A Theoretical Analysis of Test-Driven Code Generation

Nicolas Menet[1,2] nicolas.menet@ibm.com
Michael Hersche[1] michael.hersche@ibm.com
Andreas Krause[2] krausea@ethz.ch
Abbas Rahimi[1] abr@zurich.ibm.com

[1]IBM Research – Zurich, [2]Department of Computer Science, ETH Zürich

Abstract

Code assistants are increasingly utilized in test-driven software development, yet the theoretical mechanisms behind their environment-interaction strategies remain underexplored. We provide a probabilistic framework for two dominant paradigms: code selection after generation using the execution environment, and code generation conditioned on environment feedback. First, we formalize several well-established selection heuristics as environment-aware estimators of code correctness. We theoretically prove that estimators based on fuzzy functional similarity add an inductive bias and strictly dominate estimators based on functional equivalence in terms of signal-to-noise ratio. Second, we frame backprompting as an in-context approximation of Thompson sampling. We derive a novel regret bound for reward functions with unobservable components, theoretically explaining why the effectiveness of backprompting is limited by the ambiguity of the informal task description (an irreducible regret). Using seven state-of-the-art open weight models, we corroborate these findings across BigCodeBenchHard, LeetCodeDataset, QiskitHumanEvalSim, and SWE-bench Verified. Our formalization also suggests how to improve task descriptions effectively, leading to an improved benchmark for LLM-assisted quantum programming, QiskitHumanEvalSimX.

1 Introduction

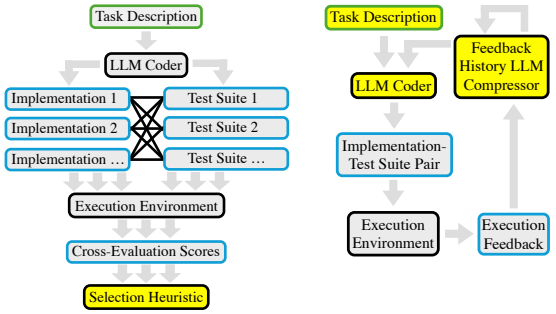
Large Language Models (LLMs) have demonstrated strong capabilities across several domains [OpenAI, 2023], including coding [Chen et al., 2021]. Building on these capabilities, code assistants have been integrated into disruptive products, such as GitHub Copilot, Claude Code, and Cursor.

With code assistants increasingly adopted for critical large-scale software, their capability to conduct test-driven development [Beck, 2003] becomes paramount. In addition to facilitating collaboration and quality control, test-driven development provides a unique opportunity for boosting LLM performance via post-generation selection [Li et al., 2022, Shi et al., 2022, Chen et al., 2023, Mündler et al., 2024, Li et al., 2025, Liu et al., 2026] or iterative backprompting [Shinn et al., 2023, Olausson et al., 2024, Chen et al., 2024c, Li et al., 2026]. Environment-aware post-generation selection leverages test suites to score implementations and group equally (or similarly) behaving code. In contrast, backprompting of execution results provides the language model with an opportunity to refine its own beliefs on the behavior of code. These strategies are illustrated in Figure 1. Although widely used, their theoretical mechanisms remain underexplored, resulting in a zoo of heuristics without a common foundation.

In this work, we adopt a probabilistic perspective on test-driven generation. Our contributions are: (1) We formalize post-generation selection heuristics as estimators of environment-aware probability of correctness, introducing a unifying categorization of prior works in Table 1. We prove that estimators adopting functional similarity add an inductive bias that diminishes neural approximation artifacts and strictly dominate estimators built on functional equivalence in terms of signal-to-noise ratio. This provides a theoretical basis for why aggregating "similar" behaviors is more robust than

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

counting identical outputs.
(2) We explore the limitations of conditioning the LLM on environment feedback during generation. To that end, we frame backprompting as an in-context approximation of Thompson sampling. We then derive a novel regret bound for reward functions with unobservable components. The bound identifies an irreducible regret: the effectiveness of backprompting is strictly limited by the ambiguity of the task description, regardless of the quantity of execution feedback. Lastly, we propose a practical estimator of the irreducible regret that may be used to trigger clarification requests.
(3) We confirm these findings experimentally using seven open weight models (Qwen3-235B-Instruct, Qwen3-Next, Qwen3-Next-Coder, Qwen3.5-397B, GPT-OSS-120B, MiniMax-M2.1, MiniMax-M2.7) on competitive coding (BigCodeBenchHard, LeetCodeDataset, and QiskitHumanEvalSim) and agentic coding (SWE-bench Verified). Our results confirm that "soft" estimators (adopting functional similarity) consistently outperform "hard" ones (adopting functional equivalence), and that backprompting is most effective in settings with a small unobservable reward component. Finally, we verify the importance of clear task descriptions for effective backprompting by deriving a novel dataset QISKITHUMANEVALSIMX.



(a) Section 3: Selection via Environment after Generation

(b) Section 4: Feedback during Generation

Figure 1: We consider environment interaction in two settings. The scope of our analysis is marked in yellow.

2 Preliminaries

Notation Let $\mathcal{V}$ be a finite vocabulary of tokens and $\mathcal{V}^{\leq L}$ the set of all token sequences up to a maximum output length L. We denote the number of test cases in a test suite $t \in \mathcal{V}^{\leq L}$ by $|t|$.

Problem setting We assume a generative process driven by a prior distribution over abstract *algorithms* $a \in \mathcal{A}$ and *environments* $e \in \mathcal{E}$ (execution contexts). The environment constrains the available tools (e.g., libraries) and required syntax to implement the algorithm. We observe a natural language task *description* $d \in \mathcal{V}^{\leq L}$ (the informal specification), which is a lossy compression of the algorithm a and environment e. Crucially, the abstract algorithm a is inherently unobservable beyond d. In contrast, the environment e can be partially observed. Given a description d and an observation history $\mathcal{H}$, we seek two latent variables: a *test suite* $t \in \mathcal{V}^{\leq L}$ (the executable specification) that formalizes the requirements of algorithm a in environment e, and an *implementation* $c \in \mathcal{V}^{\leq L}$ valid under this test suite. We model $p(c, t \mid \mathcal{H}, d)$ with a large language model and shorten the notation by omitting the dependency on d. Figure 2 represents the generative process as a structural causal model.

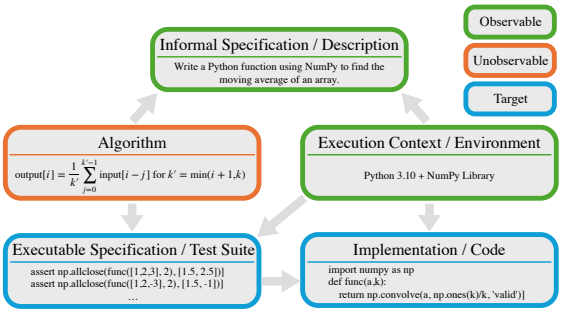


Figure 2: Generative process represented as a structural causal model (arrows indicate causality). We observe an informal specification (description) corresponding to an algorithm in an execution context (environment). We seek an executable specification (test suite) and an implementation that satisfies it (code).

Environment feedback Given an environment $e \in \mathcal{E}$, code $c \in \mathcal{V}^{\leq L}$, and test suite $t \in \mathcal{V}^{\leq L}$, we consider both scalar and textual environment feedback. Let $\mathbb{O}$ be a finite vocabulary of test outputs. Then, $O(c, t \mid e) \in \mathbb{O}^{|t|}$ is the direct output of the test harness. Typically, $\mathbb{O} = \{0, 1\}$ with 0 indicating failure and 1 success. In this case, a scalar reward is recovered via $R(c, t \mid e) := \frac{1}{|t|} \sum_i O_i(c, t \mid e) \in [0, 1]$. More generally, strategies such as functional consensus rely on generalized test suites with large $\mathbb{O}$, e.g., corresponding to arbitrary bounded function output. Lastly, more informative textual feedback can be obtained from a test report $U(c, t \mid e) \in \mathcal{V}^{\leq L}$, e.g., in the form of captured *stderr*

and assertion tracebacks. In the following, we adopt an environment-centric view to first formalize popular post-generation selection mechanisms that leverage the environment to establish fine-grained consensus, before examining in-the-loop strategies based on backprompting of environment feedback.

