$\mathcal{B}^4$: Towards Optimal Assessment of Plausible Code Solutions with Plausible Tests

Mouxiang Chen
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, China
chenmx@zju.edu.cn

Zhongxin Liu*†
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, China
liu_zx@zju.edu.cn

He Tao
Zhejiang University
Hangzhou, China
tao_he@zju.edu.cn

Yusu Hong
Zhejiang University
Hangzhou, China
yusuhong@zju.edu.cn

David Lo
Singapore Management University
Singapore, Singapore
davidlo@smu.edu.sg

Xin Xia
Zhejiang University
Hangzhou, China
xin.xia@acm.org

Jianling Sun
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, China
sunjl@zju.edu.cn

ABSTRACT

Selecting the best code solution from multiple generated ones is an essential task in code generation, which can be achieved by using some reliable validators (*e.g.*, developer-written test cases) for assistance. Since reliable test cases are not always available and can be expensive to build in practice, researchers propose to automatically generate test cases to assess code solutions. However, when both code solutions and test cases are plausible and not reliable, selecting the best solution becomes challenging. Although some heuristic strategies have been proposed to tackle this problem, they lack a strong theoretical guarantee and it is still an open question whether an optimal selection strategy exists. Our work contributes in two ways. First, we show that within a Bayesian framework, the optimal selection strategy can be defined based on the posterior probability of the observed passing states between solutions and tests. The problem of identifying the best solution is then framed as an integer programming problem. Second, we propose an efficient approach for approximating this optimal (yet uncomputable) strategy, where the approximation error is bounded by the correctness of prior knowledge. We then incorporate effective prior knowledge to tailor code generation tasks. Both theoretical and empirical studies confirm that existing heuristics are limited in selecting the best solutions with plausible test cases. Our proposed approximated optimal strategy $\mathcal{B}^4$ significantly surpasses existing heuristics in selecting code solutions generated by large language models (LLMs) with LLM-generated tests, achieving a relative performance improvement by up to 50% over the strongest heuristic and 246% over the random selection in the most challenging scenarios. Our code is publicly available at https://github.com/ZJU-CTAG/B4.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → **Software design engineering**.

KEYWORDS

Code Generation, Software Engineering, Large Language Models

ACM Reference Format:
Mouxiang Chen, Zhongxin Liu, He Tao, Yusu Hong, David Lo, Xin Xia, and Jianling Sun. 2024. $\mathcal{B}^4$: Towards Optimal Assessment of Plausible Code Solutions with Plausible Tests. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695536

1 INTRODUCTION

Code generation is an important task in the field of software engineering [23], aiming to generate code solutions that satisfy the given requirement. In practice, we often face the problem of selecting the best code solution from multiple generated alternatives [7, 22]. A common practice is using some validators (*e.g.*, test cases) to assess the validity of each solution and choose the best one [5, 33, 36, 46]. However, in real-world scenarios, reliable test

*Corresponding Author
†Also with Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10…$15.00
https://doi.org/10.1145/3691620.3695536

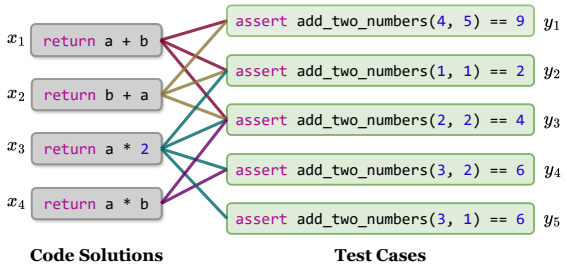


Figure 1: A simple example showing the problem "return the sum of a and b". A link between a generated code solution and a generated test case indicates that the solution passes the test. How can we select the best code solution solely based on these links?

cases are not always available. Developing and maintaining reliable test cases can also be resource-intensive and laborious. With advancements in deep learning and large language models (LLMs), using auto-generated test cases has gained popularity among researchers and practitioners [20, 26, 27, 35]. Unfortunately, selecting code solutions based on these potentially unreliable tests poses significant challenges, since incorrect test cases can disrupt our decision-making. Fig. 1 provides an example, where selecting the best code solution becomes difficult since the fourth and fifth test cases are incorrect.

Few studies systematically explore how to assess *plausible* code solutions and select the best using *plausible* test cases. Under the assumption that the generated test cases are (mostly) correct, some existing research favors the solutions that pass the most test cases [18, 19, 22, 33]. However, this strategy is ineffective when test cases are merely plausible, indicated by our theoretical analysis (see Section 4). Other research addresses this challenge by designing clustering-based heuristic rules. For instance, Shi *et al.* [36] and Li *et al.* [22] clustered code solutions based on test outputs, and selected the solutions from the largest cluster. Chen *et al.* [5] similarly clustered code solutions based on the passed test cases, and selected the best cluster according to the count of solutions and passed test cases in each. However, these heuristics rely on human-designed rules and lack strong theoretical foundations, leading to potentially suboptimal performance. To the best of our knowledge, the optimal selection strategy for this problem is still an open question.

In this work, we aim to develop a general framework to define and compute the optimal selection strategy. We first show that under a Bayesian framework, the optimal strategy can be defined based on the posterior probability of the observed passing states between solutions and tests. The problem of identifying the optimal strategy is then framed as an integer programming problem. Under a few assumptions, this posterior probability can be further expanded into four integrals, which cannot be directly computed due to four unknown prior distributions. We then leverage Bayesian statistics techniques to deduce a computable form for approximating this posterior probability and optimize the integer programming from exponential to polynomial complexity. The approximation error is bounded by the correctness of prior knowledge. Based on this bound, we investigate two effective priors and incorporate them into our framework to enhance code generation performance. Given that the approximated optimal strategy involves scoring code solutions with four Beta functions [10], we refer to it as $\mathcal{B}^4$.

Based on our developed framework, we further provide a theoretical analysis to compare $\mathcal{B}^4$ with existing heuristics. We observe that some heuristics require sufficient correct test cases, while others necessitate a higher probability of correct code solutions, as confirmed by subsequent simulated experiments. In real-world applications involving selecting LLM-generated code solutions with LLM-generated test cases, $\mathcal{B}^4$ significantly outperforms existing heuristics across five LLMs and three benchmarks.

In summary, our paper makes the following contributions:

- **Optimal Strategy.** We systematically address the challenging problem of selecting plausible code solutions with plausible tests and establish an optimal yet uncomputable strategy.
- **Technique.** We derive an efficiently computable approach to approximate the uncomputable optimal strategy with an error bound. While our framework is broadly applicable, we adapt it to code generation by incorporating two effective priors.
- **Theoretical Study.** Using our framework, we explore the conditions under which existing heuristics are effective or ineffective and compare them to the approximated optimal strategy $\mathcal{B}^4$.
- **Empirical Study.** We empirically evaluate our selection strategy with five code LLMs on three benchmarks. Experimental results show that our strategy demonstrates up to a 12% average relative improvement over the strongest heuristic and a 50% improvement in the most challenging situations where there are few correct solutions.

2 PRELIMINARIES

Notations. We use bold lowercase letters to denote vectors (*e.g.*, $\mathbf{x}$ and $\mathbf{y}$), bold uppercase letters to denote matrices (*e.g.*, $\mathbf{E}$), and thin letters to denote scalars (*e.g.*, x and y). We also use thin uppercase letters to denote random variables (*e.g.*, X, Y, and E). $\boldsymbol{e}_i$ denotes the i-th row in matrix $\mathbf{E}$. The index set $[N]$ denotes $\{1, 2, \cdots, N\}$. $\{0,1\}^N$ denotes a length-N binary vector, and $\{0,1\}^{N\times M}$ denotes an $N \times M$ binary matrix.

2.1 Problem Definition

Code generation is a crucial task in software engineering, which aims at automatically generating a code solution x from a given context c. We explore the selection of the best code solution from N code solutions generated based on c, with M test cases (also generated based on c) to aid this selection. It is worth noting that the correctness of both code solutions and test cases is *plausible*; they might be either correct or incorrect, which is unobserved however. All we can observe is a matrix $\mathbf{E} = \{e_{ij}\}_{N\times M} \in \{0,1\}^{N\times M}$ where $e_{ij} = 1$ indicates the i-th code solution passes the j-th test case, and 0 indicates failure. We term $\mathbf{E}$ as **passing matrix**, and e_{ij} as **passing state**.

Let $\mathbf{x} = \{x_1, \cdots, x_N\} \in \{0,1\}^N$ denote the ground-truth correctness of each code solution (unknown to us), in which 1 denotes *correct* and 0 denotes *incorrect*. We assume at least one code solution is correct since designing a selection strategy would be meaningless without any correct code. Similarly, the correctness of each test case is denoted by $\mathbf{y} = \{y_1, ..., y_M\} \in \{0,1\}^M$. We assume that all correct code solutions share identical *functionality* and all tests

are not flaky, meaning that all solutions pass the same test cases on the same context c. This can be formulated as the following assumption.

Assumption 1 (Consistency). *For all $i, j \in [N]$, if $x_i = 1$ and $x_j = 1$, then $\mathbf{E}$ should satisfy:*

$$\boldsymbol{e}_i = \boldsymbol{e}_j \quad (i.e., e_{ik} = e_{jk}, \quad \forall k \in [M]).$$

Furthermore, the correctness of test cases $\mathbf{y}$ corresponds to the passing states of the correct code solutions. Formally, if $x_i = 1, i \in [N]$, then:

$$\mathbf{y} = \boldsymbol{e}_i \quad (i.e., y_k = e_{ik}, \quad \forall k \in [M]).$$

This assumption indicates that $\mathbf{E}$ and $\mathbf{y}$ should be *consistent* with $\mathbf{x}$. Intuitively, $\mathbf{E}$ should satisfy that the rows corresponding to the correct code solutions are the same. $\mathbf{y}$ is defined based on these rows. For example, in Fig. 1, we have $\mathbf{x} = \{1, 1, 0, 0\}$, $\mathbf{y} = \{1, 1, 1, 0, 0\}$, and

$$\mathbf{E} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}. \tag{1}$$

In this paper, our goal is to use $\mathbf{E}$ to assess the correctness of code solutions and select the best one by recovering $\mathbf{x}$ and $\mathbf{y}$ from $\mathbf{E}$. Following Chen *et al.* [5], we do not rely on any specific details of the code solutions or test cases in this paper.

