingshi@whu.edu.cn  
Wuhan University  
Wuhan, Hubei, China

Lingxiao Jiang  
lxjiang@smu.edu.sg  
Singapore Management University  
Singapore

Yang Liu  
yangliu@ntu.edu.sg  
Nanyang Technological University  
Singapore

Bo Du  
dubo@whu.edu.cn  
Wuhan University  
Wuhan, Hubei, China

## Abstract

With the rapid advancement of large language models (LLMs), code generation has become a key benchmark for evaluating LLM capabilities. However, existing benchmarks face two major challenges: (1) the escalating cost of constructing high-quality test suites and reference solutions, and (2) the increasing risk of data contamination, which undermines the reliability of benchmark-based evaluations.

In this paper, we propose **BIS**, a prompt-centric evaluation framework that enables ground-truth-free prediction of LLM performance on code generation tasks. Rather than executing generated code, **BIS** estimates performance metrics by analyzing the prompt distribution alone. Built on importance sampling theory and implemented using Importance Weighted Autoencoders (IWAE), our method reweights samples from existing annotated benchmarks to estimate performance on new, unseen benchmarks. To stabilize the estimation, we introduce weight truncation strategies and compute marginal expectations across the fitted distributions. **BIS** serves as a complementary tool that supports benchmark development and validation under constrained resources, offering actionable and quick feedback for prompt selection and contamination assessment.

We conduct extensive experiments involving 8,000 evaluation points across 4 CodeLlama models (7B–70B) and 9 diverse benchmarks. Our framework achieves an average absolute prediction error of 1.1% for code correctness scores, with best- and worst-case errors of 0.3% and 1.9%, respectively. It also generalizes well to other metrics, attaining average absolute errors of 2.15% for pass@1. These results demonstrate the reliability and broad applicability of **BIS**, which can significantly reduce the cost and effort of benchmarking LLMs in code-related tasks.

\*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference’17, Washington, DC, USA

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM  
<https://doi.org/XXXXXXX.XXXXXXX>

## Keywords

Code Generation, Large Language Models, Importance Sampling, Benchmarking, Prompt Engineering

## 1 Introduction

Large Language Models (LLMs) have demonstrated exceptional capabilities across various software engineering tasks, with code generation being a particularly notable example. As coding becomes one of the most critical benchmarks for assessing the performance of LLMs, it is essential to evaluate their coding abilities rigorously [4]. To this end, numerous benchmarks have been developed to measure LLMs’ coding proficiency, including SWE-Bench [6], HumanEval [5], and BigCodeBench [39]. Despite their utility, current LLM evaluation benchmarks face two major challenges:

**High Benchmark Development Costs:** Constructing reliable benchmarks requires significant manual effort, especially for developing detailed test suites and reference solutions. For example, BigCodeBench [39] required the manual creation of over 5,000 tests. Attempts to automate test suite generation have been inadequate, with current automated methods achieving only around 39.2% accuracy [13].

**Risk of Data Contamination:** Publicly available benchmarks and test suites can unintentionally become part of the LLM training datasets directly or indirectly, leading to artificially inflated performance over time and undermining benchmark reliability.

Motivated by these limitations, we introduce a novel evaluation paradigm (**BIS**, Prompt Importance Sampling) inspired by a previously overlooked insight: under fixed trained-well LLM parameters and evaluation criteria, prompt distributions inherently determine the LLM observed performance. By leveraging this relationship, we propose an evaluation framework that can estimate LLM performance by analyzing prompt distributions alone—without executing generated code or relying on costly ground-truth solutions.

To the best of our knowledge, this is the first work to predict LLM code generation performance without ground-truth execution. Our framework is grounded in importance sampling, a statistical technique widely used in off-policy reinforcement learning [10, 22, 32], which allows us to estimate expectations under a target distribution using samples from a different, known distribution.

In our setting, this means estimating the expected performance of an LLM on a new set of prompts by reweighting existing prompts

from known benchmarks. We achieve this by modeling prompt distributions using Importance Weighted Autoencoders (IWAE), which are well-suited for capturing complex, multimodal distributions [3]. By learning rich latent representations of prompt distributions, IWAE enables effective approximation of the importance weights, making it possible to reuse prior benchmark data to predict performance on novel code generation tasks.

Our method complements ground-truth-based evaluation by alleviating two key challenges: reducing benchmark construction costs and mitigating data contamination risks. First, to mitigate high benchmark development costs, **BIS** enables early-stage performance estimation by transferring knowledge from existing, fully annotated benchmarks. This allows benchmark designers to assess the value of candidate tasks before committing to expensive test suite construction. Second, to help reduce the risk of data contamination, **BIS** provides an indirect signal of model familiarity with the prompt distribution. By analyzing discrepancies between expected and predicted performance patterns, it can help flag tasks that may overlap with training data, even in the absence of ground-truth execution. This complementary role positions **BIS** as a valuable tool for guiding benchmark design and validation, especially when operating under limited resources or concerns about data leakage.

In theory, we analyze why **BIS** is effective and how it provides an unbiased estimation of LLM performance on the target benchmark. Furthermore, we discuss the practical feasibility of our approach. Additional details are provided in subsection 3.3. We conducted extensive experiments involving approximately 8,000 data points across 4 LLMs (CodeLlama models ranging from 7B to 70B parameters) and 9 diverse benchmarks. Results demonstrate that our framework achieves an average prediction error as low as 0.9% on code correctness metrics, outperforming baseline distribution modeling methods. Additionally, our framework accurately generalizes to other crucial code-related metrics, including maintainability and security scores, validating its broad applicability.

In summary, the contributions of this paper include:

- (1) Introducing the theoretical formalization explicitly linking prompt distributions to LLM performance, providing a rigorous conceptual foundation for prompt-based evaluation methods.
- (2) Proposing the first prompt-centric, ground-truth-free evaluation framework, **BIS**, tailored specifically for code-generation tasks, effectively addressing high benchmark development costs and data contamination risks.
- (3) Innovatively integrating importance sampling with IWAE, empirically validated through comprehensive experiments demonstrating improved accuracy, interpretability, and robustness.

The remainder of the paper is organized as follows: section 2 discusses related work. section 3 presents our methodology. section 4 reports the experimental evaluation. section 5 discusses implications and limitations. section 6 concludes.

## 2 Related work

### 2.1 Code Benchmark

So far, benchmarking for LLMs in code generation now encompasses a wide range of scenarios, including smart contracts[35],

realistic environment settings[6], and large-scale codebases[15]. These benchmarks typically comprise at least two core components: (1) the code evaluation prompt, and (2) the test suite. Many benchmarks additionally incorporate reference solutions, problem sources, and other metadata. During evaluation, the LLM receives the code evaluation prompt as input and generates an output. This output undergoes formatting before being executed against the test suite. Consequently, a high-quality test suite constitutes a critical element of a valid and reliable code benchmark. The development of such test suites frequently necessitates manual curation by human experts, resulting in significant costs. Some alternative ground-truth-free methodologies have been proposed in other domains, such as benchmark-free scoring employing human or LLM-based reviewers. But they are unsuitable for code tasks given they may introduce extra bias and substantial cost.

### 2.2 Importance Sampling

Importance sampling[31] is a Monte Carlo method[14] designed to approximate expectations under a target distribution that is difficult to sample from directly, achieved by reweighting samples drawn from an accessible proposal distribution. Within this framework, two distinct distributions are involved. First, the target distribution  $p(x)$  is intractable to sample from directly. Second, the proposal distribution  $q(x)$  is from which samples can be readily generated. The objective of importance sampling is to estimate the expectation of a function  $f(x)$  with respect to the target distribution  $p(x)$ . This is formally expressed as:  $\mathbb{E}_{x \sim p}[f(x)] = \int f(x)p(x)dx$ . Importance Sampling estimates the expectation under a target distribution by weighting samples drawn from a proposal distribution. This process is formalized as follows, where  $x_i$  denotes a sample drawn from  $q$ :  $\mathbb{E}_{x \sim p}[f(x)] \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \cdot \frac{p(x_i)}{q(x_i)}$ ,  $x_i \sim q(x)$ .