3 Selection via environment after generation

Code generation should target the underlying *algorithm* rather than a specific implementation. Yet, autoregressive next-token prediction leads to probability mass being fragmented across functionally identical implementations—an issue known as *program aliasing* [Bunel et al., 2018, Zhong et al., 2018]. To remedy such modeling artifacts, selection strategies must operate at an algorithmic granularity that groups equally behaving codes. In this section, we first define key concepts such as functional similarity and functional equivalence. We then establish important properties such as their signal-to-noise ratio, and conclude that functional similarity is significantly more robust. Based on our theory, we then unify several prior works in Table 1. All proofs, including proof sketches, are in Appendix E. Moreover, the appropriateness of each adopted assumption is discussed in Section E.1.

Definition 3.1 (Functional code s-similarity)**.** Given an environment $e \in \mathcal{E}$, a distribution over test suites $p(t)$, and a sharpness exponent $s \in \mathbb{N}$, we define similarity between implementations c_1, c_2 as

$$\mathrm{sim}^s_{p,e}(c_1, c_2) := \left(\mathbb{E}_{t\sim p}\left[\tfrac{1}{|t|}\sum_{k=1}^{|t|} \mathbb{1}_{O_k(c_1,t|e)=O_k(c_2,t|e)}\right]\right)^s,$$

where $O_k(c, t|e)$ denotes the output of the k-th test case in a test suite t for the implementation c.

Note that this similarity implicitly depends on the task description d via $p(t)$. This dependency focuses the functional comparison on the intended input domain, diminishing fragmentation due to divergent handling of edge cases undefined by d, and reducing variance of Monte Carlo estimation.

Definition 3.2 (Functional code equivalence)**.** Given an environment $e \in \mathcal{E}$ and a distribution over test suites $p(t)$, we define the equivalence of two implementations c_1, c_2 as

$$\mathrm{sim}^\infty_{p,e}(c_1, c_2) := \mathbb{1}_{O(c_1,t|e)=O(c_2,t|e)\ \forall t\in\mathcal{V}^{\le L}:p(t)>0}.$$

The notation is deliberate. In addition to a kernel structure that will be central to proving Theorem 3.8, we can prove that functional s-similarity converges to functional equivalence as $s \to \infty$.

Proposition 3.3 (Properties of s-similarity)**.** *The similarity* $\mathrm{sim}^s_{p,e}(c_1, c_2)$ *is a positive semi-definite kernel with* $\mathrm{sim}^s_{p,e} \ge 0$ *and* $\mathrm{sim}^s_{p,e}(c, c) = 1$*. Moreover,* $\mathrm{sim}^\infty_{p,e}(c_1, c_2) = \lim_{s\to\infty} \mathrm{sim}^s_{p,e}(c_1, c_2)$.

3.1 Functional similarity neighborhoods induce measure smoothing

Functional equivalence is the canonical structure we must impose on $\mathcal{V}^{\le L}$ to factor out redundant representations, but functional similarity offers a smoother objective that can be used to reduce approximation error (see Section 3.2). Functional code similarity defines similarity neighborhoods:

Definition 3.4 (Fuzzy similarity neighborhood)**.** For any code $c \in \mathcal{V}^{\le L}$, let $\mathcal{N}^s_c$ be the fuzzy set representing the neighborhood of implementations functionally similar to implementation c. It is defined by the membership function $\mu_{\mathcal{N}^s_c} : \mathcal{V}^{\le L} \to [0, 1]$ where $\mu_{\mathcal{N}^s_c}(x) := \mathrm{sim}^s_{p,e}(c, x)$.

For functional equivalence ($s \to \infty$), the sets cease to be fuzzy and become strict equivalence classes.

Proposition 3.5 (Functional equivalence classes)**.** *Define* $\mathcal{N}^\infty_c = \{c' \in \mathcal{V}^{\le L} : \mathrm{sim}^\infty_{p,e}(c, c') = 1\}$. *Then* $\{\mathcal{N}^\infty_c : c \in \mathcal{V}^{\le L}\}$ *is a set of equivalence classes partitioning* $\mathcal{V}^{\le L}$.

We now lift the concept of functional s-similarity to the distribution over implementations $p(c)$.

Definition 3.6 (Smoothed code distribution)**.** Given a distribution over implementations $p(c)$, we define the probability measure of a fuzzy similarity neighborhood $\mathcal{N}^s_c$ as expected membership:

$$p(\mathcal{N}^s_c) := \mathbb{E}_{c'\sim p}[\mu_{\mathcal{N}^s_c}(c')] = \textstyle\sum_{c'\in\mathcal{V}^{\le L}} p(c')\mathrm{sim}^s_{p,e}(c, c').$$

This measure aggregates the probability mass of all functionally similar implementations, effectively smoothing the distribution over codes. It allows us to consider the probability of program behavior rather than focusing on specific token strings. In the limit of strict equivalence, $p(\mathcal{N}^\infty_c)$ recovers the probability of strict behavioral equivalence classes. However, as we discuss next, Monte Carlo estimation of strict equivalence classes is biased and suffers from high variance, whereas smoothing reduces the former by linearization and the latter by aggregating information from similar implementations.

3.2 Estimating functional similarity is more robust than estimating functional equivalence

In practice, precisely computing $\mathrm{sim}^s_{p,e}$ and $p(\mathcal{N}^s_c)$ is prohibitively expensive due to requiring integration over the combinatorial $\mathcal{V}^{\leq L}$. An effective fallback is provided by Monte Carlo estimation:

Definition 3.7. For $c_i \sim p(c)$ and $t_j \sim p(t)$ i.i.d., we define the Monte Carlo estimators:

$$\widehat{\mathrm{sim}}^s_{p,e;m}(c_1, c_2) := \left(\tfrac{1}{m}\sum_{j=1}^{m}\tfrac{1}{|t_j|}\sum_k \mathbb{1}_{O_k(c_1,t_j|e)=O_k(c_2,t_j|e)}\right)^s, \widehat{p}_{n,m}(\mathcal{N}^s_c) := \tfrac{1}{n}\sum_{i=1}^{n}\widehat{\mathrm{sim}}^s_{p,e;m}(c, c_i).$$

Thus far, we have introduced Monte Carlo estimators of algorithmic code correctness $p(\mathcal{N}^s_c)$ for any sharpness coefficient $s \in \mathbb{N}$. But what is the benefit of smooth estimators? Since we assume p to be parameterized by an LLM, measure smoothing is desirable for three key reasons: (1) it introduces a beneficial inductive bias that diminishes neural approximation artifacts, (2) it reduces the Monte Carlo estimator's bias, and (3) it significantly increases the estimator's signal-to-noise ratio (SNR).

Theorem 3.8 (Measure smoothing inductive bias)**.** *Let* $\mathrm{sim}^s_{p,e}(c, c') \geq 1 - \varepsilon$ *for* $\varepsilon \in (0, 1)$*. Then*

$$|p(\mathcal{N}^s_c) - p(\mathcal{N}^s_{c'})| \leq \sqrt{2\varepsilon}.$$

Here, the sharpness exponent s determines the granularity at which two solutions are considered similar and thus tunes the strength of the environment-dependent inductive bias.