2.2 Existing Heuristics

In this section, we briefly review two representative heuristic methods for addressing this problem. The first family of methods MaxPass [18, 19, 22, 33] always rewards passing test cases. The best code solution can be selected by counting the passed cases, *i.e.*,

$$\text{Select code solution } i, \text{ where } i = \arg\max_{i \in [N]} \sum_{j=1}^{M} e_{ij}.$$

The other family of methods examines the consensus between code solutions and test cases, and clusters the code solutions with the same functionality [5, 22, 36]. One of the most representative methods is CodeT [5]. It divides the code solutions into K disjoint subsets based on functionality: $S^x = \{S_1^x, \cdots, S_K^x\}$, where each set S_i^x ($i \in [K]$) consists of code solutions that pass the same set of test cases, denoted by S_i^y. The tuple (S_i^x, S_i^y) is termed a *consensus set*. Taking Fig. 1 as an example, there are three consensus sets: $(\{x_1, x_2\}, \{y_1, y_2, y_3\})$, $(\{x_3\}, \{y_2, y_3, y_4, y_5\})$ and $(\{x_4\}, \{y_3, y_4\})$.

CodeT proposes that a consensus set containing more code solutions and test cases indicates a higher level of consensus, and thus the more likely they are correct. Therefore, CodeT scores each consensus set based on the capacity and selects the code solutions associated with the highest-scoring set, *i.e.*,

$$\text{Select code solutions } i \in S_k, \text{ where } k = \arg\max_{k \in [K]} |S_k^x| \cdot |S_k^y|.$$

Similarly, other clustering methods, such as MBR-exec [36] and AlphaCode-C [22], also cluster the code solutions based on test cases, but only score each set by the number of code solutions $|S_k^x|$. We focus our analysis on CodeT as it was verified to outperform other existing scoring strategies [5].

In this study, we develop a systematic analysis framework, to evaluate the effectiveness of these heuristics and address the following research questions (RQs):

- *RQ1: Given a passing matrix $\mathbf{E}$, what constitutes the optimal selection strategy?*
- *RQ2: Is this optimal strategy computable?*
- *RQ3: Can a practical algorithm be developed to compute (or approximate) this optimal strategy efficiently?*
- *RQ4: Under what conditions do existing heuristics not work, based on our developed analysis framework?*
- *RQ5: If the answer to RQ3 is true, how does the computable (or approximated) optimal strategy compare to these heuristics?*

3 METHODOLOGY

In this section, we outline our proposed methodology to address this problem.

3.1 Optimal Strategy

We use $X = \{X_1, \cdots, X_N\} \in \{0,1\}^N$, $Y = \{Y_1, \cdots, Y_M\} \in \{0,1\}^M$, and $E = \{E_{ij}\}_{N \times M} \in \{0,1\}^{N \times M}$ to denote random variables of code solutions' and tests' correctness, and the passing matrix, respectively. Note that all X, Y, and E depend on the same context C, which we omit for ease of notation. A strategy's estimation for X and Y is denoted by $\hat{\mathbf{x}} = \{\hat{x}_1, \cdots, \hat{x}_N\}$ and $\hat{\mathbf{y}} = \{\hat{y}_1, \cdots, \hat{y}_M\}$. To answer *RQ1*, our goal is to find *the most probable* $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ given an observation $E = \mathbf{E}$. This motivates us to design the optimal strategy by modeling $P(X, Y \mid E)$. Based on Bayes' theorem, we have:

$$\underbrace{P(X, Y \mid E)}_{\text{posterior}} = \frac{P(E \mid X, Y)}{P(E)} P(X, Y) \propto \underbrace{P(E \mid X, Y)}_{\text{likelihood}} \underbrace{P(X, Y)}_{\text{prior}}.$$

Therefore, we propose to use **maximum a posteriori** (MAP) estimator to obtain the best solution [11]:

$$\hat{\mathbf{x}}^*, \hat{\mathbf{y}}^* = \arg\max_{\hat{\mathbf{x}} \in \{0,1\}^N, \hat{\mathbf{y}} \in \{0,1\}^M} \underbrace{P(E = \mathbf{E} \mid X = \hat{\mathbf{x}}, Y = \hat{\mathbf{y}})}_{\text{likelihood}} \underbrace{P(X = \hat{\mathbf{x}}, Y = \hat{\mathbf{y}})}_{\text{prior}}. \tag{2}$$

That is to say, we exhaustively explore all 2^N possible configurations of $\hat{\mathbf{x}}$ and 2^M configurations of $\hat{\mathbf{y}}$, computing the likelihood and prior for each pair. We then find the $\hat{\mathbf{x}}^*$ and $\hat{\mathbf{y}}^*$ that yield the highest posterior and select the correct code solutions and test cases indicated by $\hat{\mathbf{x}}^*$ and $\hat{\mathbf{y}}^*$. This optimization problem is a 0/1 integer programming problem, in which all variables are restricted to 0 or 1. The following then answers *RQ1*.

> **Answer to *RQ1*:** Given a passing matrix $\mathbf{E}$, the optimal selection strategy can be framed as a 0/1 integer programming problem, by finding the one $\hat{\mathbf{x}} \in \{0,1\}^N$ and $\hat{\mathbf{y}} \in \{0,1\}^M$ that maximizes the posterior probability $P(X = \hat{\mathbf{x}}, Y = \hat{\mathbf{y}} \mid E = \mathbf{E})$.

Before calculating Eq.(2), we first introduce the following two assumptions which are necessary for our subsequent computation.

Assumption 2. *The code solutions X and the test cases Y are independent and randomly sampled.*

Assumption 3. *Each E_{ij} is only dependent by the X_i and Y_j, $\forall i \in [N], j \in [M]$.*

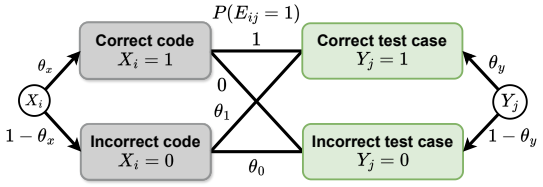


Figure 2: Illustration of the generation process. The correctness of code X_i and test case Y_j is sampled using parameters θ_x and θ_y respectively. E_{ij} is generated based on X_i and Y_j, using the corresponding parameters (1, 0, θ_1 or θ_0).

Remark 1. *Assumption 2 is also used by Chen et al. [5]. Assumption 3 assumes that a passing state E_{ij} is independent of any other variables except for the corresponding code X_i and test case Y_j, which means that $E_{ij}(i \in [N], j \in [M])$ are conditional independent when given X and Y. We will further discuss these assumptions in Section 6.*

Based on Assumption 3, we can explicitly formulate $P(E_{ij} \mid X_i, Y_j)$ as follows,

$$\begin{aligned} &P(E_{ij}=1 \mid X_i=1, Y_j=1)=1, \quad P(E_{ij}=1 \mid X_i=1, Y_j=0)=0, \\ &P(E_{ij}=1 \mid X_i=0, Y_j=1)=\theta_1, \quad P(E_{ij}=1 \mid X_i=0, Y_j=0)=\theta_0, \end{aligned} \tag{3}$$

where θ_1 and θ_0 are unknown parameters, indicating the probabilities of an incorrect solution passing a correct test case (θ_1) and passing an incorrect test case (θ_0). Eq.(3) suggests that if a solution is correct ($X_i = 1$), E_{ij} is *deterministic* by Y_j to fulfill the consistency (Assumption 1). When a solution is incorrect ($X_i = 0$), E_{ij} is a Bernoulli *random* variable, *i.e.*, a random variable that can only take 0 or 1, where the probability depends on Y_j.

Based on Assumption 2, the correctness of code solutions X and test cases Y are independent and therefore follow Bernoulli distributions as well. Suppose that:

$$P(X_i=1)=\theta_x, \quad P(Y_j=1)=\theta_y, \quad \forall i \in [N], j \in [M],$$

where θ_x and θ_y are two unknown parameters. To summarize, Fig. 2 illustrates the generation process of E based on four unknown parameters θ_1, θ_0, θ_x and θ_y for a clear presentation.

For ease of notation, we omit the random variables in the probability expressions in subsequent sections, *e.g.*, using $P(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ to replace $P(X = \hat{\mathbf{x}}, Y = \hat{\mathbf{y}})$. In the following sections, we provide a detailed explanation of how to derive the likelihood and prior in Eq.(2) based on the generation process proposed in Fig. 2.

Computing the likelihood. Based on Assumption 3 and Remark 1, we can expand the likelihood $P(\mathbf{E} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}})$ into the following form:

$$\begin{aligned} P(\mathbf{E} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) &= \prod_i \prod_j P(e_{ij} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ &= \underbrace{\prod_{\hat{x}_i=1} \prod_j P(e_{ij} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}})}_{P_1} \underbrace{\prod_{\hat{x}_i=0} \prod_j P(e_{ij} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}})}_{P_0}, \end{aligned} \tag{4}$$

where $i \in [N]$ and $j \in [M]$. The first equality is based on the independence of E_{ij}. The second equality splits e_{ij} into two parts, *i.e.*, P_1 and P_0, based on $\hat{x}_i$.