As shown in the equation above, the importance weight for sample  $x_i$  is obtained by dividing the marginal probability of  $x_i$  under the target distribution by its marginal probability under the proposal distribution. Within reinforcement learning, importance sampling is extensively utilized in scenarios characterized by data scarcity or challenging sampling conditions[17]. When direct sampling from the target environment proves difficult or costly, trajectories generated by a behavioral policy may be employed for policy evaluation. By adjusting these trajectories through importance sampling weights, the performance of the target policy can be estimated without altering the policy itself. This technique offers several distinct advantages:

*Distributional Agnosticism:* Importance sampling does not rely on a pre-specified family of probability distributions. It requires no assumption that the data adheres to a specific parametric form, thereby accommodating a broad spectrum of distribution types.

*Mathematical Grounding and Interpretability:* By computing weight explicitly, importance sampling provides inherent interpretability and is unbiased estimation[31].

### 2.3 Variational Inference and Variational Autoencoder

Variational inference(VI) [2] is a technique for approximating complex posterior distributions given observed data. Computationally, VI achieves this approximation by maximizing the evidence

lower bound (ELBO) on the marginal likelihood of the target distribution. Fundamentally, this approach minimizes the Kullback-Leibler (KL) divergence[33] between the approximating distribution and the true target distribution, as formalized below:  $\log p(x) = \mathcal{L}(q) + D_{KL}(q(z; \theta) \| p(z|x))$ , where  $p$  denotes the target posterior distribution, while  $q$  represents the approximating distribution. The parameters of the  $q$  distribution are denoted by  $\theta$ , and  $z$  refers to a sample drawn from the latent variable space. The observed data is represented by  $x$ , and  $\mathcal{L}(q)$  stands for the ELBO, which serves as a variational objective function.

Variational Autoencoder (VAE) [23] integrates variational inference with deep learning, commonly employed for unsupervised learning tasks. Its architecture comprises an encoder layer, which maps input data to a latent space; a sampling layer, responsible for drawing samples from this latent space; and a decoder layer, which reconstructs the input from the sampled latent representations. The VAE uses ELBO as reconstruction loss and uses KL divergence as a regularization term.

IWAE (Importance Weighted Autoencoder), a variant of VAE, achieves a tighter ELBO by performing multiple latent space samplings and weighting the importance of each sample. The ELBO constructed by IWAE is given below:

$$ELBO_{IWAE} = \mathbb{E}_{z_1, \dots, z_K \sim q(z|x)} \left[ \log \left( \frac{1}{K} \sum_{k=1}^K \frac{p(x, z_k)}{q(z_k|x)} \right) \right] \quad (1)$$

Where  $K$  is the number of samples,  $z_k \sim q(z|x)$  are  $K$  samples drawn from the approximate posterior distribution,  $x$  is the observed data, and  $q(z_k|x)$  is the variational approximation produced by IWAE. The term  $p(x, z_k)$  represents the joint distribution of the observed data and the sampled latent variables. As  $K$  increases, the ELBO of IWAE approaches the log-likelihood of the target distribution more closely. When  $K = 1$ , IWAE reduces to the standard VAE.

### 3 Study Design

#### 3.1 Motivation

Our motivation originates from a basic yet under-explored observation: the performance of LLMs in code generation tasks is highly sensitive to variations in prompt wording, structure, and contextual details. While it is widely recognized that prompts significantly influence the performance of code-generating LLMs, existing evaluation methodologies largely neglect leveraging prompt distributional characteristics. Instead, they heavily rely on executing generated code against manually crafted test suites—an approach inherently expensive and prone to data contamination issues if there is not quick feedback.

A natural yet unexplored insight from our formal analysis reveals that when the model and benchmark conditions are fixed, prompt distribution itself uniquely determines the expected model performance. While intuitive, this theoretical equivalence has been overlooked by prior works, which inherently assumed that explicit code execution and reference solutions are indispensable for accurate benchmarking.

Some of notations we use in this paper is shown in Table 1. Formally, consider an LLM as a probabilistic generative model

$P_{LLM}(c|t)$ . Given a fixed model and benchmark evaluation metrics  $f_b$ , the expected score for a given prompt distribution  $P_t$  can be succinctly represented as:

$$\mathbb{E}_{t \sim P_t} [score] = \int_t \int_c f_b(c) \cdot P_{LLM}(c|t) \cdot P_t(t) dcdt \quad (2)$$

This equation reveals that, under fixed model parameters and evaluation setup, performance estimation fundamentally reduces to capturing the underlying prompt distribution  $P_t$ , which has not been previously exploited explicitly. Leveraging this insight, our work proposes an innovative, prompt-centric evaluation framework that eliminates the need for code execution and expensive ground-truth test suites.

**Table 1: Notation**

Notation	Meaning
$t$	prompt
$c$	code
$b$	benchmark
$f_b$	the evaluation function in benchmark
$P_{LLM}$	the output distribution of LLM
$P_t$	the output distribution of prompt
$p_{source}^t$	the output distribution of source prompt set
$p_{target}^t$	the output distribution of target prompt set
$E_{P_t}(score)$	the expectation score of LLM on prompt distribution $P_t$
$t_{source}^{(i)}$	the $i$ -th prompt from source prompt set
$IWAE_{source}$	The IWAE model trained with source prompt set
$IWAE_{target}$	The IWAE model trained with target prompt set
$\hat{y}_{target}$	Prediction score for target prompt set

In practice, we use average values of all samples to estimate expected score of LLM on  $P_t$ .

$$\mathbb{E}_{t \sim P_t} [score] \approx \frac{1}{N} \sum_{i=1}^N f_b(c_i) \quad (3)$$

However, as noted previously, the test suite embodied by  $f_b$  is often challenging to develop in practice. Consequently, for a given code-base, we may find that  $f_b$  is difficult to obtain. Therefore, we return to the initial point. In Equation 2, we note that although actual implementations require testing suites for distinct code samples in practice,  $f_b$  can theoretically be regarded as a deterministic and known function. On the other hand, we can assume LLM is also known. This holds because while the precise mechanism by which the LLM generates its probability distribution remains undisclosed, we can approximate this distribution through sampling. This implies that to obtain the expectation in Equation 4, we only need to compute the distribution corresponding to the prompts. Thus, we have successfully formalized the statement and proven our conjecture: by determining the distribution of prompts, we can derive the expected performance of the LLM on a given code benchmark.

We emphasize that while intuitive, this prompt-centric approach has remained unexplored in the literature. One possible reason for its neglect is the implicit assumption in prior works that explicit execution and verification through test suites are indispensable. However, as we demonstrate empirically, prompt distributions carry rich and sufficient information to reliably estimate model performance.

Thus, our approach not only introduces significant efficiency improvements but also provides a novel evaluation methodology that addresses the growing challenges of benchmark cost and data contamination in code-generation tasks.

### 3.2 Methodology

Our approach builds directly upon the key observations established in the motivation: under a fixed LLM and evaluation metric, the expected model performance is entirely determined by the distribution of input prompts (Equation 2). Based on this, we aim to estimate the performance of an LLM on a new benchmark (the *target prompt set*,  $P_t^{\text{target}}$ ) without executing code, by reweighting existing performance data from an annotated *source prompt set*,  $P_t^{\text{source}}$ .