Theorem 3.9 (Consistency and bias)**.** $\widehat{\mathrm{sim}}^s_{p,e;m}$ *is a strongly consistent estimator of* $\mathrm{sim}^s_{p,e}$*, i.e.,* $\mathbb{P}[\lim_{m\to\infty}\widehat{\mathrm{sim}}^s_{p,e;m}(c_1, c_2) = \mathrm{sim}^s_{p,e}(c_1, c_2)] = 1\ \forall c_1, c_2 \in \mathcal{V}^{\leq L}$*. Yet, it is biased unless* $s = 1$*.*

Here, the bias arises from Jensen's inequality combined with the unbiasedness of $\widehat{\mathrm{sim}}^1_{p,e;m}$.

Theorem 3.10 (SNR dominance)**.** *Let* $\mathrm{sim}^1_{p,e}(c_1, c_2) = \mu \in (0, 1)$ *be the true functional similarity. We define the signal-to-noise Ratio (SNR) of an estimator* $\widehat{X}$ *as* $\mathrm{SNR}(\widehat{X}) := (\mathbb{E}[\widehat{X}])^2/\mathrm{Var}(\widehat{X})$*. For any number of test suites* $m \geq 1$*, the SNR of the soft estimator dominates the hard estimator by a factor of at least* m^2*:*

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^1_{p,e;m}(c_1, c_2))/\mathrm{SNR}(\widehat{\mathrm{sim}}^\infty_{p,e;m}(c_1, c_2)) \geq m(1/\mu)^{m-1} \cdot \tfrac{1-\mu^m}{1-\mu} \geq m^2$$

Theorem 3.10 contains a uniform lower bound of m^2, and the much stronger μ-dependent lower bound in $\Omega(m(1/\mu)^m)$. Thus, the SNR dominance is strongest for small μ, i.e., for diverse generations.

3.3 A dual expression for code correctness

Revisit the generative process of Figure 2. Given an environment $e \in \mathcal{E}$ and an executable specification $t \in \mathcal{V}^{\leq L}$, a correct implementation $c \in \mathcal{V}^{\leq L}$ is such that $R(c, t \mid e) = 1$. Thus, code correctness can be estimated via $p(\mathcal{N}^\infty_c)$, or, alternatively, via $\mathbb{E}_{t\sim p}[R(c, t \mid e)^\infty]$. Both are equally valid a priori.

Definition 3.11 (Code-test calibration)**.** A model $p(c, t)$ is code-test calibrated in an environment $e \in \mathcal{E}$ if

$$p(\mathcal{N}^\infty_c) = \mathbb{P}_{t\sim p}[R(c, t \mid e) = 1].$$

Note that if a model is code-test calibrated, we expect equal downstream performance with both the primal and dual expressions. Yet, as our experiments in Section 5.2 demonstrate, current frontier models are miscalibrated in practice. In analogy to $p(\mathcal{N}^s_c)$, a smooth estimator $\mathbb{E}_{t\sim p}[R(c, t \mid e)^s]$ is also considered. The SNR dominance of Theorem 3.10 still applies, motivating the smooth variant.

3.4 Popular selection heuristics as self-consistency

Our framework allows us to identify several well-established selection heuristics as maximizing an estimate of the probability of correctness $p(\mathcal{N}^\infty_c)$. Table 1 provides a summary of the code selection literature based on our formalism, with Appendix B.1 providing more information. In the experiments Section 5, we focus on boolean test outputs $\mathbb{O}$, i.e., the canonical case of test-driven development.

Table 1: Selection heuristics maximize pass@1 using different estimators of code correctness. The test output $\mathbb{O}$ is binary $\mathbb{B}$ (pass/fail) or arbitrary $\mathbb{A}$. The asterisk marks that only single-test test suites were considered by Lahiri et al. [2022], Rozière et al. [2022], Le et al. [2022], Chen et al. [2024b].

	$\mathbb{O}$	selection criterion		$\mathbb{O}$	selection criterion
MBR-EXEC HARD [Shi et al., 2022]	$\mathbb{B}$	$\hat{p}_{n,m}(\mathcal{N}_c^\infty)$	MBR-EXEC SOFT [Shi et al., 2022]	$\mathbb{B}$	$\hat{p}_{n,m}(\mathcal{N}_c^1)$
ALPHACODE [Li et al., 2022]	$\mathbb{A}$	$\hat{p}_{n,m}(\mathcal{N}_c^\infty)$	FUNCODER [Chen et al., 2024a]	$\mathbb{A}$	$\hat{p}_{n,m}(\mathcal{N}_c^1)$
MAXPASS HARD (various works*)	$\mathbb{B}$	$\hat{\mathbb{E}}_{t_{1:m}}[R^\infty]$	MAXPASS SOFT (ours)	$\mathbb{B}$	$\hat{\mathbb{E}}_{t_{1:m}}[R]$
CODET HARD [Chen et al., 2023]	$\mathbb{B}$	$\hat{p}_{n,m}(\mathcal{N}_c^\infty)\hat{\mathbb{E}}_{t_{1:m}}[R]$	CODET SOFT (ours)	$\mathbb{B}$	$\hat{p}_{n,m}(\mathcal{N}_c^1)\hat{\mathbb{E}}_{t_{1:m}}[R]$

3.5 Maximizing pass@k via probability of correctness

Our analysis focuses on robust estimators of code correctness, i.e., pass@1. Nevertheless, as Proposition 3.12 shows, such estimators can also be utilized to directly maximize pass@k.

Proposition 3.12 (Additivity of probability of correctness)**.** *Suppose that the true test suite $t \in \mathcal{V}^{\leq L}$ verifying an implementation $c \in \mathcal{V}^{\leq L}$ in environment $e \in \mathcal{E}$ is almost surely such that $\exists!$ correct $\mathcal{N}_c^\infty$. Define* $\text{pass@k}(\{\mathcal{N}_{c_i}^\infty\}_{i=1}^k) := \mathbb{P}_{t\sim p}[\exists i \in \{1,\ldots,k\} : R(c_i, t \mid e) = 1]$*. Then,* pass@k *is maximized by greedily selecting the k code classes $\{\mathcal{N}_{c_i}^\infty\}_{i=1}^k$ with largest* pass@1*.*

4 Environment feedback during generation

Having established a unifying framework for environment-based post-generation selection, we now turn to environment feedback *during* generation. Recall from Figure 2 that the executable specification (test suite) and the algorithm implementation depend causally on the latent abstract algorithm $a \in \mathcal{A}$ and the execution environment $e \in \mathcal{E}$. Whereas the abstract algorithm remains unobservable beyond the task description $d \in \mathcal{V}^{\leq L}$, the environment can be probed to retrieve properties like *syntax rules* and *library behaviors* via execution results $R(c, t \mid e) \in [0, 1]$ or descriptive reports $U(c, t \mid e) \in \mathcal{V}^{\leq L}$.

4.1 Backprompting approximates Thompson sampling

To permit theoretical analysis of backprompting, we model the iterative refinement loop as a sequential bandit problem where the LLM acts as an in-context conditioned policy. We define an action $x := (c, t) \in \mathcal{V}^{\leq L} \times \mathcal{V}^{\leq L}$ as a pair consisting of an implementation and a test suite. Thompson sampling then selects an action x according to the probability that it is the optimal solution x^* given the task d and observation history $\mathcal{H}_n$, i.e., its policy is $\pi(x \mid \mathcal{H}_n, d) := \mathbb{P}[x = x^* \mid \mathcal{H}_n, d]$.