According to Eq.(3), P_1 is either 1 or 0. If $\hat{\mathbf{y}}$ and $\mathbf{E}$ are consistent with $\hat{\mathbf{x}}$ (*i.e.*, satisfy Assumption 1), then P_1 is 1; otherwise P_1 is 0. Here we only focus on consistent configurations that satisfy Assumption 1. Under this condition, $P(\mathbf{E} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) = P_0$, so we only need to compute P_0. Suppose:

$$\begin{aligned} \mathbf{E}_1 &= \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 1, i \in [N], j \in [M]\}, \\ \mathbf{E}_0 &= \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 0, i \in [N], j \in [M]\}. \end{aligned} \tag{5}$$

Based on Eq.(3), $\mathbf{E}_1$ (or $\mathbf{E}_0$) contains a set of independent Bernoulli variables related to θ_1 (or θ_0). Therefore:

$$\begin{aligned} P_0 &= \prod_{\hat{x}_i=0} \prod_{\hat{y}_j=1} P(e_{ij} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) \cdot \prod_{\hat{x}_i=0} \prod_{\hat{y}_j=0} P(e_{ij} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ &= P(\mathbf{E}_1 \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) \cdot P(\mathbf{E}_0 \mid \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ &= \int_0^1 P(\mathbf{E}_1 \mid \theta_1) P(\theta_1) \mathrm{d}\theta_1 \int_0^1 P(\mathbf{E}_0 \mid \theta_0) P(\theta_0) \mathrm{d}\theta_0 \\ &= \int_0^1 \theta_1^{n_1} (1-\theta_1)^{|\mathbf{E}_1|-n_1} P(\theta_1) \mathrm{d}\theta_1 \int_0^1 \theta_0^{n_0} (1-\theta_0)^{|\mathbf{E}_0|-n_0} P(\theta_0) \mathrm{d}\theta_0, \end{aligned} \tag{6}$$

where the third equality uses the fact that $\mathbf{E}_1$ only depends on θ_1 and $\mathbf{E}_0$ only depends on θ_0, which follows Bernoulli distributions based on Eq.(3). We leverage the law of total probability, where $P(\theta_1)$ and $P(\theta_0)$ are prior distributions for the two unknown parameters. The fourth equality leverages the formulation of the Bernoulli distribution, where $n_1 = \sum_{e_{ij} \in \mathbf{E}_1} e_{ij}$ and $n_0 = \sum_{e_{ij} \in \mathbf{E}_0} e_{ij}$ are the element sums of $\mathbf{E}_1$ and $\mathbf{E}_0$ respectively.

Computing the prior. To compute the prior $P(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, following the similar derivation as above, we have:

$$\begin{aligned} P(\hat{\mathbf{x}}, \hat{\mathbf{y}}) &= P(\hat{\mathbf{x}}) P(\hat{\mathbf{y}}) \\ &= \int_0^1 P(\hat{\mathbf{x}} \mid \theta_x) P(\theta_x) \mathrm{d}\theta_x \int_0^1 P(\hat{\mathbf{y}} \mid \theta_y) P(\theta_y) \mathrm{d}\theta_y \\ &= \int_0^1 \theta_x^{n_x} (1-\theta_x)^{N-n_x} P(\theta_x) \mathrm{d}\theta_x \int_0^1 \theta_y^{n_y} (1-\theta_y)^{M-n_y} P(\theta_y) \mathrm{d}\theta_y, \end{aligned} \tag{7}$$

where $P(\theta_x)$ and $P(\theta_y)$ are prior distributions. $n_x = \sum_{\hat{x}_i \in \hat{\mathbf{x}}} \hat{x}_i$ and $n_y = \sum_{\hat{y}_j \in \hat{\mathbf{y}}} \hat{y}_j$ are the element sums of $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, respectively.

> **Answer to *RQ2*:** Under Assumptions 2 and 3, the posterior of the optimal strategy can be expanded into four integrals (Eq.(6) and Eq.(7)) related to some *observed* events (n_1, n_0, n_x, and n_y) and prior distributions on four *unobserved* parameters (θ_1, θ_0, θ_x, and θ_y), which is not computable.

3.2 Practical Implementation

Recall that to compute the optimal strategy, we need to compute likelihood (Eq.(6)) and prior (Eq.(7)), which is not computable however due to complicated integrals and unknown prior distributions. In this section, we describe how to design an efficient approach to approximate the optimal strategy.

Computing integrals. In Bayesian statistics, employing *conjugate distributions* for prior distributions is a standard technique to simplify integrals in posterior computation [31]. In our case, all the variables X, Y, and E follow the Bernoulli distributions, whose conjugate prior is the Beta distribution [4]. Thus, we assume the four parameters follow **Beta distributions**, formally,

$$\begin{aligned} &P(\theta_0) \propto \theta_0^{\alpha_0-1}(1-\theta_0)^{\beta_0-1}, \quad P(\theta_1) \propto \theta_1^{\alpha_1-1}(1-\theta_1)^{\beta_1-1}, \\ &P(\theta_x) \propto \theta_x^{\alpha_x-1}(1-\theta_x)^{\beta_x-1}, \quad P(\theta_y) \propto \theta_y^{\alpha_y-1}(1-\theta_y)^{\beta_y-1}, \end{aligned} \tag{8}$$

where α and β are eight hyperparameters that reflect our existing belief or prior knowledge. We ignore all probability normalizing constants for ease of notation since they will not change the selection decision. These hyperparameters allow us to integrate some effective prior knowledge, which will be elaborated in Section 3.3.

To illustrate how Beta distributions simplify computation, we take θ_x as an example. Combining the integral about θ_x in Eq.(7) with $P(\theta_x)$ in Eq.(8), we obtain:

$$\begin{aligned}
&\int_0^1 \theta_x^{n_x}(1-\theta_x)^{N-n_x}P(\theta_x)\mathrm{d}\theta_x \\
\propto &\int_0^1 \theta_x^{n_x}(1-\theta_x)^{N-n_x}\theta_x^{\alpha_x-1}(1-\theta_x)^{\beta_x-1}\mathrm{d}\theta_x \\
= &\int_0^1 \theta_x^{n_x+\alpha_x-1}(1-\theta_x)^{N-n_x+\beta_x-1}\mathrm{d}\theta_x \\
= &\,\mathrm{B}\left(n_x+\alpha_x, N-n_x+\beta_x\right),
\end{aligned}$$

where $\mathrm{B}(\cdot)$ is known as the *Beta function* [10], which can be efficiently computed by modern scientific libraries like SciPy [41]. This deduction is applicable to θ_1, θ_0, and θ_y as well. Combining Eq.(2), Eq.(4), Eq.(6), and Eq.(7), and applying the similar transformation to integrals yields the formula for the computable posterior:

$$\begin{aligned}
&P(\mathbf{E}\mid\hat{\mathbf{x}},\hat{\mathbf{y}})P(\hat{\mathbf{x}},\hat{\mathbf{y}}) = P_1\cdot P_0\cdot P(\hat{\mathbf{x}},\hat{\mathbf{y}}) \\
\propto\, &P_1\cdot\left[\mathrm{B}\left(n_1+\alpha_1,|\mathbf{E}_1|-n_1+\beta_1\right)\mathrm{B}\left(n_0+\alpha_0,|\mathbf{E}_0|-n_0+\beta_0\right)\right] \\
&\cdot\left[\mathrm{B}\left(n_x+\alpha_x,N-n_x+\beta_x\right)\mathrm{B}\left(n_y+\alpha_y,M-n_y+\beta_y\right)\right] \quad (9)
\end{aligned}$$

This formula implies that the posterior probability can be approximated by multiplying four Beta functions, multiplied by a term P_1 indicating whether $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\mathbf{E}$ are consistent. We next present an error bound for this approximation (Proof can be found in the online Appendix [6]).

Theorem 1 (Approximation error bound). *Let Δ denote the absolute error between the true posterior (i.e., $P(\hat{\mathbf{x}},\hat{\mathbf{y}}\mid\mathbf{E})$) and the estimated posterior probability (i.e., multiplying the four Beta functions with the probability normalizing constants in Eq.(8)). Then:*

$$\Delta \le \frac{2}{P(\mathbf{E})}\left(c_1\Delta_{\theta_1}+c_0\Delta_{\theta_0}+c_x\Delta_{\theta_x}+c_y\Delta_{\theta_y}\right),$$

where Δ_{θ_1} is the total variance distance [38] between $P(\theta_1)$ and our assumed Beta prior distribution for θ_1. Δ_{θ_0}, Δ_{θ_x}, and Δ_{θ_y} are defined similarly. c_1, c_0, c_x, and c_y are some positive constants less than 1.

Theorem 1 shows that the difference of scores given by the approximated approach and the optimal strategy (*i.e.*, the true posterior probability) is bounded by the approximation errors in the prior distributions of the four parameters. If we can accurately give the prior distributions for each parameter θ, then $\Delta_{\theta_1}=\Delta_{\theta_0}=\Delta_{\theta_x}=\Delta_{\theta_y}=0$ and this approach can reduce to the optimal strategy. This highlights the importance of incorporating appropriate prior knowledge for different contexts.

Reducing computation complexity. Recall that the MAP strategy in Eq.(2) requires enumerating all 2^{N+M} combinations. Although the posterior probability is computable in Eq.(9), the enumeration cost still constrains the efficient identification of the optimal solution. Fortunately, given the role of the indicator P_1, only consistent combinations where $P_1=1$ need consideration. To be specific, for any $\hat{\mathbf{x}}\in\{0,1\}^N$ and $\hat{\mathbf{y}}\in\{0,1\}^M$ combination:

- $\hat{\mathbf{x}}$ must conform to the consistency assumption (Assumption 1). Thus, any correct solution i with $\hat{x}_i=1$ must pass the same test cases, *i.e.*, they should be within the same consensus set.
- $\hat{\mathbf{y}}$ must match the test cases passable by any correct solution, meaning all correct test cases j with $\hat{y}_j=1$ should also reside in the corresponding consensus set of the correct solutions.

Therefore, we claim that valid combinations must ensure that *all correct solutions and test cases should be in the same consensus set.* To reduce computations further, we consider any two solutions within the same consensus set. As these solutions pass identical test cases, they are completely symmetric and indistinguishable in $\mathbf{E}$. Therefore, it is illogical to differentiate between them. Thus, we assume that *solutions within the same consensus set should have identical predicted correctness.*

Based on these insights, we propose an enumeration method based on consensus sets. Similar to CodeT, we initially divide solutions and test cases into K consensus sets $(S_i^x,S_i^y)_{i=1}^K$. Within each set (S_i^x,S_i^y), we predict all solutions in S_i^x as 1 and all test cases in S_i^y as 1, while others are predicted as 0. This forms a consistent configuration $(\hat{\mathbf{x}},\hat{\mathbf{y}})$. We then calculate the posterior of $(\hat{\mathbf{x}},\hat{\mathbf{y}})$ with Eq.(10), where $P_1=1$ is always satisfied. This significantly reduces the number of explored configurations from 2^{N+M} to K.