To operationalize this idea, we adopt an importance sampling framework (**BIS**) as shown by Figure 1, which enables estimating expectations under a target distribution using samples drawn from a different, known source distribution.

**BIS** takes both the prompt for a target code generation task and an LLM as inputs. It employs an **Embedding Module** to obtain the prompt embedding. We model both the source and target prompt distributions using Importance-Weighted Autoencoders (IWAE) [3], a latent-variable model well-suited for approximating complex, multimodal distributions. The IWAE provides tractable estimates of the marginal likelihood of prompts under each distribution, from which importance weights can be derived. The **IWAE Module** is responsible for modeling the distribution  $P_t^{\text{target}}$  of target prompts by learning model  $\text{IWAE}_{\text{target}}$  from the feature vectors from **Embedding Module**.

Simultaneously, the other **IWAE Module** models the distribution  $P_t^{\text{source}}$  of prompts by learning model  $\text{IWAE}_{\text{source}}$  from the source benchmark. Concurrently, the **pre-loaded source benchmark** within our framework is evaluated on the LLM, yielding test results  $\mathcal{R}_{\text{source}} = \{r_1, r_2, \dots, r_N\}$ .

For each prompt sample  $\mathbf{t}_{\text{source}}^{(i)}$  within the source benchmark, we infer an *importance weight*  $w_i$  using aforementioned two trained IWAE models. The importance weight  $w_i$  can be obtained by the division of the marginal probability of  $\mathbf{t}_{\text{source}}^{(i)}$  in  $\text{IWAE}_{\text{target}}$  and the marginal probability of  $\mathbf{t}_{\text{source}}^{(i)}$  in  $\text{IWAE}_{\text{source}}$ :

$$w_i = \frac{\text{IWAE}_{\text{target}}(\mathbf{t}_{\text{source}}^{(i)})}{\text{IWAE}_{\text{source}}(\mathbf{t}_{\text{source}}^{(i)})} \quad (4)$$

The final prediction score  $\hat{y}_{\text{target}}$  for the target code generation task prompt is then computed as the weighted output based on these importance weights:

$$\hat{y}_{\text{target}} = \sum_{i=1}^N w_i \cdot f_b(P_{\text{LLM}}(\mathbf{t}_{\text{source}}^{(i)})) \quad (5)$$

where  $P_{\text{LLM}}$  denotes the LLM’s response function.

Specifically, our framework **BIS** comprises four modules:

**Embedding Module.** We employ a BERT model to extract high-dimensional embeddings from the target prompt as features. To ensure the extracted features are both expressive and concise, we utilize the embedding corresponding to the <CLS> token position within BERT as our feature representation.

**Source Benchmark Module.** This module contains a comprehensive code benchmark, incorporating a source prompt set from an alternative code task dataset along with their corresponding test suites and evaluation metrics. It also takes an LLM as input and outputs the LLM’s performance metrics on the source prompts.

**IWAE Module.** This module is responsible for fitting the posterior distribution over the features of the source prompts and target prompts, outputting corresponding IWAE models for the two prompt sets. We adopt the Importance Weighted Autoencoder (IWAE), a variant of VAE, as our distribution fitting method.

**Importance Weight Module.** This module takes as input two IWAE models and a source prompt set generated by the IWAE module. It computes an importance weight for each sample within the source prompt set. These weights are normalized and subsequently multiplied by the respective sample scores to yield the final prediction.

Within this module, the importance weight for each sample is calculated as the ratio of its marginal probability under the proposal distribution to its marginal probability under the target distribution, using the two IWAE models. To mitigate the potential instability arising from extreme weights—which can occur when high-probability regions of the proposal and target distributions are misaligned, leading to prediction dominance by a small number of samples—we employ weight clipping. This technique enhances the robustness of our predictions.

### 3.3 Unbiased Estimation Analysis of BIS

This section proves our framework is capable of providing unbiased estimation without implementing  $f_b$  for prompts sampled from  $P_t^{\text{target}}$ . There is one key assumption for the following theorem.

**Assumption 1:** The target distribution is absolutely continuous with respect to the source distribution. Specifically, for all prompts  $t$  satisfying  $P_t^{\text{target}}(t) > 0$ , we have  $P_t^{\text{source}}(t) > 0$ .

**THEOREM 3.1.** *Given source prompt distribution  $P_t^{\text{source}}$  and target prompt distribution  $P_t^{\text{target}}$ , under the above conditions, we can provide unbiased estimates for LLM’s score on target prompt set  $E_{P_t^{\text{target}}}(f_b \cdot P_{\text{LLM}})$  using prompts sampled from  $P_t^{\text{source}}$  instead of  $P_t^{\text{target}}$ .*

**PROOF.** We employ importance sampling to rewrite the target expectation in terms of the source distribution:

$$\begin{aligned} & E_{P_t^{\text{target}}}(f_b \cdot P_{\text{LLM}}) \\ &= \int_t \int_c f_b(c) \cdot P_{\text{LLM}}(c|t) \cdot P_t^{\text{target}}(t) dc dt \\ &= \int_t \int_c f_b(c) \cdot P_{\text{LLM}}(c|t) \cdot P_t^{\text{source}}(t) \cdot \frac{P_t^{\text{target}}(t)}{P_t^{\text{source}}(t)} dc dt \quad (6) \\ &= E_{P_t^{\text{source}}}(f_b \cdot P_{\text{LLM}} \cdot \frac{P_t^{\text{target}}(t)}{P_t^{\text{source}}(t)}) \end{aligned}$$

Assumption 1 ensures that the target dataset should have the statistical relationship to the source distribution, thereby keeping importance weights finite. The second equality follows from this assumption, which ensures the density ratio is well-defined wherever  $P_t^{\text{target}}(t) > 0$ . The final equality reinterprets the integral as

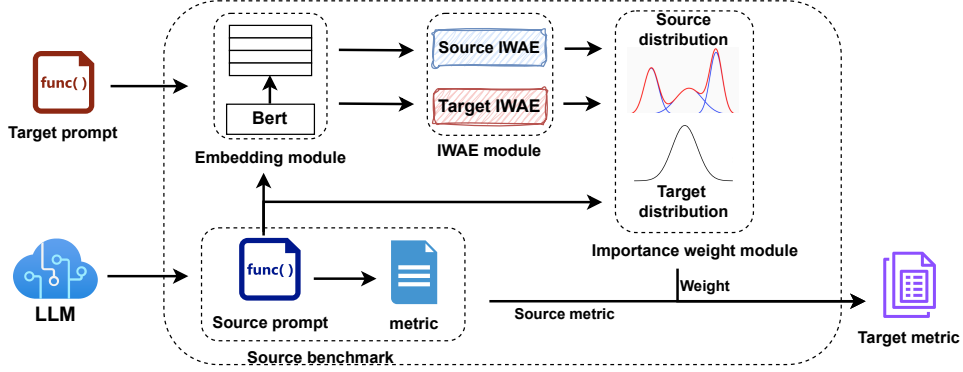


Figure 1: Workflow of our framework, *BIS*.

an expectation with respect to the source distribution, completing the importance sampling transformation.  $\square$

This result demonstrates that we only need prompts sampled from  $P_t^{\text{source}}$ , and there is no need to implement  $f_b$  for prompts sampled from  $P_t^{\text{target}}$ . The unbiased estimator is given by:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n f_b(c^{(i)}) \cdot \frac{P_t^{\text{target}}(t^{(i)})}{P_t^{\text{source}}(t^{(i)})}$$

where  $t^{(i)} \sim P_t^{\text{source}}$  and  $c^{(i)} \sim P_{\text{LLM}}(\cdot|t^{(i)})$ .