We argue that LLMs naturally learn to approximate this policy. Pre-training data for code often contains description-attempt-solution triplets $(d, \mathcal{H}, x^*)$—for example, software repositories where commit histories document failed attempts followed by a verified solution. Following Sutter et al. [2025], training a sequence model on such trajectories with a cross-entropy loss forces the model to minimize the Kullback-Leibler divergence to the true conditional distribution of the optimum:

$$\begin{aligned}\theta^* &:= \operatorname{argmin}_\theta \mathbb{E}_{(d,\mathcal{H},x^*)\sim p_{\text{data}}}[-\log p_\theta(x^* \mid \mathcal{H}, d)] \\ &= \operatorname{argmin}_\theta \mathbb{E}_{(d,\mathcal{H})\sim p_{\text{data}}}[D_{KL}(p_{\text{data}}(x^*|\mathcal{H}, d)||p_\theta(x^*|\mathcal{H}, d))].\end{aligned}$$

Hence, by backprompting failed attempts during generation, we sample $x_{n+1} \sim p_\theta(\cdot \mid \mathcal{H}_n, d)$, which, due to pre-training, results in drawing from the model's learned belief over the optimal solution x^*. To empirically verify this approximation, Appendix C.8 provides indirect evidence of the LLM's sampling distribution: the generated programs & test suites remain diverse yet stable across rounds, with the model's implicit posterior belief naturally contracting as execution feedback accumulates.

This equivalence allows us to lift regret bounds from Bayesian optimization to our setting. We establish a novel regret bound for Thompson sampling which accounts for unobservable reward components, reflecting irreducible under-specification of the abstract algorithm in the task description.

4.2 A Bayesian regret bound for Thompson sampling with unobservable reward components

Suppose a reward function $r(x) = r_{obs}(x) + r_{hid}(x)$. For us, $r(c, t) = 1$ if and only if c correctly implements the algorithm, i.e., if $R(c, t_{true} \mid e) = 1$. Now, $r_{obs}(c, t) = R(c, t \mid e)$ is an observable consistency score between codes, tests, and environments. In contrast, $r_{hid}(c, t)$ is a corrective factor $r(c) - r_{obs}(c, t)$ that captures the task relevance of (c, t) to the unobservable latent algorithm $a \in \mathcal{A}$,

partially described by $d \in \mathcal{V}^{\leq L}$. Note that test pass rates r_{obs} are bounded in $[0, 1]$ and thus inherently subgaussian by Hoeffding's Lemma. The same holds for $r_{hid} \in [-1, 1]$ as the algebraic residual.

We consider the regret against the true optimal solution $x^* := \arg\max_x r(x)$ and against Thompson sampling with full knowledge of the environment, i.e., against $x^*_{obs} \sim \mathbb{P}[x^* \mid r_{obs}]$.

Theorem 4.1. *Decompose the reward into independent components $r := r_{obs} + r_{hid}$, where $\mathbb{E}[r_{hid}]$ is finite, and $r_{obs}(x)$ is $\sigma_n(x)$-subgaussian given observation history $\mathcal{H}_n := (x_i, r_{obs}(x_i))_{i=1}^{n-1}$. Define the Thompson sampling policies $x^*_{obs} \sim \mathbb{P}[x^* \mid r_{obs}]$ and $x_n \sim \mathbb{P}[x^* \mid \mathcal{H}_n]$ for $x^* := \arg\max_x r(x)$. Then*

$$\mathbb{E}[\textstyle\sum_{n=1}^{T} r(x^*_{obs}) - r(x_n)] \leq \beta\sqrt{T \cdot \Gamma_T}$$

for $\beta := \sqrt{2\log(|\mathrm{dom}(r)|)} + 3$ and cumulative uncertainty $\Gamma_T := \mathbb{E}[\sum_{n=1}^{T} \sigma_n^2(x_n)]$.

Interpretation If r_{obs} has independent entries, then the cumulative uncertainty Γ_T is upper bounded by $\sum_{x \in \mathrm{dom}(r)} \sigma_1^2(x)$. Conversely, if r_{obs} has correlated entries, we expect faster convergence. Indeed, for correlated strictly subgaussian rewards, observing $r_{obs}(x_n)$ reduces uncertainty at $x \neq x_n$ in expectation (see Lemma E.2). For Gaussian r_{obs}, we can formalize the benefit of correlation information in closed form: according to Lemma E.3, Γ_T is upper bounded by the maximal information gain [Srinivas et al., 2010], which reflects the true degrees of freedom of the function r_{obs}. E.g., if r_{obs} is a linear function of D features (with standard Gaussian weights), $\Gamma_T \in \mathcal{O}(D \log T)$.

The regret in Theorem 4.1 is stated with respect to a Thompson sampling policy with full knowledge of r_{obs}. The regret with respect to the ground-truth optimal solution follows:

Corollary 4.2. *Consider the setting of Theorem 4.1. Then for $\Delta := \mathbb{E}[r(x^*) - r(x^*_{obs})]$ it holds that*

$$\mathbb{E}[\textstyle\sum_{n=1}^{T} r(x^*) - r(x_n)] \leq \beta\sqrt{T \cdot \Gamma_T} + T\Delta.$$

Interpretation The bound decomposes cumulative regret into two distinct terms. The first term, $\mathcal{O}(\sqrt{T})$, is sub-linear and represents the *learnable* part of the problem: by interacting with the environment (running tests on implementations), the assistant reduces its uncertainty about r_{obs} (syntax, library behavior, etc.). The second term, $T\Delta$, is linear. It represents the *irreducible regret*: even if the assistant perfectly understands the execution environment (r_{obs}), the informal description d may still be ambiguous regarding the true latent algorithm a. No amount of execution feedback can resolve this specific ambiguity. Δ can be directly estimated by measuring the performance gap at convergence between an oracle setting (where t_{true} is provided such that $r_{hid} = 0$ and thus $x^*_{obs} = x^*$) and a self-test setting. In practice, the oracle is typically unavailable. As an alternative, Section 4.3 provides a simple estimator for Δ via the collision probability of functional equivalence.

Predicting reward Given tight regret bounds, a saturated $\Gamma_T \approx \Gamma$, and with $\sqrt{T}$-$\sqrt{T\text{-}1} \approx T^{\text{-}1/2}/2$, we get the behavior

$$\mathbb{E}[r(x_T)] \approx \mathbb{E}[r(x^*)] - \Delta - T^{-1/2} \cdot \beta \cdot \sqrt{\Gamma}/2. \tag{1}$$

We thus expect in-context Thompson sampling to follow a reward curve that approaches a ceiling determined by Δ at a rate of $\Theta(1/\sqrt{T})$. As Figure 3 demonstrates, this indeed occurs if the irreducible regret Δ is small and does not dominate the learnable part (e.g., purple lines across all three datasets).

4.3 Estimating irreducible regret during backprompting

Suppose $r(c) \in \{0, 1\}$ is 1 if and only if $R(c, t_{true} \mid e) = 1$. Then, the irreducible regret Δ satisfies

$$\begin{aligned} 1 - \Delta &= 1 - \mathbb{E}[r(x^*) - r(x^*_{obs})] = \mathbb{E}[r(x^*_{obs})] = \mathbb{P}[R(c^*_{obs}, t_{true} \mid e) = 1] \\ &= \mathbb{E}_{c^*_{obs}}[\mathbb{E}_{t_{true}}[\mathbb{1}_{R(c^*_{obs}, t_{true} \mid e)=1}]] = \mathbb{E}_{c^*_{obs}}[p(\mathcal{N}^{\infty}_{c^*_{obs}})] = \mathbb{E}_{c^*_{obs,1}, c^*_{obs,2}}[\mathrm{sim}^{\infty}_{p,e}(c^*_{obs,1}, c^*_{obs,2})]. \end{aligned}$$

All expectations are understood conditioned on full knowledge of r_{obs}. Note that the expression for $1 - \Delta$ corresponds to the collision probability of the equivalence classes $\mathcal{N}^{\infty}_c$, whose Monte Carlo estimator can be derived from Section 3.2. To sample x^*_{obs}, a moving window $(x_n)_{n=T-w+1}^{T}$ of size w may be used upon convergence of the Thompson sampling policy $\pi_n(x) = \mathbb{P}[x = x^* \mid \mathcal{H}_n]$. If Δ exceeds a user-defined threshold, clarifications to resolve the task ambiguity can be requested.