3.3 Incorporating Prior Knowledge

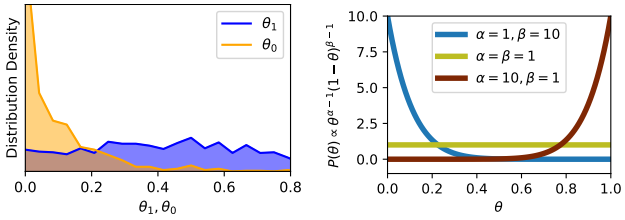
We have derived a general explicit expression for the posterior probability in Eq.(9), which includes eight hyperparameters corresponding to the Beta distribution for four θ. According to Theorem 1, we should incorporate proper prior knowledge to effectively approximate the optimal strategy. In this section, we investigate how to achieve this in the context of code generation.

Priors for θ_0 and θ_1. In practical scenarios, a test suite, not to mention a test case, is often incomplete. Therefore, a correct test case can fail to identify an incorrect solution, causing incorrect solutions to have a moderate probability of passing correct test cases (*i.e.*, θ_1). Conversely, to pass incorrect test cases that validate flawed functionalities, incorrect solutions must "accidentally" match this specific flaw to pass, making such occurrences (θ_0) relatively rare. This suggests that in practice, θ_0 may be very small, but θ_1 may not have a clear pattern.

To validate this conjecture, we analyzed code and test case generation tasks with five different models on HumanEval (See Section 5.2.1 for details of models) and computed the actual values of θ_1 and θ_0 for each problem in HumanEval using ground-truth solutions. Fig. 3(a) displays the true distributions of these parameters, showing that most θ_0 values are concentrated near zero, while θ_1 tends to follow a uniform distribution.

Based on this finding, we propose adopting a prior distribution approaching zero for θ_0 and a uniform prior distribution for θ_1. Therefore, we choose a beta prior distribution parameterized by $(\alpha_0=1,\beta_0\gg 1)$ for θ_0, and choose $(\alpha_1=\beta_1=1)$ for θ_1. As demonstrated in Fig. 3(b), such choice aligns with the findings in Fig. 3(a). In practice, β_0 serves as a tunable hyperparameter.

Priors for θ_x and θ_y. As discussed previously, each consistent $(\hat{\mathbf{x}},\hat{\mathbf{y}})$ corresponds to a consensus set. Chen *et al.* [5] identified a heuristic rule that the consensus set with the largest capacity (*i.e.*,



(a) Distributions for θ_0 and θ_1 (b) Beta distributions

Figure 3: (a) Real distributions for two parameters θ_0 and θ_1. (b) Three Beta distributions with different hyperparameters.

$n_x n_y$) is most likely correct. We will validate this rule theoretically in Section 4. Accordingly, we want the prior distribution $p(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ to favor configurations containing more ones and reward larger consensus sets. This can be implemented by setting the hyperparameters for θ_x as $(\alpha_x \gg 1, \beta_x = 1)$, and for θ_y as $(\alpha_y \gg 1, \beta_y = 1)$, as illustrated in Fig. 3(b). Moreover, we find it sufficient to combine α_x and α_y into a single hyperparameter α_{xy}, further reducing the parameter tuning space (see Section 5.2.4 for details).

Answer to *RQ3*: A practical strategy to approximate uncomputable optimal strategy is to score K consensus sets and select solutions within the highest-score set. The score is determined by multiplying 4 Beta functions, *i.e.*,

$$\begin{aligned} &\mathrm{B}\left(n_1+1, |\mathbf{E}_1| - n_1 + 1\right) \cdot \mathrm{B}\left(n_0+1, |\mathbf{E}_0| - n_0 + \beta_0\right) \\ &\cdot \mathrm{B}\left(n_x + \alpha_{xy}, N - n_x + 1\right) \cdot \mathrm{B}\left(n_y + \alpha_{xy}, M - n_y + 1\right), \end{aligned} \tag{10}$$

where β_0 and α_{xy} are tunable hyperparameters.

3.4 Further Analysis of Algorithm $\mathcal{B}^4$

Given that the score in Eq.(10) is multiplied by four Beta functions, we name this practical strategy $\mathcal{B}^4$. In this section, we provide a detailed analysis of the proposed $\mathcal{B}^4$ to deepen the understanding.

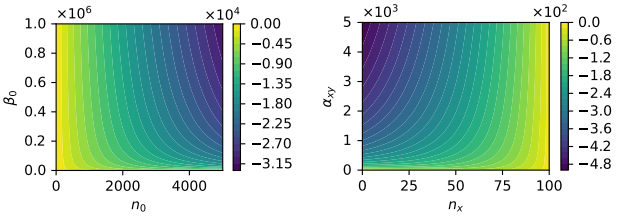
Full algorithm. Algorithm 1 outlines the workflow. Line 1 starts by collecting the set of test cases each code i passes (denoted as $\boldsymbol{e}_i$, *i.e.*, $\{e_{i1}, \cdots, e_{iM}\}$) and removes duplicates. In Line 3, we iterate over all unique test case sets. For each $\hat{\mathbf{y}}$ processed, we identify solutions whose passed test cases precisely match $\hat{\mathbf{y}}$ as $\hat{\mathbf{x}}$ in Line 4. Note that $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ define a consensus set together. Lines 5-9 compute the score of this consensus set (*i.e.*, the posterior) by Eq.(10). Ultimately, Lines 10-11 identify the consensus set with the highest score as the prediction. For numerical stability, we often store the logarithm of the scores in practice, by summing the logarithms of the four Beta functions.

A running example. We reuse Fig. 1 to illustrate how $\mathcal{B}^4$ works, using the hyperparameters $\beta_0 = \alpha_{xy} = 10$. Firstly, we deduplicate the rows in Eq.(1) and obtain $S^y = \{[1,1,1,0,0], [0,1,1,1,1], [0,0,1,1,0]\}$, indicating there are three distinct sets of passed test cases corresponding to three consensus sets. We need to iterate all three sets and score for each one. For the first iteration, $\hat{\mathbf{y}} = [1,1,1,0,0]$ and $\hat{\mathbf{x}} = [1,1,0,0]$. It indicates the first consensus set is $(\{x_1, x_2\}, \{y_1, y_2, y_3\})$. Using Eq.(5), we obtain:

Algorithm 1: Algorithm for $\mathcal{B}^4$

Input: Passing matrix $\mathbf{E} = \{e_{ij}\} \in \{0,1\}^{N \times M}$, hyperparameters $\beta_0 > 1, \alpha_{xy} > 1$
Output: $\hat{\mathbf{x}}^* \in \{0,1\}^N$ and $\hat{\mathbf{y}}^* \in \{0,1\}^M$ indicating the predicted correct solutions and test cases

1 $S^y \leftarrow \textsc{Deduplicate}(\{\boldsymbol{e}_i \mid i \in [N]\})$;
2 $Score^* \leftarrow -\infty$;
3 **for** $\hat{\mathbf{y}} \in S^y$ **do**
4 $\quad \hat{\mathbf{x}} \leftarrow \left\{\mathbb{1}_{\boldsymbol{e}_i = \hat{\mathbf{y}}} \mid i \in [N]\right\}$;
5 $\quad \mathbf{E}_1 \leftarrow \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 1, i \in [N], j \in [M]\}$;
6 $\quad \mathbf{E}_0 \leftarrow \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 0, i \in [N], j \in [M]\}$;
7 $\quad n_1 \leftarrow \sum_{e \in \mathbf{E}_1} e, \qquad n_0 \leftarrow \sum_{e \in \mathbf{E}_0} e$;
8 $\quad n_x \leftarrow \sum_{i \in [N]} \hat{x}_i, \quad n_y \leftarrow \sum_{j \in [M]} \hat{y}_j$;
9 $\quad Score \leftarrow$
$\quad\quad \mathrm{B}\left(n_1+1, |\mathbf{E}_1| - n_1 + 1\right) \cdot \mathrm{B}\left(n_0+1, |\mathbf{E}_0| - n_0 + \beta_0\right) \cdot$
$\quad\quad \mathrm{B}\left(n_x + \alpha_{xy}, N - n_x + 1\right) \cdot \mathrm{B}\left(n_y + \alpha_{xy}, M - n_y + 1\right)$;
10 $\quad$ **if** $Score > Score^*$ **then**
11 $\quad\quad (Score^*, \hat{\mathbf{x}}^*, \hat{\mathbf{y}}^*) \leftarrow (Score, \hat{\mathbf{x}}, \hat{\mathbf{y}})$;
12 **return** $\hat{\mathbf{x}}^*, \hat{\mathbf{y}}^*$;



(a) $\log \mathrm{B}\left(n_0+1, |\mathbf{E}_0| - n_0 + \beta_0\right)$ (b) $\log \mathrm{B}\left(n_x + \alpha_{xy}, N - n_x + 1\right)$

Figure 4: Visualization of two Beta functions used in our scoring strategy. We set $|\mathbf{E}_0| = 5000$ and $N = 100$.

$$\begin{aligned} \mathbf{E}_1 &= \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 1\} = \{e_{31}, e_{32}, e_{33}, e_{41}, e_{42}, e_{43}\}, \\ \mathbf{E}_0 &= \{e_{ij} \mid \hat{x}_i = 0, \hat{y}_j = 0\} = \{e_{34}, e_{35}, e_{44}, e_{45}\}, \end{aligned}$$

where $\mathbf{E}_1$ (or $\mathbf{E}_0$) represents the events that an incorrect solution passes a correct (or an incorrect) test case, under the prediction $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. We count these events: $n_1 = \sum \mathbf{E}_1 = 3$, $n_0 = \sum \mathbf{E}_0 = 3$, $n_x = \sum \hat{\mathbf{x}} = 2$, and $n_y = \sum \hat{\mathbf{y}} = 3$. Following this, the score is:

$$\begin{aligned} &\mathrm{B}\left(3+1, 6-3+1\right) \times \mathrm{B}\left(3+1, 4-3+10\right) \\ &\times \mathrm{B}\left(2+10, 4-2+1\right) \times \mathrm{B}\left(3+10, 5-3+1\right) = \mathbf{1.20 \times 10^{-12}}. \end{aligned}$$

For the second iteration, we have $\hat{\mathbf{y}} = [0,1,1,1,1]$ and $\hat{\mathbf{x}} = [0,0,1,0]$, resulting the score $\mathbf{1.15 \times 10^{-13}}$. For the third iteration, we have $\hat{\mathbf{y}} = [0,0,1,1,0]$ and $\hat{\mathbf{x}} = [0,0,0,1]$, resulting the score $\mathbf{1.24 \times 10^{-15}}$. One can find that the first consensus set has the largest score 1.20×10^{-12}, leading to the selection of $\{x_1, x_2\}$ as the optimal solution.