**3.3.1 Practical Feasibility and Effect of Assumption 1.** While Assumption 1 requires certain overlap between source and target prompt distributions, this condition is difficult to fully meet in cross-domain or cross-lingual settings, potentially resulting in increased variance of importance weights. Nevertheless, *BIS* still offers the following unique advantages:

- (1) **Cost-Effective Preliminary Screening.** When evaluating multiple candidate benchmarks, *BIS* offers a coarse performance ranking at low computational cost (requiring only prompt embedding calculations). This enables researchers to prioritize limited resources toward the most promising directions, avoiding the high expense of blindly constructing test suites.
- (2) **Quantitative Diagnosis of Distribution Shift.** Through statistical characteristics of importance weights (e.g., weight variance, extreme value ratio), *BIS* quantifies the alignment between source and target distributions. This provides data-driven support for subsequent evaluation strategy selection: prompts exhibiting high weight variance necessitate either source dataset augmentation or hybrid evaluation approaches.
- (3) **Significant Mitigation of Data Contamination Risks.** By eliminating reliance on complete test suites and reference answers, *BIS* inherently avoids the training data leakage issues plaguing traditional benchmarks. This holds substantial methodological significance in the era of large language models.

When importance weights exhibit extreme distributions, it indicates severe violation of Assumption 1. In such cases, we recommend: (1) augmenting the diversity of the source benchmark,

(2) performing traditional evaluation on samples with anomalous weights, or (3) employing robust estimation via weight truncation.

### 3.4 Quantitative Analysis: A formal analysis on error upper bound

In this section, we conduct a theoretical analysis of our framework to obtain a quantitative understanding of its generalization ability. We formalize the problem setup, define key symbols, and derive the loss function under standard assumptions.

**Problem Setup and Symbol Definitions.** We adopt the following assumptions and definitions:

- (1) **VAE Distribution:** IWAE learns a distribution  $\text{IWAE}_{t^{\text{target}}}(\mathbf{x})$  to approximate true distribution of source prompt set  $P_t^{\text{source}}$ . Another IWAE model is used to approximate target distribution  $\text{IWAE}_{t^{\text{target}}}(\mathbf{x})$  from  $P_t^{\text{target}}$ , which is true distribution of target prompt set.
- (2) **Objective:** Estimate the expectation  $\mu = \mathbb{E}_{\mathbf{x} \sim P_t^{\text{source}}} [f(\mathbf{x})]$ , where  $f(\cdot)$  is benchmark metric  $\mu$  can be written as:

$$\mu = \frac{1}{n} \sum_{i=1}^n \frac{P_t^{\text{target}}(\mathbf{x}_i)}{P_t^{\text{source}}(\mathbf{x}_i)} f(\mathbf{x}_i). \quad (7)$$

- (3) **Importance Sampling Estimator:** Given  $n$  samples  $\mathbf{x}_i \sim P_t^{\text{source}}$ , the IS estimator is defined as:

$$\hat{\mu}_{\text{IS}} = \frac{1}{n} \sum_{i=1}^n \frac{\text{IWAE}_{t^{\text{target}}}(\mathbf{x}_i)}{\text{IWAE}_{t^{\text{source}}}(\mathbf{x}_i)} f(\mathbf{x}_i). \quad (8)$$

- (4) **Prediction Error:** The absolute error (AE) of the estimator is:

$$\text{AE}(\hat{\mu}_{\text{IS}}) = |\hat{\mu}_{\text{IS}} - \mu|. \quad (9)$$

Inspired by the concept of chi-square divergence, we employ a function  $\varepsilon(x)$  to characterize the difference between the posterior distribution approximated by the IWAE and the actual distribution. Assuming sufficient proximity between the IWAE and the actual distribution with adequate training time and sample size, we subsequently derive an upper bound on the prediction error of our framework:

THEOREM 3.2. Let  $\text{IWAE}_{\text{target}} = p_t^{\text{target}}(1 + \varepsilon_{\text{target}}(x))$ , where  $|\varepsilon_{\text{target}}(x)| \ll 1$  and  $\text{IWAE}_{\text{source}} = p_t^{\text{source}}(1 + \varepsilon_{\text{source}}(x))$ , where  $|\varepsilon_{\text{source}}(x)| \ll 1$ .

We have:

$$\text{AE}(\hat{\mu}_{\text{IS}}) \leq |\mu(\text{sup}(\varepsilon_{\text{source}}(x) - \varepsilon_{\text{target}}(x)))| \quad (10)$$

Proof. We have

$$\text{AE}(\hat{\mu}_{\text{IS}}) = \left( \frac{1}{n} \sum_{i=1}^n \left( \frac{\text{IWAE}_{\text{target}}(\mathbf{x}_i)}{\text{IWAE}_{\text{source}}(\mathbf{x}_i)} - \frac{p_{\text{pr}}^{\text{target}}(\mathbf{x}_i)}{p_{\text{pr}}^{\text{source}}(\mathbf{x}_i)} \right) f(\mathbf{x}_i) \right)^{2 \times \frac{1}{2}} \quad (11)$$

$$= \left( \frac{1}{n} \sum_{i=1}^n \frac{p_t^{\text{source}}(\mathbf{x}_i)}{p_t^{\text{target}}(\mathbf{x}_i)} f(\mathbf{x}_i) \left( 1 - \frac{1 + \varepsilon_{\text{target}}(\mathbf{x}_i)}{1 + \varepsilon_{\text{source}}(\mathbf{x}_i)} \right) \right)^{2 \times \frac{1}{2}} \quad (12)$$

$$\approx \left( \frac{1}{n} \sum_{i=1}^n \frac{p_t^{\text{target}}(\mathbf{x}_i) f(\mathbf{x}_i)}{p_t^{\text{source}}(\mathbf{x}_i)} (\varepsilon_{\text{source}}(\mathbf{x}_i) - \varepsilon_{\text{target}}(\mathbf{x}_i)) \right)^{2 \times \frac{1}{2}} \quad (13)$$

$$\leq |\mu(\text{sup}(\varepsilon_{\text{source}}(\mathbf{x}_i) - \varepsilon_{\text{target}}(\mathbf{x}_i)))| \quad (14)$$

□

Since  $\mu$  depends solely on the actual prompt distribution and the dataset, we consider it a constant. Consequently, our results indicate that when the learning of the Importance-Weighted Autoencoder (IWAE) sufficiently approximates the target distribution, *the disparity in the fitting quality between the two IWAEs dominates the upper bound of the mean squared error for importance sampling.*

However, this conclusion holds under the prerequisite that *both* IWAEs achieve sufficiently close approximations to their respective target distributions. Under such conditions, since both IWAEs exhibit high fitting quality, the disparity between them remains minimal. Nevertheless, individual *out-of-distribution* samples may significantly elevate the loss upper bound. This occurs because the two IWAEs could produce markedly divergent responses to such samples. Therefore, we recommend maintaining diversity within the prompt sets in practical applications.

## 4 Evaluation

In this section, we conduct thorough experiments to validate the framework proposed in our study. We first describe our **experimental setup**, followed by the experimental design and results addressing the following five research questions:

- RQ1:** Can our framework effectively predict model correctness metrics?
- RQ2:** Can the predictive accuracy of our framework be generalized to other code-related metrics?
- RQ3:** Do alternative methods achieve superior performance compared to our framework?
- RQ4:** How do other factors (e.g., embedding dimensionality, prompt set size) influence the performance of our framework?