5 Experiments

To corroborate our theory, we empirically compare the selection heuristics unified in Section 3, and confirm the fundamental limits of in-the-loop backprompting of environment feedback, established theoretically in Section 4 via identification of backprompting with approximate in-context Thompson sampling. Of particular interest is the impact of the sharpness factor s and irreducible regret Δ.

5.1 Setup

Datasets We consider three challenging python competitive coding datasets. *BigCodeBenchHard* is the partition of the 148/1140 most difficult tasks in BigCodeBench [Zhuo et al., 2025]. It adopts the format of HumanEval [Chen et al., 2021], but covers interaction with 139 external libraries. *QiskitHumanEvalSim* is a novel subset of the 143/151 QiskitHumanEval [Vishwakarma et al., 2024] questions that can be solved without access to a real-world quantum computer, democratizing the dataset. Finally, *LeetCodeDataset* consists of 228 diverse puzzle-style questions from LeetCode released after 2024-07-01 and collected by Xia et al. [2025]. In contrast to the other datasets, LeetCodeDataset typically includes example input-output pairs in natural language, significantly mitigating the irreducible regret Δ identified in Corollary 4.2. Lastly, we consider agentic coding with *SWE-bench Verified* [Jimenez et al., 2024], a benchmark where LLM agents must navigate Github code repositories and resolve real-world software issues. For more information, see Appendix D.1.

Models We consider seven open-weight LLMs: Qwen3-235B-A22B-Instruct-2507 [Qwen Team, 2025], the hybrid Transformer-SSM models: Qwen3-Next, Qwen3-Coder-Next, and Qwen3.5-397B-A17B [Qwen Team, 2026], GPT-OSS-120B [OpenAI, 2025], and MiniMax-M2.1 as well as MiniMax-M2.7 [MiniMax, 2025]. Whereas the Qwen3 models respond immediately, the others (including Qwen3.5) are chain-of-thought reasoning models. Only MiniMax-M2.1, MiniMax-M2.7, and Qwen3-Coder-Next are dedicated coding models. More information is in Appendix D.2.

5.2 Post-generation selection heuristics

Consider Table 2 (and its more detailed version in Figure S1 of Appendix C.1), reporting on the accuracy of three models (Qwen3-235B-A22B-Instruct-2507, GPT-OSS-120B, MiniMax-M2.1) achieved through selection heuristics applied after 10 independent code and test suite generations. The accuracy improvements over RANDOM SELECTION are highly sensitive to the post-generation selection method, with significant gains achieved by MAXPASS and CODET strategies. In particular, note that soft estimators generally outperform hard estimators. This corroborates the theoretical insights of Section 3.2: soft estimators of probability of correctness benefit from a smoothness inductive bias and from an improved SNR. In Appendix C.9, we isolate this effect explicitly: while hard estimators exhibit slightly better rank correlation with oracle correctness, soft estimators achieve vastly higher empirical SNR, directly mediating the observed pass@1 improvements. Table 2 also reveals that MAXPASS consistently outperforms MBR-EXEC across datasets and models. This performance gap provides empirical evidence that current frontier models are not perfectly code-test calibrated. Because the theoretical duality between code-based and test-based estimation breaks down in practice, our results indicate that the test-based dual estimation of code correctness (MAXPASS)

Table 2: Pass@1 (%) of post-generation selection heuristics (see Table 1) after 10 rounds of independent code and test generation ($m = n = 10$). Within a family of heuristics, we bold the more accurate one. The soft variants generally outperform the hard ones, corroborating the insights of Theorem 3.8, Theorem 3.9, and Theorem 3.10. We report mean ($\pm$ standard error) over 5 seeds.

	BigCodeBenchHard (BigCode)			QiskitHumanEvalSim (Qiskit)			LeetCodeDataset (LeetCode)		
	Qwen3	GPT-OSS	MM-M2.1	Qwen3	GPT-OSS	MM-M2.1	Qwen3	GPT-OSS	MM-M2.1
RANDOM SELECTION	$27.43^{\pm 0.31}$	$37.16^{\pm 0.63}$	$34.46^{\pm 0.45}$	$52.73^{\pm 0.19}$	$41.96^{\pm 0.36}$	$43.78^{\pm 0.47}$	$18.60^{\pm 0.22}$	$69.12^{\pm 0.31}$	$62.54^{\pm 0.58}$
MBR-EXEC HARD	$29.32^{\pm 0.43}$	$37.30^{\pm 0.42}$	$35.95^{\pm 0.55}$	$54.97^{\pm 0.13}$	$\mathbf{47.69}^{\pm 0.40}$	$45.73^{\pm 0.34}$	$18.33^{\pm 0.18}$	$\mathbf{72.81}^{\pm 0.41}$	$73.07^{\pm 0.33}$
MBR-EXEC SOFT	$\mathbf{29.73}^{\pm 0.42}$	$37.30^{\pm 0.54}$	$\mathbf{36.76}^{\pm 0.31}$	$54.97^{\pm 0.16}$	$47.13^{\pm 0.35}$	$\mathbf{46.71}^{\pm 0.23}$	$\mathbf{18.60}^{\pm 0.16}$	$72.19^{\pm 0.32}$	$\mathbf{73.33}^{\pm 0.45}$
CODET HARD	$31.22^{\pm 0.32}$	$37.30^{\pm 0.24}$	$36.08^{\pm 0.59}$	$59.02^{\pm 0.19}$	$51.33^{\pm 0.38}$	$50.49^{\pm 0.23}$	$21.23^{\pm 0.12}$	$73.25^{\pm 0.36}$	$78.60^{\pm 0.24}$
CODET SOFT (ours)	$\mathbf{35.00}^{\pm 0.36}$	$\mathbf{38.24}^{\pm 0.34}$	$\mathbf{42.03}^{\pm 0.42}$	$\mathbf{62.24}^{\pm 0.28}$	$\mathbf{54.13}^{\pm 0.27}$	$\mathbf{53.85}^{\pm 0.33}$	$\mathbf{24.30}^{\pm 0.23}$	$\mathbf{74.56}^{\pm 0.49}$	$\underline{\mathbf{79.56}}^{\pm 0.24}$
MAXPASS HARD	$33.51^{\pm 0.45}$	$37.30^{\pm 0.46}$	$39.59^{\pm 0.39}$	$61.40^{\pm 0.15}$	$48.53^{\pm 0.40}$	$51.75^{\pm 0.22}$	$24.82^{\pm 0.26}$	$70.09^{\pm 0.54}$	$76.32^{\pm 0.50}$
MAXPASS SOFT (ours)	$\mathbf{35.41}^{\pm 0.38}$	$\mathbf{38.24}^{\pm 0.34}$	$\underline{\mathbf{42.30}}^{\pm 0.31}$	$\underline{\mathbf{63.22}}^{\pm 0.19}$	$\mathbf{54.13}^{\pm 0.27}$	$\underline{\mathbf{54.13}}^{\pm 0.34}$	$\mathbf{25.09}^{\pm 0.29}$	$\mathbf{74.56}^{\pm 0.49}$	$\mathbf{79.21}^{\pm 0.30}$

is the superior, more robust heuristic for current uncalibrated models given binary test outputs of test-driven development. Finally, note that Appendix C.10 demonstrates that MAXPASS SOFT outperforms B4 [Chen et al., 2024b], a recent Bayesian selection baseline relying on strict equivalence.