Understanding Beta functions. To further explore the role of two hyperparameters used in the $\mathcal{B}^4$ and our scoring strategy, we visualize two Beta functions related to two hyperparameters β_0 and α_{xy} in Fig. 4. Fig. 4(a) reveals that the function value is insensitive

to n_0 when β_0 is very small. As β_0 increases, the Beta function has little change for small n_0 but has a particularly small value for large n_0. This suggests that a larger β_0 leads the algorithm to reward predictions with smaller n_0. Recall that n_0 represents the number of incorrect solutions passing incorrect test cases, which is generally small in the real world (as discussed in Section 3.3). This indicates that our $\mathcal{B}^4$, which uses a $\beta_0 \gg 1$, aligns with practical conditions well. Similarly, Fig. 4(b) shows a large α_{xy} leads the algorithm to predict more correct solutions or tests (*i.e.*, larger n_x or n_y), which rewards a larger consensus set as we expected in Section 3.3.

4 THEORETICAL ANALYSIS

In this section, we address *RQ4* by a theoretical accuracy analysis of the two representative heuristics, MaxPass and CodeT, to investigate under what conditions they can and cannot work. MaxPass is a widely-used heuristic [18, 19, 22, 33] and CodeT is the state-of-the-art heuristic for code generation. Furthermore, these theoretical analyses further explain why the priors for $P(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ introduced in Section 3.3 are chosen. We assume that Assumptions 1-3 are satisfied, and the data follows the generation process in Fig. 2. All proofs can be found in the online Appendix [6].

We begin with a theorem which assesses MaxPass's accuracy when there is a large number of test cases:

Lemma 4.1. *Suppose there exist n_y correct test cases and $\overline{n}_y$ incorrect test cases ($n_y + \overline{n}_y = M$). When both n_y and $\overline{n}_y$ are large enough, the probability of any incorrect code passing Y ($Y \geq n_y$) test cases is:*

$$P(Y \geq n_y) \sim \Phi\left(\frac{\overline{n}_y\theta_0 - n_y(1-\theta_1)}{\sqrt{n_y\theta_1(1-\theta_1) + \overline{n}_y\theta_0(1-\theta_0)}}\right),$$

where Φ is the cumulative distribution function (CDF) of the standard normal distribution. θ_0 and θ_1 are defined in Eq.(3).

Theorem 2 (Impact of correct test cases for MaxPass). *If $\theta_1 < 1$, the accuracy of MaxPass (i.e., the probability of all incorrect solutions passing less than n_y test cases) can exponentially converge to 1 as $n_y \to \infty$.*

Theorem 3 (Impact of incorrect solutions for MaxPass). *If there are $\overline{n}_x$ incorrect solutions, the accuracy of MaxPass can exponentially converge to 0 as $\overline{n}_x \to \infty$.*

Theorem 2 demonstrates the working condition for MaxPass: it requires a large amount of correct test cases n_y to make the accuracy converge to 1. However, Theorem 3 also underscores a limitation of MaxPass: it lacks *scalability* to the number of code solutions N. As N increases, $\overline{n}_x$ increases and the accuracy of MaxPass will exponentially converge to zero.

Following this, we analyze the error of CodeT. Considering the problem's complexity, we *fix* the M test cases and explore how the error evolves as the number of generated code solutions N grows, as shown in the following theorem.

Lemma 4.2. *Suppose the correctness of code solutions and test cases are $\mathbf{x}$ and $\mathbf{y}$. Let $n_x = \sum \mathbf{x}$ and $n_y = \sum \mathbf{y}$ denote the number of correct code solutions and test cases, respectively. For any* ***incorrect*** *consensus set that corresponds to a prediction $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, similarly let $n_{\hat{x}} = \sum \hat{\mathbf{x}}$ and $n_{\hat{y}} = \sum \hat{\mathbf{y}}$. For arbitrary $\mathbf{y}$ and $\hat{\mathbf{y}}$, if N is sufficiently large, the probability of this consensus set being scored higher than the correct one by CodeT (i.e., $n_{\hat{x}}n_{\hat{y}} > n_x n_y$) follows:*

$$P(n_{\hat{x}}n_{\hat{y}} > n_x n_y) \sim \Phi\left(\frac{\sqrt{N}(\theta' n_{\hat{y}} - \theta_x n_y)}{\sqrt{n_{\hat{y}}^2\theta'(1-\theta') + n_y^2\theta_x(1-\theta_x) - 2n_{\hat{y}}n_y\theta'\theta_x}}\right),$$

where θ' is a constant, defined as:

$$\theta' = (1-\theta_x)\theta_1^{\hat{\mathbf{y}}^\top \mathbf{y}}(1-\theta_1)^{(1-\hat{\mathbf{y}})^\top \mathbf{y}}\theta_0^{\hat{\mathbf{y}}^\top(1-\mathbf{y})}(1-\theta_0)^{(1-\hat{\mathbf{y}})^\top(1-\mathbf{y})}.$$

Theorem 4 (Impact of θ_x and N for CodeT). *If θ_x is large enough such that $\theta' n_{\hat{y}} < \theta_x n_y$, then the error probability $P(n_{\hat{x}}n_{\hat{y}} > n_x n_y)$ can exponentially converge to 0 as $N \to \infty$. Otherwise, if θ_x is low enough such that $\theta' n_{\hat{y}} > \theta_x n_y$, the error probability converges to 1 as $N \to \infty$.*

Theorem 4 elucidates the working condition for CodeT: it requires a sufficient high correct probability of code solutions (high θ_x). If the generated solutions contain excessive incorrect solutions, CodeT may not work well. An important insight is that under the condition of high θ_x, CodeT offers better scalability compared to MaxPass: as the number of solutions N increases, CodeT's selection accuracy can exponentially converge towards 1 (Theorem 4), whereas MaxPass's accuracy will converge towards 0 (Theorem 3).

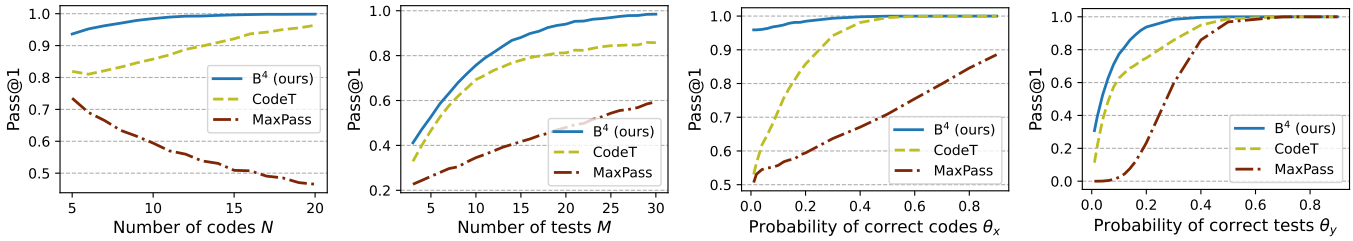
> **Answer to *RQ4*:** Existing heuristics work under specific conditions. MaxPass requires sufficient correct test cases, while CodeT requires a high correct probability of solutions. When both of their requirements are satisfied, CodeT has better scalability with the number of solutions N than MaxPass.

Considering the analyzing complexity, whether a similar error probability analysis can be directly provided for $\mathcal{B}^4$ is an open question.[1] Fortunately, these theoretical analyses still indirectly support the effectiveness of $\mathcal{B}^4$. For example, Theorem 4 validates the effectiveness of the priors for θ_x and θ_y of our $\mathcal{B}^4$. Recall that our introduced priors for $P(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ are similar to CodeT's assumptions (Section 3.3), which offers similar scalability benefits under the condition that θ_x is relatively large. However, it is crucial to note that these priors are just part of our methods. Besides the priors for θ_x and θ_y, we also incorporate priors for θ_0 and θ_1, which effectively compensates for the limitations of CodeT's priors, particularly in scenarios where θ_x is low. As our subsequent experiments confirm, $\mathcal{B}^4$ significantly outperforms CodeT in such challenging scenarios.

5 EXPERIMENT

In this section, we conduct experiments to further answer *RQ4* and *RQ5*. We start with exploring the conditions under which existing heuristics can work efficiently through simulation experiments in different controlled environments, to validate the theoretical insights discussed in Section 4. Subsequently, we compare the performance of $\mathcal{B}^4$ with existing heuristics on real-world datasets.

[1] To show the complexity, note that computing the distribution for $\mathcal{B}^4$'s score is necessary for estimating error probability. The score can be represented as the product of n_x, n_1, and n_0 after nonlinear transformations (here we assume n_y is given, as Lemma 4.2). However, despite oversimplification, *i.e.*, treating three variables as normal, linearizing the transformations, and assuming their independence, the computation is still a challenge in the literature [37].



(a) Varying N (b) Varying M (c) Varying θ_x (d) Varying θ_y

Figure 5: Pass@1 results of the three methods under different conditions in the simulated experiments. By default, we set $N = 10$, $M = 30$, $\theta_x = 0.2$, and $\theta_y = 0.3$ except for the varied one.