### 4.1 Setup

To obtain high-quality and sufficient data for our experiments, we collected approximately 8,528 data points encompassing **4 models** and **9 benchmarks**. Considering the potential impact of data contamination on our experiments, we employed open-source models

Table 2: Number of data points per benchmark

Benchmark	Datapoints
BigCodeBench	1,140
HumanEval	164
EvoEval_difficult	100
EvoEval_creative	100
EvoEval_subtle	100
EvoEval_combine	100
EvoEval_tool_use	100
EvoEval_verbose	164
EvoEval_concise	164
<b>Total</b>	<b>2,132</b>

released before the publication of these benchmarks, deliberately excluding all closed-source models. Furthermore, to mitigate the risk that low scores of the chosen open-source models on these two code benchmarks would render most problems inconsequential to the final results, we utilized the CodeLlama[27] family for our experiments. It is worth noting that although models within this family share a common lineage, architectural differences exist among them, including variations in the number of layers, hidden dimensions, attention mechanisms, and training data. The evaluation was conducted on the CodeLlama family of models, specifically including: CodeLlama-7B, CodeLlama-13B, CodeLlama-34B, and CodeLlama-70B.

As shown in Table 2, the benchmark suite comprises: BigCodeBench, HumanEval, EvoEval[36], which integrates 7 sub-benchmarks.

To mitigate class imbalance across datasets and better approximate real-world application scenarios, we merged HumanEval with EvoEval to form the **Evo** dataset comprising 992 samples. Concurrently, BigCodeBench was maintained as a standalone **bigcode** dataset containing 1,140 samples. In subsequent experiments, for each large language model evaluated, we implemented a reciprocal cross-prediction framework between these datasets:

- Performance on *Evo* predicts performance on *bigcode*
- Performance on *bigcode* predicts performance on *Evo*

This bidirectional evaluation protocol was systematically applied across all models. For multiple indicators spanning diverse value ranges, we employ min-max normalization to standardize all measurements within the unit interval  $[0, 1]$ . After normalization, the average score of the LLM on the target prompt set is regarded as our prediction target, while the prediction error is calculated by subtracting this prediction target from our weighted score. To collect the data, we rented 8 server with L20 GPUs with the bill 280\$.

### 4.2 RQ1: Predict Model Correctness Metrics

To represent correctness of code generated by LLM, we employ the **CodeBLEU score** [25], which integrates multiple similarity measures between the LLM-generated code and reference solutions. We conduct experiments to preliminarily validate the performance of our framework in predicting code correctness metrics. As shown in Table 3, our results demonstrate that when training on the *BigCode* dataset to predict the *Evo* dataset, the framework achieves an average absolute error rate of **1.05%**. Conversely, when training on the *Evo* dataset to predict the *BigCode* dataset, it achieves an average absolute error rate of **1.15%**. We also computed the

**Table 3: Error rates of CodeBLEU score across datasets and language models**

Source dataset	LLM	Error
BigCode	codellama-70b	0.011
	codellama-34b	0.007
	codellama-13b	0.003
	codellama-7b	-0.011
Avg absolute Error		<b>0.008</b>
Evo	codellama-70b	-0.016
	codellama-34b	-0.005
	codellama-13b	-0.015
	codellama-7b	-0.019
Avg absolute Error		<b>0.014</b>

**Table 4: Error rates of pass@1 score on CodeLlama-7b**

Source dataset	BigCode	Evo
Error	-0.022	0.021

pass@1 metric. Due to computational constraints, this metric was calculated exclusively on CodeLlama-7B. For all benchmarks used in our study, we executed CodeLlama-7B ten times to determine the pass@1 score. The result is shown in Table 4. These findings substantiate the robust predictive performance of our approach.

**Answer for RQ1:** Our framework can effectively predict LLM’s performance on CodeBLEU and pass@1 metric with low absolute error.

### 4.3 RQ2: Evaluation On Other Metric

In addition to accuracy metrics, recognizing the need for a more comprehensive evaluation of code generated by LLMs, several recent studies have begun incorporating alternative metrics that capture broader aspects of code quality—such as maintainability index [34] and security-related scores. In this work, we aim to extend the scope of importance sampling beyond simple accuracy measures to encompass these additional dimensions. We categorize the target metrics into two groups. The first group consists of **semantic-level** metrics, including cyclomatic complexity [7](CC), and security scores (SS), which are derived from the semantic properties of the generated code. We consider these metrics to be particularly critical in evaluating the practical utility and robustness of LLM-generated code.

The metrics are formally defined as follows: 1).  $CC = E - N + 2P$ , where  $E$  is edges,  $N$  is nodes in control flow graph, and  $P$  is connected components. 2).  $SS = \max(100 - \sum_{i=1}^n w_i \cdot c_i, 0)$ , where  $v_i$  represents vulnerability severity and  $c_i$  its confidence. In our framework, we choose bandit [16] as out static code auditory toolkit. The confidence and severity levels along with their corresponding weights are shown in Table 5.

These semantic metrics provide deeper insights into code quality compared to syntactic measures, evaluating complexity via CC, and vulnerability risks through SS. Our framework prioritizes these indicators as they directly reflect long-term software health and operational reliability.

**Table 5: Confidence level, severity and their corresponding weights in our framework**

Confidence level	weight	Severity	weight
High	1	High	50
Medium	0.6	Medium	30
Low	0.2	Low	10

Another category is **code-level** metrics, inspired by Halstead complexity [12]. We classify those metrics that are directly related to the characters of the source code itself into this level. Specifically, they include program length, volume, effort, and time. These metrics originate from the basic elements of source code and quantify the size, complexity, and development cost of the program. Although they are not as closely related to runtime behavior as semantic-level metrics, they still play an important role in evaluating structural characteristics and coding conventions, especially suitable for preliminary analysis and comparison of code quality.

**Table 6: Evaluation error on other metrics.**

Metric level	Source dataset	Bigcode				Evo			
	Codellama-	7b	13b	34b	70b	7b	13b	34b	70b
<b>Semantic</b>	SS	-0.039	-0.037	-0.040	-0.040	0.043	0.040	0.043	0.041
	CC	-0.046	-0.045	-0.028	-0.028	0.040	0.045	0.026	0.025
	Codebleu	0.011	0.007	0.003	0.011	-0.019	-0.016	-0.007	-0.015
<b>Code</b>	Length	-0.046	-0.045	-0.048	-0.031	0.079	0.076	0.090	0.107
	Volume	-0.035	-0.037	-0.037	-0.022	0.050	0.054	0.060	0.080
	Effort	-0.022	-0.022	-0.023	-0.018	0.029	0.030	0.028	0.050
	Time	-0.021	-0.022	-0.023	-0.018	0.026	0.028	0.032	0.045

Halstead complexity is a software measurement method proposed by Maurice Halstead, which is based on the number of operators and operands in a program. It measures the complexity and development workload of a program from the character level of the code. In our work, we classify a set of indicators inspired by Halstead complexity as code-level metrics, which mainly include the following four core indicators:

- Length (Program Length,  $L$ ): This refers to the total number of all operators and operands in the program. The calculation formula is:  $L = n_1 + n_2$ , where  $n_1$  represents the number of operators, and  $n_2$  represents the number of operands.
- Volume (Program Volume,  $V$ ): This indicates the total information content of the program, measured in "bits," reflecting the minimum mental effort required to understand the program. Its formula is:  $V = L \times \log_2(n_1 + n_2)$ .
- Effort (Effort,  $E$ ): This is used to estimate the degree of effort required by programmers to write the program, with units that can be understood as the number of basic operations. Its calculation method is:  $E = (\frac{n_1}{2} + n_2) \times V$ .
- Time (Time,  $T$ ): This is an estimate of the development time for the program, usually measured in seconds. The calculation formula is:  $T = \frac{E}{18}$ .

Our evaluation result on above metric is shown in Table 6. The experimental results demonstrate that our framework excels at predicting performance on semantic-level metrics compared to

code-level metrics. The max absolute errors (MAEs) for the predictions of security score and cyclomatic complexity were 4.3%, and 4.6%, respectively. In contrast, the performance on code-level metrics was relatively weaker, with MAEs of 10.7%, 8%, 5%, and 4.5% for the four indicators of code length, volume, effort, and time, respectively. We believe this discrepancy may be attributed to the direct influence of randomness at the token-level output of the LLM on code-level metrics. In contrast, semantic-level metrics encode higher-level information, making them more stable and reliable in terms of prediction performance.