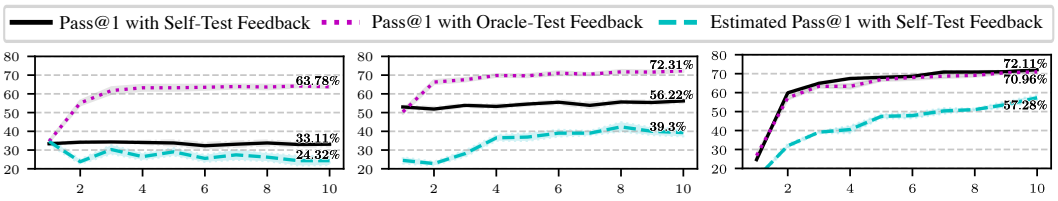
5.3 Approximate in-context Thompson sampling via iterative backprompting

Next, let us consider conditioning the LLM during generation on execution results from running a potential implementation c on a potential test suite t. Based on the development in Section 4, we provide the LLM in-context with an informative test report $U(c, t \mid e) \in \mathcal{V}^{\leq L}$. Since test reports can become very long, direct backprompting of all information from multiple rounds quickly becomes computationally infeasible. Thus, we compress $U(c, t \mid e)$ into a much shorter token sequence using the LLM. We ablate on the compression technique and the factorization of $p(c, t)$ in Appendix C.4. This section conducts the main experiments using Qwen3-235B-A22B-Instruct-2507, with findings on other models, such as their low test quality (GPT-OSS-120B and MiniMax-M2.1) and positive generalization across architectures within the Qwen family (Qwen3-Next and Qwen3-Coder-Next, Qwen3.5-397B-A17B), discussed at the end. Now, examine Figure 3, where we compare three settings:

Pass@1 with ORACLE-TEST FEEDBACK First, we consider the simplified setting where the test suite t_{oracle} is given and the model only has to find a code c such that $R(c, t_{oracle} \mid e) = 1$. In the formalism of Theorem 4.1, this corresponds to the setting where r_{hid} is known, which according to Corollary 4.2 permits global convergence to x^*. We observe that under these circumstances, the LLM acts as a highly effective approximate Thompson sampler, finding solutions that are much better than what can be achieved with post-selection heuristics. Indeed, in-context Thompson sampling achieves a pass@1 of 63.78%, 72.31%, and 72.11% across BigCodeBenchHard, QiskitHumanEvalSim, and LeetCodeDataset, notably beating pass@10 of unconditioned generation: 46.35%, 67.83%, 28.60%. To tie into our theory, we may use Equation (1) (with $T = 1, 10$) to estimate the rate of information gain $\beta \cdot \sqrt{\Gamma}/2$. We obtain the values 42.69% (BigCode), 32.12% (Qiskit), and 64.66% (LeetCode).

Estimated pass@1 with SELF-TEST FEEDBACK Next, we let the language model generate both codes c and test suites t, but report on $R(c, t \mid e)$ instead of $R(c, t_{oracle} \mid e)$. Note that in the formalism of Theorem 4.1, the language model is asked to optimize $r = r_{obs} + r_{hid}$, but we only report on the trajectory of $r_{obs}(x_n)$. Although the LLM successfully optimizes r_{obs} on QiskitHumanEvalSim and LeetCodeDataset, it struggles to do so for BigCodeBenchHard, demonstrating that joint optimization of code and tests proves more challenging than only optimizing code, even for observable objectives.

Pass@1 with SELF-TEST FEEDBACK Finally, we consider the most challenging setting of optimizing $R(c, t_{oracle} \mid e)$ with only access to $R(c, t \mid e)$. Recall Corollary 4.2, which predicts a small uncertainty on r_{hid} required for success, i.e., the task description must contain sufficient information on the algorithmic behavior. Whereas on LeetCodeDataset the assistant recovers almost the same trajectory as if ORACLE-TEST FEEDBACK was present, the transfer on QiskitHumanEvalSim is more modest and completely missing on BigCodeBenchHard. Note that our theory actually predicts these findings: the task description on LeetCodeDataset includes input-output examples, decreasing the uncertainty of r_{hid} and thus the irreducible regret Δ. In fact, the difference in final performance between SELF-TEST



(a) BigCodeBenchHard (b) QiskitHumanEvalSim (c) LeetCodeDataset

Figure 3: Pass@1 (%) during 10 rounds of in-context Thompson sampling using Qwen3-235B-A22B-Instruct-2507. We report similar trends across other members of the Qwen3 family, including hybrids, in Appendix C.3. W.r.t. Theorem 4.1, the black solid line denotes the true reward r, and the cyan dashed line r_{obs}. The purple dotted line is the simplified setting of known r_{hid}, which, according to Corollary 4.2, permits global convergence to x^*. We report mean ($\pm$ standard error) over 5 seeds.

FEEDBACK and ORACLE-TEST FEEDBACK can be used as an estimator for the irreducible regret Δ of each dataset, giving 30.68%, 16.08%, and $\approx 0\%$, for BigCode, Qiskit, and LeetCode, respectively.

The importance of capturing the test manifold Appendix C.2 evaluates GPT-OSS-120B and MiniMax-M2.1 as in-context Thompson samplers. While they are highly effective optimizers under ORACLE-TEST FEEDBACK, both models show almost no improvement when backprompting SELF-TEST FEEDBACK. Moreover, the gap between self-estimated pass@1 and true pass@1 is substantially larger for these two models, indicating poorer modeling of the test manifold. In contrast, our primary model effectively captures this manifold. As detailed in Appendix C.7, the tests generated by Qwen3 provide a meaningful, discriminative evaluation signal with a typical bimodal spectrum of difficulty.

Generalization within Qwen family Motivated by Qwen3-235B-A22B's strong performance, we evaluate the hybrid 80B-A3B parameter models Qwen3-Next and Qwen3-Coder-Next and the hybrid Qwen3.5-397B-A17B (Appendix C.3). The strong self-test backprompting capabilities within the architecturally diverse Qwen3 model family suggests that neither model architecture nor size substantially limit the effectiveness of backprompting and instead hints at training as a potential limiting factor. This highlights a critical lesson for autonomous software engineering: to unlock the benefits of in-the-loop environment interaction, writing tests is just as important as writing code.

5.4 QiskitHumanEvalSimX

The central lesson of Section 5.3 is that the task description d must contain sufficient information on the abstract algorithm a such that the irreducible regret Δ does not dominate the total regret of in-context Thompson sampling. An effective strategy to improve algorithmic specification is to add input-output pairs to the task description. While LeetCodeDataset already contains such behavioral examples in the prompt, explaining the success of in-context Thompson sampling thereon, BigCodeBenchHard and QiskitHumanEvalSim do not. To further corroborate our theory, we introduce a new dataset: QISKITHUMANEVALSIM WITH E**X**AMPLES. This new dataset builds on QiskitHumanEvalSim by querying Gemini 3 Pro to add a natural language description of the oracle test to the task description, including input-output examples. The exact prompt is listed in Appendix D.1.4. As can be observed in Figure 4, in-context Thompson sampling indeed becomes more effective given the additional test description: instead of an accuracy delta over the first round of 3.21% we now report the delta 7.13%.

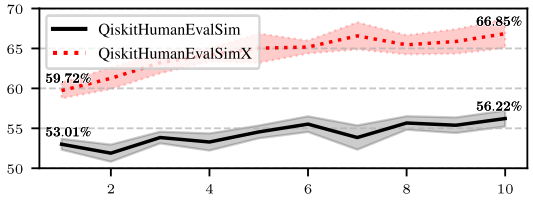


Figure 4: Pass@1 (%) during 10 rounds of backprompting on QiskitHumanEvalSimX. We use a Qwen3-235B-A22B-Instruct-2507. Results are reported as mean ($\pm$ standard error) over 5 seeds.