5.1 Simulated Experiments

In our simulated experiments, we sampled $N = 10$ solutions and $M = 30$ test cases, and set four parameters $\theta_x = 0.2$, $\theta_y = 0.3$, $\theta_1 = 0.4$ and $\theta_0 = 0.1$ by default. These default values are based on our measurement of the real data generated by CodeGen [29] on HumanEval [7]. Based on these parameters, we randomly sampled a data point $(\mathbf{x}, \mathbf{y}, \mathbf{E})$ following the process shown in Fig. 2. Subsequently, we used MaxPass, CodeT, and $\mathcal{B}^4$ to select the solutions $\hat{\mathbf{x}}$ using $\mathbf{E}$, and computed the proportion of correct solutions within $\hat{\mathbf{x}}$ (*i.e.*, Pass@1) using the ground-truth $\mathbf{x}$. We repeated this process 20,000 times and averaged the results to ensure stability for each experiment. Following Section 3.3, the hyperparameters β_0 and α_{xy} should be larger than 1, and we preliminarily chose $\beta_0 = \alpha_{xy} = 10$.

Figs. 5(a) and 5(b) display the results as the scale of data N and M change. One can observe in Fig. 5(a) that CodeT's performance gradually improves with an increase in the number of code solutions N, whereas MaxPass shows a decline as N increases. This confirms our theoretical results in Section 4: CodeT has better scalability with N than MaxPass. Fig. 5(b) shows that unlike with N, MaxPass tends to improve as M increases. Regardless of the values of N and M, $\mathcal{B}^4$ consistently outperforms the two baselines, proving that existing heuristic algorithms are not optimal. Specifically, $\mathcal{B}^4$ tends to provide greater performance enhancements relative to CodeT when N is small. This could be because CodeT does not perform as well when N is low, which is also validated in Theorem 4.

Figs. 5(c) and 5(d) display the results as the probability of correct solutions θ_x and test cases θ_y change. All three methods gradually improve as the accuracy increases. Specifically, both $\mathcal{B}^4$ and CodeT's accuracies can converge to 1 as θ_x increases, while all three methods converge to 1 as θ_y increases. This indicates that MaxPass is less sensitive to θ_x but more responsive to θ_y, confirming the findings of Lemma 4.1 that the number of correct test cases matters for MaxPass. $\mathcal{B}^4$ consistently outperforms all the two heuristics under all conditions. Notably, when θ_x is low, it significantly outperforms CodeT with a large improvement. This suggests that CodeT struggles under the condition of few correct solutions and affirms the findings of Theorem 4.

5.2 Real-world Experiments

5.2.1 Experiment setup. We conducted experiments on three public code generation benchmarks, HumanEval [7], MBPP [2] (sanitized version), and APPS [16] with three difficulty levels. These benchmarks have been widely used by LLM-based code generation studies [7, 13, 21, 29, 34]. Specifically, each benchmark contains some coding tasks, and each task consists of a natural language requirement, a function signature, and a golden test suite for evaluating the correctness of generated solutions. Notably, these golden test suites and the generated test cases are not the same; the generated test cases are used by each selection strategy to select the generated code, while the golden test suites are solely used to evaluate the performance of selection strategies.

We used the same zero-shot prompt format as CodeT [5] for both code and test case generation. Following CodeT, the numbers of generated solutions and test cases are 100 for HumanEval and MBPP and 50 for APPS. Both solutions and tests are generated by the same model.

For models, our experiments are based on Codex [7] (code-davinci-002 version), CodeGen [29] (16B Python mono-lingual version), and three recent open-source models, StarCoder [21], CodeLlama [34] (7B Python version) and Deepseek-Coder [13] (6.7B Instruct version). The generation hyperparameters such as temperature, top p, and max generation length are the same as [5]. Additionally, as APPS has significantly more problems (5,000) compared to HumanEval (164) and MBPP (427), testing all models on it is prohibitively expensive. Given that Codex outperforms the other models on HumanEval and MBPP in most of our experiments (using CodeT strategy), we followed Chen *et al.* [5] by only evaluating Codex's outputs on the APPS dataset.

For baselines, in addition to MaxPass [18, 19] and CodeT [5], we also used MBR-exec [22, 36], which is similar to CodeT but scores each consensus set with the number of solutions, and a naive Random, which picks a code from the generated solutions randomly. We reported the average Pass@1 of the selected solutions. Our method is presented in the format of $\mathcal{B}^4(\log_{10}\beta_0, \log_{10}\alpha_{xy})$. For example, $\mathcal{B}^4(4,3)$ represents $\beta_0 = 10^4$ and $\alpha_{xy} = 10^3$. For a fair comparison, all the methods operate on the same passing matrices $\mathbf{E}$. We reported three variants of methods: $\mathcal{B}^4(4,3)$, $\mathcal{B}^4(5,3)$, and $\mathcal{B}^4(6,3)$, and compared each of them with CodeT using Wilcoxon signed-rank significance test [43].

To comprehensively evaluate different selection methods, we filtered the problems based on the proportion of correct solutions among all generated solutions (*i.e.*, θ_x). We first filtered out problems with $\theta_x = 1$ and $\theta_x = 0$, as the solutions for these problems are either entirely correct or incorrect, which can not differentiate selection strategies. We name this setting **discriminative problems**. To provide a more challenging environment for selecting correct

Table 1: Pass@1 (%) of the code solutions selected by different strategies across various datasets and models with two settings (RD=Random, MP=MaxPass, MBR=MBR-exec, CT=CodeT). We also reported the average relative improvement of the three $\mathcal{B}^4$ variants over the strongest heuristic CodeT and the p-values derived from the Wilcoxon signed-rank test.

Dataset	Model	Discriminative Problems ($0 < \theta_x < 1$)							Hard Problems ($0 < \theta_x < 0.5$)						
		RD	MP	MBR	CT	ours			RD	MP	MBR	CT	ours		
						$\mathcal{B}^4$(4,3)	$\mathcal{B}^4$(5,3)	$\mathcal{B}^4$(6,3)					$\mathcal{B}^4$(4,3)	$\mathcal{B}^4$(5,3)	$\mathcal{B}^4$(6,3)
HumanEval	CodeGen	32.5	28.6	44.8	51.5	56.8	**58.0**	56.9	13.0	11.1	21.0	31.4	38.2	40.0	**40.8**
	Codex	39.2	57.8	55.0	71.7	70.6	**73.1**	**73.1**	19.2	43.2	27.6	54.6	52.9	**56.9**	**56.9**
	StarCoder	29.8	32.2	47.9	55.0	59.0	**59.3**	57.8	15.0	16.3	29.3	38.9	44.4	**44.8**	42.8
	CodeLlama	34.1	40.6	52.6	61.7	63.5	**64.8**	64.0	15.8	24.5	30.8	44.1	46.7	**48.6**	47.4
	Deepseek-Coder	65.3	58.2	80.4	79.2	**80.5**	78.5	78.5	24.7	33.7	35.0	30.6	**35.5**	31.3	31.3
MBPP	CodeGen	42.4	48.1	56.4	64.9	**66.7**	64.9	64.7	21.8	30.8	28.4	43.5	**45.6**	42.5	42.3
	Codex	55.1	70.5	71.9	80.0	80.8	81.3	**81.9**	23.9	46.4	32.5	53.9	55.1	56.6	**58.0**
	StarCoder	46.1	55.6	65.6	69.6	**70.6**	**70.6**	**70.6**	21.5	39.5	37.8	45.6	47.5	**47.9**	**47.9**
	CodeLlama	47.2	60.0	65.4	72.4	72.6	73.4	**73.8**	19.8	39.0	30.7	44.7	45.0	46.7	**47.5**
	Deepseek-Coder	56.5	71.4	66.9	75.2	**75.9**	**75.9**	**75.9**	22.3	45.6	25.7	45.9	46.7	**47.6**	**47.6**
APPS introductory	Codex	36.2	46.4	41.6	59.5	63.7	63.7	**64.4**	17.6	29.5	15.9	41.6	46.6	47.6	**48.3**
APPS interview		15.6	26.0	14.7	36.0	40.4	40.8	**41.1**	11.2	22.4	8.0	30.6	35.1	35.5	**35.9**
APPS competition		7.9	16.8	3.1	17.3	23.1	**25.2**	**25.2**	7.0	16.2	2.5	16.0	21.9	**24.0**	**24.0**
Avg. relative improvement over the strongest heuristic CodeT						+6.1%	**+7.5%**	+7.2%					+10.1%	**+12.0%**	**+12.0%**
p-value						0.001	0.0003	0.0006					0.0009	0.0004	0.0004

solutions, we propose a new setting on a subset of discriminative problems where $0 < \theta_x < 0.5$, named **hard problems**.

5.2.2 Main results. Table 1 presents the main results, showing that all three $\mathcal{B}^4$ variants consistently and significantly outperform existing heuristics. Specifically, each single variant of B4 outperforms all baselines in most cases. On average, each variant surpasses the strongest heuristic baseline, CodeT, by 6-12% with statistically significant differences (proven by significance tests). This highlights a substantial gap between existing heuristics and the optimal strategy and suggests our method effectively approximates the optimal.

Additionally, $\mathcal{B}^4$ shows a greater performance improvement over CodeT in more challenging scenarios (*i.e.*, smaller θ_x). It achieves a 6.1%-7.5% relative improvement in discriminative problems and a 10.1%-12.0% improvement in hard problems. In the most challenging scenario (APPS competition on hard problems), $\mathcal{B}^4$ can even deliver up to a 50% enhancement over CodeT and 246% over random selection. These findings align with the conclusions of Lemma 4.2 and the simulated experiments depicted in Fig. 5(c), confirming that existing heuristics struggle with more difficult tasks. We also observed that the gains from $\mathcal{B}^4$ on the MBPP dataset are less significant than on HumanEval and APPS, likely because the MBPP problems are inherently simpler, as indicated by Random.