**Answer for RQ2:** The predictive capability of our framework can be extended to various metrics; however, given the same dataset scale, our framework performs better on semantic-level metrics.

#### 4.4 RQ3: Comparison With Baselines

In this section, we compare our framework with alternative approaches and demonstrate that our method achieves state-of-the-art performance. We first investigate whether different fitting distributions can improve performance under the importance sampling framework. The baseline methods employed for comparison include: Gaussian Mixture Models[26] (GMMs), Restricted Boltzmann Machines[8] (RBMs), Conditional Maximum Entropy Models[21], and Variational Autoencoders (VAEs).

For the GMM implementation, we evaluate two distinct configurations: one with 8 components and another with 80 components. Regarding the RBMs and Conditional Maximum Entropy Models, considering the learning challenges in high-dimensional sparse embedding spaces, we utilize both the original 768-dimensional inputs and their reduced 128-dimensional counterparts. Our test results is shown in Table 7. We also consider the average value of absolute value (avg of abs) of error for each solution.

From the experimental results, we observe that our approach achieves the lowest prediction error among all compared schemes. Furthermore, it can be noted that the performance of both Restricted Boltzmann Machines and Conditional Maximum Entropy models remains virtually unchanged after applying PCA dimensionality reduction. This finding suggests that these two methods are capable of effectively learning from sparse samples in high-dimensional spaces for our specific task.

Next, we compare our framework with non-importance sampling approaches. Since our objective only requires obtaining importance weights for each sample from the source prompt set, we formulate this as a regression problem and employ commonly used regression models for comparison.

The baseline models we employ can be categorized into two groups: (1) machine learning models that provide interpretability, such as Random Forest Regression[28](RSR), Linear Regression[30](LR), Decision Tree Regression[37](DTR), and Ridge Regression[20](RR); and (2) deep learning-based models, for which we adopt Multilayer Perceptron[24] (MLP) and Recurrent Neural Network[29] (RNN) as our baselines. For the RNN implementation, instead of using only the embedding corresponding to the BERT-generated <cls> token as input, we sequentially feed all token embeddings generated by the model into the RNN. For the MLP, we directly train the model

**Table 7: Evaluation error on other method under importance sampling framework.**

Source	Bigcode				Evo				avg of abs
	7b	13b	34b	70b	7b	13b	34b	70b	
LLM									
GMM(80)	-0.265	-0.256	-0.221	-0.302	-0.268	-0.303	-0.333	-0.374	0.290
GMM(8)	0.426	-0.350	-0.373	-0.411	-0.400	-0.403	-0.405	-0.511	0.328
RBM(768)	-0.002	-0.019	-0.032	0.012	0.008	0.025	0.038	-0.007	0.018
RBM(128)	-0.002	-0.019	-0.032	0.012	0.008	0.025	0.038	-0.007	0.018
MaxEnt(768)	-0.005	-0.022	-0.034	0.009	0.005	0.0224	0.034	-0.009	0.017
MaxEnt(128)	-0.005	-0.022	-0.034	0.009	0.005	0.022	0.034	-0.009	0.017
VAE	-0.009	-0.026	-0.020	0.006	-0.032	-0.003	0.0028	-0.024	0.015
<b>Ours</b>	0.011	0.007	0.003	0.011	-0.019	-0.016	-0.007	-0.015	<b>0.011</b>

**Table 8: Evaluation error on other method that are not under importance sampling framework. Notation: Ex=Explainable, Unex=Unexplainable**

	Source dataset	Bigcode				Evo				avg of abs
		7b	13b	34b	70b	7b	13b	34b	70b	
<b>Ex</b>	RSR	-0.033	-0.045	-0.060	0.015	0.019	0.019	0.034	-0.029	0.031
	LR	-0.005	-0.022	-0.034	<b>0.009</b>	<b>0.005</b>	0.022	0.034	-0.009	0.017
	DTR	-0.032	-0.054	-0.065	-0.026	0.019	0.017	0.044	-0.018	0.035
	RR	<b>-0.005</b>	-0.022	-0.034	0.009	0.005	0.022	0.034	<b>-0.009</b>	0.017
	<b>Ours</b>	0.011	<b>0.007</b>	<b>0.003</b>	0.011	-0.019	<b>-0.016</b>	<b>-0.007</b>	-0.015	<b>0.011</b>
<b>Unex</b>	RNN	0.285	0.278	0.317	0.278	0.331	0.318	0.324	0.314	0.298
	MLP	-0.108	-0.133	-0.138	-0.132	0.061	0.096	0.093	0.060	0.105

on the source prompt set and subsequently employ the trained MLP to predict the performance of samples in the target prompt set.

Our experimental results are presented in Table 8. As can be observed, the interpretable machine learning-based approaches significantly outperform the non-interpretable deep learning-based methods. This performance gap may be attributed to the fact that MLP and RNN architectures are prone to either overfitting or underfitting phenomena. Furthermore, our proposed method still achieves the best performance among all compared solutions.

**Answer for RQ3:** Our solution outperforms all baseline methods in terms of prediction accuracy. We also observe that machine learning and statistical learning-based approaches perform better than deep learning-based methods, which may indicate that there is still room for development in applying deep learning models to this task.

#### 4.5 RQ4: Ablation Study

In this section, we investigate whether other factors in our framework could potentially affect its performance. Specifically, we focus on examining four key aspects: (1) Feature dimensionality. (2) The size of the prompt set. (3) The number of importance-weighted autoencoder samples. (4) The percentage ratios of truncated weight.

*The feature dimensionality.* As the embedding dimensionality continues to increase, the sparse distribution of samples in high-dimensional space may lead to the curse of dimensionality. However, dimensionality reduction of samples could potentially result in information loss within the embeddings. We employed Principal Component Analysis[1] (PCA) for dimensionality reduction. We



Table 9: Evaluation error of our framework with PCA.

Source	Dimension	576	384	192
BigCode	codellama-70b	0.013	0.017	0.016
	codellama-34b	0.006	0.093	0.011
	codellama-13b	0.009	0.104	0.019
	codellama-7b	0.019	-0.021	0.022
Evo	codellama-70b	-0.017	-0.039	-0.016
	codellama-34b	-0.007	-0.016	-0.007
	codellama-13b	-0.014	0.020	-0.015
	codellama-7b	-0.021	-0.003	-0.022

Table 10: Evaluation error of our framework with linear layer.

Source	Dimension	576	384
BigCode	codellama-70b	-0.410	-0.410
	codellama-34b	-0.264	-0.265
	codellama-13b	-0.349	-0.350
	codellama-7b	-0.290	-0.291
Evo	codellama-70b	-0.391	-0.392
	codellama-34b	-0.275	-0.276
	codellama-13b	-0.319	-0.320
	codellama-7b	-0.291	-0.292

subsequently examined the performance of our method in different dimensionality. Experimental result is shown in Table 9.

The experimental results demonstrate that the absolute error value exhibits a generally increasing trend with decreasing dimensionality. While this relationship is not strictly monotonic, a positive correlation can be observed between the dimension reduction level and the magnitude of the final prediction error. It is also worth noting that although linear layers are a common dimensionality reduction approach, employing linear layers for dimensionality reduction in our framework leads to a significant increase in error, as shown in Table 10.