5.5 Estimating irreducible regret during in-context Thompson sampling

Having demonstrated in Section 5.3 and Section 5.4 that effective in-context Thompson sampling requires unambiguous task descriptions, we investigate if the irreducible regret Δ can be reliably estimated online, e.g., to trigger a task clarification from the user. As shown in Table 3, a Monte Carlo estimator of $\Delta = 1 - \mathbb{E}_{c^*_{obs,1}, c^*_{obs,2}}[\mathrm{sim}^{\infty}_{p,e}(c^*_{obs,1}, c^*_{obs,2})]$, developed in Section 4.3 (ONLINE ESTIMATOR, $w = 3$), roughly captures the oracle-based estimation of the irreducible regret $\Delta = \mathbb{E}[r(x_T) - r(x_T^{oracle})]$ from Section 5.3 (ORACLE ESTIMATOR). Here, we omit LeetCodeDataset since the Thompson sampling policy π_n has not yet converged (to $\mathbb{P}[x^* \mid r_{obs}]$) after 10 rounds, as evidenced by the non-saturated r_{obs} in Figure 3 (cyan dashed line).

Table 3: Online estimation of the irreducible regret Δ (%) on Qwen3-235B-A22B-Instruct-2507 trajectories. We report mean ($\pm$ standard error) over 5 seeds.

Method	BigCode	Qiskit
ORACLE ESTIMATOR	$30.68^{\pm 1.69}$	$16.08^{\pm 0.54}$
ONLINE ESTIMATOR	$27.07^{\pm 0.79}$	$11.66^{\pm 1.02}$

5.6 Extension to multi-step agentic code generation

Our central investigation deliberately isolates core components of coding agents [Yang et al., 2024a]—post-generation selection and iterative revision via environment feedback—within the scope of

competitive programming. This focus ensures controlled experiments with well-defined environment interactions, avoiding the confounding variables inherent in multi-file editing workflows. However, the theoretical lessons and findings presented above naturally extend to fully agentic systems. For instance, agentic benchmarks such as SWT-bench [Mündler et al., 2024] have successfully integrated post-generation selection heuristics based on CodeT to boost performance on repository-level tasks.

The concept of an unobservable reward component (r_{hid}) is also highly relevant in agentic backprompting. When agentic systems are evaluated on benchmarks like SWE-bench [Jimenez et al., 2024], providing oracle information by applying test patches to the project (a "reproduction" setting) effectively reveals r_{hid} to the agent. As predicted by our theory, eliminating the irreducible regret Δ can have a profound impact; notably, concurrent work by Li et al. [2026] shows that under such oracle information the performance of GPT-5 on SWE-bench Live improves from 30% to 75%.

To verify this phenomenon empirically in a repository-level agentic setting, we conduct experiments on SWE-bench Verified [Jimenez et al., 2024] using MiniMax M2.7 (see Figure 5). In this setting, the sequential bandit formulation adapts to a slightly more complex structure: while it still fits our contextual Thompson sampling framework, the actions are no longer as neat as a single code-test generation pair. Instead, a sequence of multiple discrete subactions (such as viewing files, invoking bash commands, and editing lines) collectively constitutes a full edit of the codebase, which is only subsequently verified via execution against self-tests.

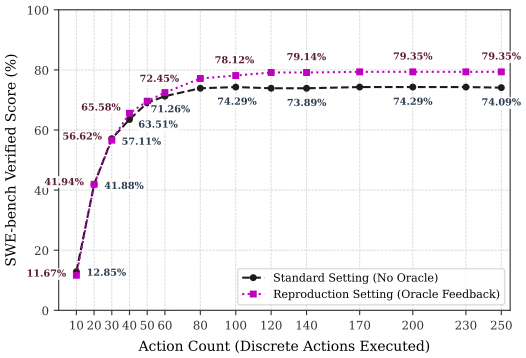


Figure 5: Solve rate (%) of MiniMax M2.7 during 250 environment interactions on SWE-bench Verified. The reproduction setting applies oracle test patches, and thus matches $\Delta = 0$ in Corollary 4.2.

Despite this multi-subaction abstraction layer, our framework's predictive power remains robust. By measuring the performance gap at convergence between the Reproduction Setting (ORACLE-TEST FEEDBACK) and the Standard Setting (SELF-TEST FEEDBACK), we can directly estimate the irreducible regret Δ, yielding 5.26%.

Interestingly, this positions SWE-bench Verified between QiskitHumanEvalSim ($\Delta = 16.08\%$) and LeetCodeDataset ($\Delta \approx 0\%$). The relatively low irreducible regret implies that real-world issue descriptions often contain rich contextual clues about the target algorithm, explaining the strong empirical success of multi-round backprompting agents when navigating repository-level codebases.

6 Conclusion

In this work, we provide a probabilistic framework for LLM-based test-driven code generation. We theoretically formalize two dominant paradigms: post-generation selection as environment-aware pass@k maximization, and iterative backprompting as in-context Thompson sampling. Our analysis proves that "soft" estimators strictly dominate "hard" ones in terms of SNR. Moreover, we identify an "irreducible regret" in the regret bound of Thompson sampling, where task ambiguity limits performance regardless of feedback quality. These theoretical insights are confirmed experimentally across four diverse benchmarks using seven different frontier open-weight large language models. Finally, a novel benchmark, QISKITHUMANEVALSIMX, is introduced to corroborate our findings. As LLMs are increasingly integrated into critical software development loops, understanding the theoretical mechanisms of their environment interactions becomes paramount. Our results suggest that progress lies not just in improving zero-shot model performance, but also in reducing the ambiguity of user intent and designing architectures that can efficiently process long execution histories.

Acknowledgment

This work is supported by the Swiss National Science Foundation (SNSF), grant 10002666.

References

Kent L. Beck. *Test-driven development - by example*. The Addison-Wesley signature series. Addison-Wesley, 2003.

Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *The 6th International Conference on Learning Representations*, 2018.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *The 11th International Conference on Learning Representations*, 2023.

Jingchang Chen, Hongxuan Tang, Zheng Chu, Qianglong Chen, Zekun Wang, Ming Liu, and Bing Qin. Divide-and-conquer meets consensus: Unleashing the power of functions in code generation. In *Advances in Neural Information Processing Systems*, volume 37, pages 67061–67105. Curran Associates, Inc., 2024a. doi: 10.52202/079017-2141.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé 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, Joshua 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. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. doi: 10.48550/arxiv.2107.03374.

Mouxiang Chen, Zhongxin Liu, He Tao, Yusu Hong, David Lo, Xin Xia, and Jianling Sun. B4: Towards optimal assessment of plausible code solutions with plausible tests. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 1693–1705. ACM, 2024b. doi: 10.1145/3691620.3695536.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to Self-Debug. In *The 12th International Conference on Learning Representations*, 2024c.

Leon Chlon, Sarah Rashidi, Zein Khamis, and MarcAntonio M. Awada. LLMs are Bayesian, in expectation, not in realization. *CoRR*, abs/2507.11768, 2025. doi: 10.48550/arxiv.2507.11768.

E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Shuvendu K. Lahiri, Aaditya Naik, Georgios Sakkas, Piali Choudhury, Curtis von Veh, Madanlal Musuvathi, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization. *CoRR*, abs/2208.05950, 2022. doi: 10.48550/arxiv.2208.05950.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu-Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 21314–21328. Curran Associates, Inc., 2022. doi: 10.48550/arXiv.2207.01780.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 15964–15978. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.865.