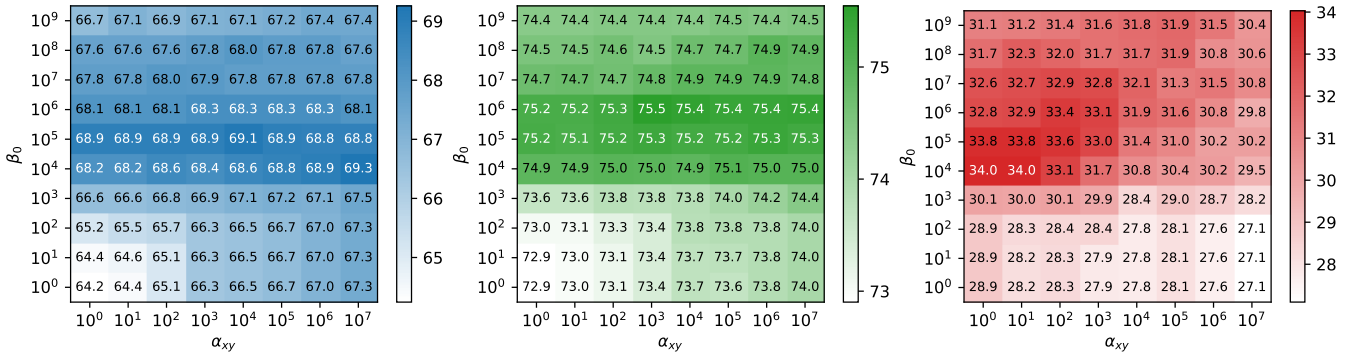
For hyperparameters, the optimal hyperparameter β_0 for $\mathcal{B}^4$ varies across different scenarios, suggesting that the prior distribution of θ_0 may differ depending on the context. This makes sense as different models might generate incorrect solutions and test cases with different patterns. For example, when models more easily misinterpret the problem, leading solutions and test cases to follow the same incorrect patterns, the probability of incorrect solutions passing incorrect test cases θ_0 can increase, thus necessitating a larger β_0 to reflect this change. We will further discuss the impact of hyperparameters in the next section.

Answer to *RQ5*: The proposed $\mathcal{B}^4$ significantly outperforms existing heuristics, achieving a 6.1%-7.5% relative improvement in discriminative problems and a 10.1%-12.0% improvement in hard problems over the strongest CodeT.

5.2.3 Ablation studies on two hyperparameters. Figs. 6(a) and 6(b) show the average performance on two datasets as influenced by two hyperparameters β_0 and α_{xy}. Recall that β_0 controls the likelihood $P(\mathbf{E} \mid \hat{\mathbf{x}}, \hat{\mathbf{y}})$ and α_{xy} controls the prior $P(\hat{\mathbf{x}}, \hat{\mathbf{y}})$. For β_0, performance on both datasets initially increases and decreases as β_0 increases, with the optimal value around 10^4-10^6. This pattern suggests that an appropriate β_0 can better align with the prior distribution of θ_0, resulting in more accurate likelihood estimates.

For α_{xy}, we found that performance improves with an increase in α_{xy} on HumanEval and MBPP, whereas the opposite is true for APPS. Recall that a larger α_{xy} makes the strategy closer to CodeT. One possible reason is that the tasks in HumanEval and MBPP are relatively simpler, so CodeT performs better on these two datasets, as shown in Theorem 4.

5.2.4 Ablation studies on splitting α_{xy} into two individual hyperparameters α_x and α_y. As discussed in Section 3.3, we combined α_x and α_y into a single α_{xy} in Eq.(8). This section examines the effects of tuning α_x and α_y independently. Section 5.2.3 shows the trend of average performance across all datasets as α_x and α_y vary, with β_0 set at 10^6. We observe that performance declines significantly when $\alpha_y - \alpha_x$ has a large value (*i.e.*, in the bottom right area of Section 5.2.3). As $\alpha_y - \alpha_x$ gradually decreases (moving from the bottom right towards the top left), performance can be gradually improved. The method achieves optimal performance when α_x and α_y are closed ($\alpha_x = 10^3$ and $\alpha_y = 10^2$). Considering that the model's performance is not sensitive to α_{xy} when β_0 is within an appropriate range, we argue that merging α_x and α_y into one hyperparameter simplifies tuning without substantially affecting performance. Therefore, we adopted $\alpha_x = \alpha_y = 10^3$ in our previous main experiment.



(a) HumanEval (b) MBPP (c) APPS

Figure 6: Pass@1 (%) results of varying α_{xy} and β_0 on HumanEval's, MBPP's, and APPS' discriminative problems.

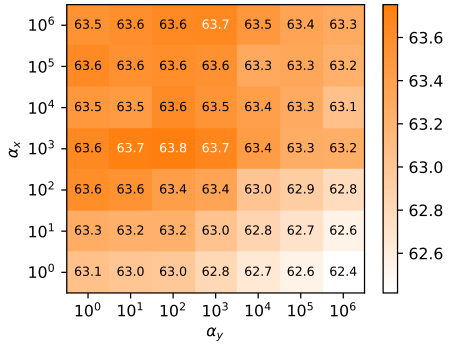


Figure 7: Pass@1 (%) results of splitting α_{xy} into two hyperparameters α_x and α_y on the discriminative problems of HumanEval and MBPP when $\beta_0 = 10^6$.

5.2.5 Computational Cost. Table 2 shows the running time of the $\mathcal{B}^4$ algorithm and CodeT, where $\mathcal{B}^4$ is slightly slower than CodeT due to the relatively higher overhead of beta functions in $\mathcal{B}^4$ compared to simple counting in CodeT. Notably, the computational complexity of both is the same, as both first partition the consensus sets and then score them. We can observe that even for large M and N (*e.g.*, $M = N = 400$), the running time is less than one second, which is much less than the time to generate 400 solutions and tests with LLMs. Therefore, we believe that the efficiency of $\mathcal{B}^4$ will not become a bottleneck for practical systems.

6 DISCUSSION

In this section, we discuss the limitations and threats to the validity of this study.

6.1 Limitations

Assumption 2 and 3. These assumptions are related to independence. Assumption 2 considers the correctness of code solutions and test cases are independent, which can be violated if there is a causal relationship in their generation, such as using a generated test case as input to an LLM for further generation. Assumption 3 states that passing probability is solely determined by the correctness of the associated code and test case. However, the independence of passing states may be broken by other unobserved factors hidden in the code. For example, if two incorrect solutions exhibit similar structures and similar error types, their passing states might be positively correlated. Considering the significant complexity introduced by the lack of independence, further exploration of the dependence case is deferred to future research.

Table 2: Computation cost with the increases of the number of code solutions N and test cases M.

N and M	100	200	300	400
CodeT	10 ms	65 ms	202 ms	455 ms
$\mathcal{B}^4$	15 ms	79 ms	243 ms	588 ms

Prior for θ_0. This prior assumes that θ_0 (*i.e.*, the probability of incorrect solutions passing incorrect test cases) is typically low. However, when LLMs misinterpret a problem, incorrect test cases may coincidently specify the functionality of incorrect solutions and potentially increase θ_0. Considering that this prior can bring considerable benefits (as shown in Section 5.2.3), we argue that its advantages significantly outweigh the limitations.

Priors for θ_x and θ_y. These priors, similar to the heuristic rule of CodeT, suggest that larger consensus sets are more likely to be correct. We have validated its theoretical effectiveness under the conditions of large N and high θ_x, as detailed in Section 4. Even though its efficacy may diminish when these conditions are not met, the prior for θ_0 effectively compensates for this situation as demonstrated in Section 5.2.3.

Hyperparameters. Our method includes two hyperparameters, α_{xy} and β_0, which may pose challenges in tuning across different usage scenarios. Fortunately, we have found that using consistent hyperparameters across all benchmarks can still yield significant improvements in our experimental scenarios. The tuning of hyperparameters for specific applications, potentially using a validation set to optimize them, remains an area for future research.

Theoretical results. To derive a closed form of the probabilities, we used *the Law of Large Numbers* to examine the scenarios where N and M are sufficiently large. Besides, in Lemma 4.2, we focus on a

single incorrect consensus set and neglect the complex interactions of multiple incorrect sets for computational convenience. Despite these simplifications, the key insights from these theorems are empirically validated in Section 5, thus we believe these theoretical analyses remain valuable. Finally, whether an error probability of $\mathcal{B}^4$ can be explicitly provided, similar to those of existing heuristics provided in Section 4, is an interesting open question.

6.2 Threats to Validity

The used benchmarks, *i.e.*, HumanEval, MBPP, and APPS, consist of small-scale function-level tasks and may not capture the nuances of more complex scenarios in practice. Additionally, some ground-truth test suites used to evaluate the solution's correctness in the benchmarks are just an approximation to the specification and can be incomplete. This leads to a few correct solutions (*i.e.*, the solutions passing the ground truth test suite) not exhibiting identical functionality and violating Assumption 1. Considering that such cases are relatively rare and most related work is centered on these benchmarks [5, 7, 13, 21, 34], we believe this threat will not significantly influence our conclusions.

Our experiments focus on Python code generation tasks, which may not reflect the effectiveness of our method on other programming languages and other software engineering (SE) generation tasks. However, Python is one of the most popular programming languages and code generation is a challenging and important SE generation task. In addition, our method is language-agnostic and our theoretical framework can be easily adapted to other SE generation tasks, such as Automated Program Repair (APR) and code translation. Therefore, we believe this threat is limited.

7 RELATED WORK

Reranking and selection for plausible solutions. Using external validators (*e.g.*, test cases) to assess, rerank, or select the generated solutions is widely used in various software engineering tasks. In code generation, Lahiri *et al.* [18] incorporated user feedback to choose test cases for code selection. In APR, Yang *et al.* [46] used test cases generated by fuzz testing to validate automatically generated patches. In code translation, Roziere *et al.* [33] leveraged EvoSuite [12] to automatically generate test cases for filtering out invalid translations. These methods are developed by assuming that the validators are reliable and can be reduced to the MaxPass strategy in our work. However, it may be ineffective when the validators are plausible, as evidenced in Section 4. In code generation, several cluster-based strategies are proposed to leverage incomplete or plausible test cases to rerank LLM-generated code solutions [5, 22, 36]. Li *et al.* [22], Shi *et al.* [36] and Chen *et al.* [5] clustered code solutions based on their test results and scored each with the cluster capacity. These cluster-based heuristics, particularly CodeT [5], can work well when the test cases are plausible but are susceptible to the incorrectness of solutions as in Section 4.