*The number of importance-weighted autoencoder samples.* In IWAE, appropriately increasing the number of samples helps IWAE better approximate the target distribution. However, excessively increasing the sample size may lead to large variance, thereby affecting test stability. Therefore, adjusting the number of samples in IWAE can help strike a balance between stability and accuracy. We experimentally investigate the relationship between our framework’s performance and the number of samples, as shown in the Table 11. It can be observed that when the sampling number equals 1, cases may occur where the absolute error exceeds 4%. On the other hand, with sampling numbers more than 25, instances of absolute errors surpassing 3.5% were detected. This empirical evidence substantiates our prior hypothesis that both excessively large and insufficiently small sampling numbers may compromise the stability of prediction outcomes.

*The percentage ratios of truncated weight.* During importance sampling, when the proposal distribution  $q(x)$  exhibits limited overlap with the high-probability regions of the target distribution  $p(x)$ , extreme importance weights may occur. This can lead to a situation where a minority of samples dominate the prediction, potentially compromising the stability of the estimation. To enhance

Table 11: Evaluation error on our method with different number of samples in IWAE.

Source	Bigcode				Evo			
Codellama	7b	13b	34b	70b	7b	13b	34b	70b
100	-0.005	-0.028	-0.031	0.017	0.006	0.025	0.031	-0.012
50	-0.026	-0.025	-0.022	0.014	<b>0.002</b>	-0.026	0.035	0.009
25	<b>0.004</b>	-0.022	-0.039	0.014	-0.008	0.011	0.028	-0.014
<b>10</b>	0.011	<b>0.007</b>	<b>0.003</b>	<b>0.011</b>	-0.019	-0.016	-0.007	-0.015
5	0.004	-0.011	0.009	0.017	-0.016	<b>0.010</b>	<b>0.005</b>	-0.014
1	-0.003	-0.041	-0.018	-0.050	-0.010	0.009	0.029	-0.038

Table 12: Evaluation error on our method with different percentile of truncated weight in IWAE.

Source	Bigcode				Evo			
Codellama	7b	13b	34b	70b	7b	13b	34b	70b
1	0.037	0.019	-0.013	0.049	-0.024	<b>0.011</b>	0.010	<b>-0.010</b>
0.95	-0.021	-0.022	-0.023	-0.018	0.026	0.028	0.032	0.015
<b>0.9</b>	<b>0.011</b>	<b>0.007</b>	<b>0.003</b>	<b>0.011</b>	<b>-0.019</b>	-0.016	<b>-0.007</b>	-0.015
0.85	-0.022	-0.022	-0.023	-0.017	-0.028	0.028	0.029	0.052
0.8	-0.021	-0.022	-0.023	-0.018	0.025	0.028	0.030	0.053

the robustness of our predictions, we employ a percentile-based weight truncation strategy.

Specifically, we truncate all weights exceeding the  $x$ -th percentile to the value of the  $x$ -th percentile weight, followed by a normalization step to prevent underestimation. Let  $w_{(1)}, w_{(2)}, \dots, w_{(n)}$  denote the ordered importance weights, and  $w_{(k)}$  be the weight at the  $x$ -th percentile. The truncated weights  $\tilde{w}_i$  are computed as:  $\tilde{w}_i = \min(w_i, w_{(k)})$ . The normalized weights are then given by:  $\tilde{w}_i = \frac{\tilde{w}_i}{\sum \tilde{w}_j}$ .

It should be noted that excessive weight truncation may introduce bias into the original importance sampling estimator, as the unbiasedness may no longer hold after truncation. Therefore, the choice of the truncation percentile  $x$  requires careful consideration to balance between variance reduction and bias introduction. We investigate the relationship between the percentile of truncated weights and the final performance in our framework, as presented in the Table 12. From the table, it can be observed that extreme values tend to emerge when the truncation weight percentile is either too large or too small. We select  $x = 0.9$  as the truncation percentile.

*The size of the prompt set.* Since our framework approximates the expectation by summing the sample scores, the number of samples may also become a factor affecting the performance of our framework. An insufficiently large prompt set size could lead to unstable predictions. To verify this hypothesis, we design five different settings for two prompt sets: *full*, 700, 500, 300, and 100, representing using the complete dataset or randomly sampled subsets of 700, 500, 300, and 100 data points respectively. Under these five configurations, we conduct mutual prediction between two datasets to observe how the prediction results vary with different sample sizes.

We present the testing results in Table 13. Each cell in the table contains two numbers separated by a forward slash (/). The value on the left side of the slash originates from the BigCode-predicts-Evo dataset, while the value on the right represents the inverse scenario. As demonstrated, the reduction in size for either the target

Table 13: Bigcode ↔ EVO Score Pairs.

CodeLlama	Setting	Full	700	500	300	100
70b	Full	-0.111 / -0.014	0.028 / 0.016	0.005 / 0.0022	-0.017 / -0.012	0.015 / -0.028
34b		0.003 / -0.007	-0.008 / 0.054	-0.061 / 0.032	-0.054/0.037	-0.029/0.019
13b		-0.011 / -0.010	-0.012 / 0.053	0.030 / 0.020	-0.058 / 0.019	-0.040 / 0.009
7b		0.011 / -0.019	0.015 / 0.031	-0.004 / 0.007	-0.033 / -0.001	-0.019 / -0.006
70b	700	-0.024 / -0.042	-0.006 / -0.013	0.003 / 0.006	0.034 / -0.011	0.029 / -0.006
34b		-0.062 / 0.042	-0.034 / 0.028	-0.054 / 0.062	0.004 / 0.049	-0.054 / 0.051
13b		-0.035 / 0.014	-0.016 / 0.028	-0.042 / 0.062	0.003 / 0.042	-0.003 / 0.058
7b		-0.038 / -0.005	-0.013 / 0.006	-0.023 / 0.022	0.021 / 0.013	0.038 / 0.025
70b	500	-0.028 / -0.009	-0.023 / -0.054	0.001 / -0.026	-0.006 / -0.039	-0.025 / -0.002
34b		-0.042 / 0.078	-0.060 / -0.026	-0.038 / 0.034	-0.018 / 0.021	-0.055 / 0.049
13b		0.014 / 0.018	-0.045 / -0.009	-0.032 / 0.034	-0.039 / 0.038	-0.024 / 0.060
7b		-0.005 / 0.018	-0.029 / -0.016	-0.017 / 0.003	0.002 / -0.010	-0.024 / -0.031
70b	300	-0.056 / -0.054	-0.007 / -0.033	0.012 / -0.017	-0.033 / -0.039	-0.050 / -0.069
34b		-0.080 / 0.016	-0.031 / 0.024	-0.034 / 0.047	-0.083 / 0.014	-0.064 / 0.011
13b		-0.030/-0.030	-0.037/-0.001	0.040/-0.005	-0.019 / -0.056	-0.012 / -0.035
7b		-0.006 / -0.031	-0.012 / 0.053	0.030 / 0.020	-0.058 / 0.019	-0.040 / 0.009
70b	100	-0.094 / -0.009	0.033 / 0.049	-0.107 / -0.038	-0.104 / -0.030	-0.062 / -0.079
34b		-0.118 / 0.045	-0.090 / 0.003	-0.148 / -0.011	-0.026 / 0.037	-0.173 / 0.058
13b		-0.098 / 0.040	0.017 / 0.030	-0.07 / -0.005	-0.071 / 0.033	0.100 / 0.092
7b		-0.086 / -0.020	-0.090 / 0.029	-0.045 / -0.034	-0.096 / -0.071	0.147 / -0.090

or source prompt set leads to a significant performance degradation. Although our experimental dataset sizes are typically on the order of thousands, we recommend using larger datasets in practical applications.

**Answer for RQ4:** We observe that the choice of dimensionality reduction method and the resulting reduced dimension both affect the performance of our framework. Statistical approaches, such as PCA, demonstrate significantly better results compared to neural network-based methods. Additionally, as the reduced dimension becomes lower, the error of our framework increases. In practical applications, truncating the weight percentiles and the number of IWAE samples also influence the performance of our framework.