Some research uses deep learning techniques for ranking LLM-generated code snippets without executable test cases. Inala *et al.* [17] introduced a neural ranker for predicting the validity of a sampled program. Chen *et al.* [7] and Zhang *et al.* [49] leveraged the LLM likelihood of the generated program for selecting the most probable code snippets. These strategies fall beyond the scope of this work since the problem we tackle does not assume the existence of additional training data or the ranking scores produced by the generation techniques. However, it is an interesting question whether these strategies have a theoretical guarantee.

Code generation. Code generation is an important task in software engineering, aimed at automating the production of code from defined software requirements [23]. Traditional techniques rely on predefined rules, templates, or configuration data to automate the process [14, 42], and often struggle with flexibility across different projects. Due to the impressive success of large language models (LLMs), recent studies focus on training LLMs on extensive code corpora to tackle complex code generation challenges [48]. Many code LLMs have shown remarkable capabilities in this domain, such as Codex [7], CodeGen [29], StarCoder [21], CodeLlama [34] and DeepSeek-Coder [13]. This paper focuses on assessing the code solutions generated by a code generation approach with plausible test cases, and is thus orthogonal to these techniques.

Test case generation. Developing and maintaining human-crafted test cases can be expensive. Many techniques have been proposed to automatically generate test cases. Traditional approaches include search-based [15, 20, 24], constrained-based [44], and probability-based techniques [30]. Although most of these approaches achieve satisfactory correctness, they are constrained by inadequate coverage and poor readability, and are typically limited to generating only regression oracles [45] or implicit oracles [3]. Recently, applying deep learning models (*e.g.*, LLMs) to generate test cases has become popular [1, 8, 9, 25–28, 32, 35, 39, 40, 47]. However, ensuring the correctness and reliability of these generated test cases remains difficult. This paper explores the challenging problem of employing such plausible test cases for selecting plausible code solutions.

8 CONCLUSION AND FUTURE WORK

In this study, we introduce a systematic framework to derive an optimal strategy for assessing and selecting plausible code solutions using plausible test cases. We then develop a novel approach that approximates this optimal strategy with an error bound and tailors it for code generation tasks. By theoretical analysis, we show that existing heuristics are suboptimal. Our strategy substantially outperforms existing heuristics in several real-world benchmarks.

Future work could explore adapting our framework to other generation tasks in software engineering, such as automatic program repair and code translation. Also, the effectiveness of our proposed priors in these contexts, as well as the potential for alternative priors, remains an open question.

Our online appendix is available on Zenodo [6].

ACKNOWLEDGMENTS

This research is supported by the National Natural Science Foundation of China (No. 62202420) and the Software Engineering Application Technology Lab at Huawei under the Contract TC20231108060. Zhongxin Liu gratefully acknowledges the support of Zhejiang University Education Foundation Qizhen Scholar Foundation. We would also like to thank Yihua Sun for inspiring the incorporation of prior knowledge and for proofreading the manuscript, as well as Zinan Zhao and Junlin Chen for their discussions on the theory.

REFERENCES

[1] Saranya Alagarsamy, Chakkrit Tantithamthavorn, and Aldeida Aleti. 2023. A3Test: Assertion-Augmented Automated Test Case Generation. arXiv:2302.10352 [cs.SE]

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.PL]

[3] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2014. The oracle problem in software testing: A survey. *IEEE transactions on software engineering* 41, 5 (2014), 507–525.

[4] Thomas Bayes. 1763. LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S. *Philosophical transactions of the Royal Society of London* 53 (1763), 370–418.

[5] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2023. CodeT: Code Generation with Generated Tests. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=ktrw68Cmu9c

[6] Mouxiang Chen, Zhongxin Liu, He Tao, Yusu Hong, David Lo, Xin Xia, and Jianling Sun. 2024. *B4: Towards Optimal Assessment of Plausible Code Solutions with Plausible Tests*. https://doi.org/10.5281/zenodo.13737381

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[8] Yinghao Chen, Zehao Hu, Chen Zhi, Junxiao Han, Shuiguang Deng, and Jianwei Yin. 2024. ChatUniTest: A Framework for LLM-Based Test Generation. arXiv:2305.04764 [cs.SE]

[9] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel C. Desmarais. 2023. Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing. arXiv:2308.16557 [cs.SE]

[10] Philip J Davis. 1972. Gamma function and related functions. *Handbook of mathematical functions* 256 (1972).

[11] Morris H DeGroot. 2005. *Optimal statistical decisions*. John Wiley & Sons.

[12] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.

[13] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).

[14] Nicolas Halbwachs, Pascal Raymond, and Christophe Ratel. 1991. Generating efficient code from data-flow programs. In *Programming Language Implementation and Logic Programming: 3rd International Symposium, PLILP'91 Passau, Germany, August 26–28, 1991 Proceedings 3*. Springer, 207–218.

[15] Mark Harman and Phil McMinn. 2010. A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search. *IEEE Transactions on Software Engineering* 36, 2 (2010), 226–247. https://doi.org/10.1109/TSE.2009.71

[16] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence With APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=sD93GOzH3i5

[17] Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-Aware Neural Code Rankers. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 13419–13432. https://proceedings.neurips.cc/paper_files/paper/2022/file/5762c579d09811b7639be2389b3d07be-Paper-Conference.pdf

[18] Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. 2023. Interactive Code Generation via Test-Driven User-Intent Formalization. arXiv:2208.05950 [cs.SE]

[19] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 21314–21328.

[20] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. 2023. CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 919–931. https://doi.org/10.1109/ICSE48619.2023.00085

[21] Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=KoFOg41haE Reproducibility Certification.

[22] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158 arXiv:https://www.science.org/doi/pdf/10.1126/science.abq1158

[23] Hui Liu, Mingzhu Shen, Jiaqi Zhu, Nan Niu, Ge Li, and Lu Zhang. 2022. Deep Learning Based Program Generation From Requirements Text: Are We There Yet? *IEEE Transactions on Software Engineering* 48, 4 (2022), 1268–1289. https://doi.org/10.1109/TSE.2020.3018481

[24] Stephan Lukasczyk and Gordon Fraser. 2022. Pynguin: Automated unit test generation for python. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. 168–172.

[25] Antonio Mastropaolo, Nathan Cooper, David Nader Palacio, Simone Scalabrino, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2023. Using Transfer Learning for Code-Related Tasks. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1580–1598. https://doi.org/10.1109/TSE.2022.3183297

[26] Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2021. Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 336–347. https://doi.org/10.1109/ICSE43902.2021.00041

[27] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 2450–2462. https://doi.org/10.1109/ICSE48619.2023.00205

[28] Pengyu Nie, Rahul Banerjee, Junyi Jessy Li, Raymond J Mooney, and Milos Gligoric. 2023. Learning deep semantics for test completion. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2111–2123.

[29] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=iaYcJKpY2B_

[30] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. 2007. Feedback-Directed Random Test Generation. In *29th International Conference on Software Engineering (ICSE'07)*. 75–84. https://doi.org/10.1109/ICSE.2007.37

[31] Howard Raiffa and Robert Schlaifer. 2000. *Applied statistical decision theory*. Vol. 78. John Wiley & Sons.

[32] Nikitha Rao, Kush Jain, Uri Alon, Claire Le Goues, and Vincent J. Hellendoorn. 2023. CAT-LM Training Language Models on Aligned Code And Tests. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 409–420. https://doi.org/10.1109/ASE56229.2023.00193

[33] Baptiste Roziere, Jie Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. 2022. Leveraging Automated Unit Tests for Unsupervised Code Translation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=cmt-6KtR4c4

[34] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[35] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2024. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation.

IEEE Transactions on Software Engineering 50, 1 (2024), 85–105. https://doi.org/10.1109/TSE.2023.3334955

[36] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I Wang. 2022. Natural Language to Code Translation with Execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 3533–3546.

[37] Željka Stojanac, Daniel Suess, and Martin Kliesch. 2017. On products of Gaussian random variables. *arXiv preprint arXiv:1711.10516* (2017).

[38] A.B. Tsybakov. 2008. *Introduction to Nonparametric Estimation*. Springer New York. https://books.google.com.hk/books?id=mwB8rUBsbqoC

[39] Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, Shao Kun Deng, and Neel Sundaresan. 2021. Unit Test Case Generation with Transformers and Focal Context. arXiv:2009.05617 [cs.SE]

[40] Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, and Neel Sundaresan. 2022. Generating accurate assert statements for unit test cases using pretrained transformers. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test (AST '22)*. ACM. https://doi.org/10.1145/3524481.3527220

[41] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[42] Michael W Whalen. 2000. High-integrity code generation for state-based formalisms. In *Proceedings of the 22nd international conference on Software engineering*. 725–727.

[43] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics: Methodology and distribution*. Springer, 196–202.

[44] Xusheng Xiao, Sihan Li, Tao Xie, and Nikolai Tillmann. 2013. Characteristic studies of loop problems for structural test generation via symbolic execution. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 246–256. https://doi.org/10.1109/ASE.2013.6693084

[45] Tao Xie. 2006. Augmenting automatically generated unit-test suites with regression oracle checking. In *European Conference on Object-Oriented Programming*. Springer, 380–403.

[46] Jinqiu Yang, Alexey Zhikhartsev, Yuefei Liu, and Lin Tan. 2017. Better test cases for better automated program repair. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 831–841.

[47] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. 2024. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. arXiv:2305.04207 [cs.SE]

[48] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large Language Models Meet NL2Code: A Survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 7443–7464. https://doi.org/10.18653/v1/2023.acl-long.411

[49] Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-Tau Yih, Daniel Fried, and Sida Wang. 2023. Coder Reviewer Reranking for Code Generation. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 41832–41846. https://proceedings.mlr.press/v202/zhang23av.html