## 5 Discussion

Our work represents the first exploration of a benchmark-free approach in the domain of code tasks. In this section, we identify several promising avenues for future research:

**Anti Data contamination:** Data contamination[9] occurs when open-source test suites and their accompanying reference solutions are incorporated into the LLM’s training data, causing validity of benchmark degrade over time[18]. This issue is not unique to code benchmarks[38] but in many other areas[11, 19]; This phenomenon can lead to artificially inflated performance on the test set, with another study indicating that LLM scores on contaminated samples can be up to five times higher than on uncontaminated samples[38]. Since our framework does not need test suite and reference answer, we can significantly reduce the risk of data contamination.

**Exploration of Cross-Scenario and Cross-Language Code Testing:** Although our evaluation utilized two distinct benchmarks, they correspond to largely consistent coding scenarios. Testing across significantly divergent scenarios and languages remains

unexplored in this work. We posit this as a highly valuable future direction. Another critical aspect involves scenarios where prompt distributions exhibit minimal overlap, causing the computed weights might disproportionately favor a minority of samples. Investigating methods to mitigate this issue is crucial for extending the applicability of our framework to broader contexts.

**Leveraging Closed-Source Benchmarks for Anti-Cheating Verification:** As discussed, existing benchmarks are often compromised by data contamination. However, employing closed-source benchmarks within our framework—where the source benchmark remains inaccessible for learning—could enable the reverse testing of potential cheating by existing LLMs on various benchmarks. This is feasible because, when the source benchmark is free from data contamination, our framework can correspondingly compute the actual performance of LLMs under uncontaminated conditions.

## 6 Conclusion

This paper introduces an innovative approach **BIS** that combines *Importance Sampling* with *Importance Weighted Autoencoders (IWAE)* to evaluate the performance of LLM on code generation tasks without requiring human annotators or LLMs as judges. This approach provides new perspectives for automatic LLM evaluation and can reduce the cost of high-quality benchmark construction.

Future work will focus on further optimizing the IWAE architecture and exploring better approaches to handle out-of-distribution samples. Meanwhile, considering that real-world code generation tasks often involve complex interactive processes, how to evaluate LLM performance in dynamic environments in real-time represents another valuable direction for in-depth investigation.

In conclusion, this study provides a novel pathway for the automated and efficient evaluation of LLMs in code generation, contributing to advancements in the field of artificial intelligence. With technological progress, we anticipate seeing more innovative applications emerge based on this framework.

## 7 Acknowledgement

This research is supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

## References

- [1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.
- [2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.
- [3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2015. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519* (2015).
- [4] Ligu Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, et al. 2024. A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498* (2024).
- [5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [6] Yaxin Du, Yuzhu Cai, Yifan Zhou, Cheng Wang, Yu Qian, Xianghe Pang, Qian Liu, Yue Hu, and Siheng Chen. 2025. SWE-Dev: Evaluating and Training Autonomous Feature-Driven Software Development. *arXiv preprint arXiv:2505.16975* (2025).
- [7] Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. 2016. Cyclomatic complexity. *IEEE software* 33, 6 (2016), 27–29.
- [8] Asja Fischer and Christian Igel. 2012. An introduction to restricted Boltzmann machines. In *Iberoamerican congress on pattern recognition*. Springer, 14–36.
- [9] Yujian Fu, Ozlem Uzuner, Meliha Yetisgen, and Fei Xia. 2024. Does Data Contamination Detection Work (Well) for LLMs? A Survey and Evaluation on Detection Assumptions. *arXiv preprint arXiv:2410.18966* (2024).
- [10] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*. PMLR, 2052–2062.
- [11] Vipul Gupta, David Pantoja, Candace Ross, Adina Williams, and Megan Ung. 2024. Changing answer order can decrease mmmlu accuracy. *arXiv preprint arXiv:2406.19470* (2024).
- [12] T Hariprasad, G Vidhyagarani, K Seenu, and Chandrasegar Thirumalai. 2017. Software complexity analysis using halstead metrics. In *2017 international conference on trends in electronics and informatics (ICEI)*. IEEE, 1109–1113.
- [13] Kush Jain, Gabriel Synnaeve, and Baptiste Rozière. 2024. Testgeneval: A real world unit test generation and test completion benchmark. *arXiv preprint arXiv:2410.00752* (2024).
- [14] Frederick James. 1980. Monte Carlo theory and practice. *Reports on progress in Physics* 43, 9 (1980), 1145.
- [15] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).
- [16] DA Kapustin, VV Shvyrov, and TI Shulika. 2023. Static analysis of corpus of source codes of python applications. *Programming and Computer Software* 49, 4 (2023), 302–309.
- [17] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).
- [18] Changmao Li and Jeffrey Flanigan. 2024. Task contamination: Language models may not be few-shot anymore. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 18471–18480.
- [19] Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. 2023. Tinygsm: achieving > 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241* (2023).
- [20] Gary C McDonald. 2009. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 1 (2009), 93–100.
- [21] Ryan McDonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon Mann. 2009. Efficient large-scale distributed training of conditional maximum entropy models. *Advances in neural information processing systems* 22 (2009).
- [22] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems* 29 (2016).
- [23] Lucas Pinheiro Cinelli, Matheus Araújo Marins, Eduardo Antônio Barros da Silva, and Sérgio Lima Netto. 2021. Variational autoencoder. In *Variational methods for machine learning with applications to deep networks*. Springer, 111–149.
- [24] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastrokakis. 2009. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems* 8, 7 (2009), 579–588.
- [25] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *arXiv:2009.10297* [cs.SE] <https://arxiv.org/abs/2009.10297>
- [26] Douglas Reynolds. 2015. Gaussian mixture models. In *Encyclopedia of biometrics*. Springer, 827–832.
- [27] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).
- [28] Mark R Segal. 2004. Machine learning benchmarks and random forest regression. (2004).
- [29] Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (2020), 132306.
- [30] Xiaogang Su, Xin Yan, and Chih-Ling Tsai. 2012. Linear regression. *Wiley Interdisciplinary Reviews: Computational Statistics* 4, 3 (2012), 275–294.
- [31] Surya T Tokdar and Robert E Kass. 2010. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics* 2, 1 (2010), 54–60.
- [32] Masatoshi Uehara, Chengchun Shi, and Nathan Kallus. 2022. A review of off-policy evaluation in reinforcement learning. *arXiv preprint arXiv:2212.06355* (2022).
- [33] Tim Van Erven and Peter Harremoës. 2014. Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory* 60, 7 (2014), 3797–3820.
- [34] Kurt D Welker. 2001. The software maintainability index revisited. *CrossTalk* 14 (2001), 18–21.
- [35] Chanuka Wijayakoon, Hai Dong, HMN Bandara, Zahir Tari, and Anurag Sooin. 2025. Legal Compliance Evaluation of Smart Contracts Generated By Large Language Models. *arXiv preprint arXiv:2506.00943* (2025).
- [36] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. 2024. Top leaderboard ranking= top coding proficiency, always? evoeval: Evolving coding benchmarks via llm. *arXiv preprint arXiv:2403.19114* (2024).
- [37] Min Xu, Pakorn Watanachaturaporn, Pramod K Varshney, and Manoj K Arora. 2005. Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment* 97, 3 (2005), 322–336.
- [38] Xin Zhou, Martin Weyssow, Ratnadira Widayarsi, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. 2025. LessLeak-Bench: A First Investigation of Data Leakage in LLMs Across 83 Software Engineering Benchmarks. *arXiv preprint arXiv:2502.06215* (2025).
- [39] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).