Kenan Li, Qirui Jin, Liao Zhu, Xiaosong Huang, Yijia Wu, Yikai Zhang, Xin Zhang, Zijian Jin, Yufan Huang, Elsie Nallipogu, et al. ORACLE-SWE: Quantifying the contribution of Oracle information signals on SWE agents. *CoRR*, abs/2604.07789, 2026. doi: 10.48550/arxiv.2604.07789.

Yujia Li, David H. 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. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Ren-Biao Liu, Jiang-Tian Xue, Chao-Zeng Ma, Hui Sun, Xin-Ye Li, and Ming Li. Dynamic-static synergistic selection method for candidate code solutions with generated test cases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32096–32104, 2026.

Nicolas Menet, Jonas Hübotter, Parnian Kassraie, and Andreas Krause. LITE: Efficiently estimating Gaussian probability of maximality. In *International Conference on Artificial Intelligence and Statistics*, volume 258, pages 4996–5004. Proceedings of Machine Learning Research, 2025.

Nicolas Menet, Andreas Krause, and Abbas Rahimi. POETS: Uncertainty-aware LLM optimization via compute-efficient policy ensembles. *Advances in Neural Information Processing Systems*, 39, 2026a.

Nicolas Menet, Aleksandar Terzic, Michael Hersche, Andreas Krause, and Abbas Rahimi. Thompson sampling via fine-tuning of LLMs. In *International Conference on Learning Representations*, 2026b.

MiniMax. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *CoRR*, abs/2506.13585, 2025. doi: 10.48550/arxiv.2506.13585.

Niels Mündler, Mark N Müller, Jingxuan He, and Martin Vechev. SWT-bench: Testing and validating real-world bug-fixes with code agents. *Advances in Neural Information Processing systems*, 37: 81857–81887, 2024. doi: 10.48550/arXiv.2406.12952.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is Self-Repair a silver bullet for code generation? In *The 12th International Conference on Learning Representations*, 2024.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arxiv.2303.08774.

OpenAI. GPT-OSS-120b & GPT-OSS-20b model card. *CoRR*, abs/2508.10925, 2025. doi: 10.48550/arxiv.2508.10925.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/arxiv.2505.09388.

Qwen Team. Qwen3-Coder-Next technical report. *CoRR*, abs/2603.00729, 2026. doi: 10.48550/arxiv.2603.00729.

Baptiste Rozière, Jie Zhang, François Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. Leveraging automated unit tests for unsupervised code translation. In *The 10th International Conference on Learning Representations*, 2022.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014. doi: 10.1287/moor.2014.0650.

Ehud Yehuda Shapiro. *Algorithmic program debugging*. Yale University, 1982.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *The 5th International Conference on Learning Representations*, 2017.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3533–3546. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.EMNLP-MAIN.231.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1015–1022. Omnipress, 2010.

Gustavo Sutter, Mohammed Abdulrahman, Hao Wang, Sriram Ganapathi Subramanian, Marc St-Aubin, Sharon O'Sullivan, Lawrence Wan, Luis A. Ricardez-Sandoval, Pascal Poupart, and Agustinus Kristiadi. Simplifying Bayesian optimization via in-context direct optimum sampling. *CoRR*, abs/2505.23913, 2025. doi: 10.48550/arxiv.2505.23913.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. doi: 10.2307/2332286.

Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Sanjay Vishwakarma, Francis Harkins, Siddharth Golecha, Vishal Sharathchandra Bajpe, Nicolas Dupuis, Luca Buratti, David Kremer, Ismael Faro, Ruchir Puri, and Juan Cruz-Benito. Qiskit HumanEval: An evaluation benchmark for quantum code generative models. In *IEEE International Conference on Quantum Computing and Engineering*, pages 1169–1176. IEEE, 2024. doi: 10.1109/QCE60285.2024.00137.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves Chain of Thought reasoning in language models. In *The 11th International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. LeetCodeDataset: A temporal dataset for robust evaluation and efficient training of code LLMs. *CoRR*, abs/2504.14655, 2025. doi: 10.48550/arxiv.2504.14655.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit Bayesian inference. In *The 11th International Conference on Learning Representations*, 2022.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing systems*, volume 37, pages 50528–50652. Curran Associates, Inc., 2024a. doi: 10.48550/arXiv.2405.15793.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing systems*, volume 37, pages 115491–115522. Curran Associates, Inc., 2024b. doi: 10.48550/arXiv.2406.06484.

Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. SemRegex: A semantics-based approach for generating regular expressions from natural language specifications. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1608–1618. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1189.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *The 13th International Conference on Learning Representations*, 2025.

A Limitations, negative results, and societal impact

Our investigation isolates two core components of coding agents [Yang et al., 2024a]: post-generation selection and iterative revision via environment feedback. To ensure controlled experiments with well-defined environment interaction, we emphasize competitive coding tasks rather than multi-file editing workflows. We encourage future work to put more emphasis on fully agentic settings. While our framework defines Δ and identifies it as a key limiting factor, future work could actively proxy Δ during inference to trigger user clarification, preventing wasted compute on unresolvable ambiguity.sc

Connection to Classical Learnability and Negative Results Our probabilistic framework assumes programs and test suites to lie within a space of token strings with maximum length L. Moreover, all our definitions, such as code similarity, assume strict execution timeouts when running programs / test suites. These assumptions are required to operate in a regime where the Halting Problem [Turing, 1936] or the negative results of Gold's [1967] identifiability in the limit are ineffective. In contrast to Shapiro's [1982] algorithmic debugging framework, we do not assume access to an oracle agent that can answer queries about the target program's exact behavior. Instead, we only rely on the initial task description, which is a finite-length text. As such, we cannot guarantee convergence to the true optimal program. Instead, our paper considers a strongly Bayesian perspective, where all statements are with respect to the learned LLM prior.

Modeling Assumptions and the Learned Prior Our theoretical interpretation of iterative backprompting relies heavily on modeling the LLM as a Thompson sampler. However, LLM pretraining only promotes approximate Thompson sampling behavior, with only vacuous PAC-learning bounds [Valiant, 1984] for LLM-sized hypothesis spaces. Consequently, formal statements that build upon the assumption of exact Thompson sampling (such as Theorem 4.1) should be interpreted as qualitative or stylized characterizations of LLM behavior rather than absolute mathematical guarantees. Additionally, the underlying reward function in software engineering is deterministic, not probabilistic. Our theoretical results are purely Bayesian; they reflect the expected reward with respect to the LLM's learned prior belief, rather than an absolute ground truth.

Irreducible Regret as a Fundamental Limit Even if an LLM were a perfect Thompson sampler, Theorem 4.1 would still identify an irreducible regret (Δ). This explicitly establishes a negative result: absolute convergence to the optimal, ground-truth solution cannot be guaranteed due to the inherent ambiguity of text-only, informal task descriptions, even under ideal assumptions.

Computational Constraints Both of the investigated types of environment interaction strategies suffer from severe computational constraints. Post-generation selection heuristics necessitate executing every generated code against every generated test, resulting in a quadratic runtime execution bottleneck. Iterative backprompting avoids this runtime execution cost, but introduces a massive memory overhead, since appending execution histories linearly grows the context window.

Impact statement

This paper advances the theoretical understanding of test-driven code generation via LLMs. As a theoretical work, its societal impact will be mostly indirect. More generally, a positive societal impact of research on LLM-assisted coding is improved developer productivity and software reliability. However, there are potential negative societal impacts to consider. Highly capable coding assistants may lead to shifts in the software engineering labor market, e.g., reducing entry-level job opportunities.

B Related work

LLMs for code generation Since the introduction of Codex [Chen et al., 2021], a model fine-tuned on GitHub, code generation has become a central application of LLMs. Modern general-purpose models now routinely include substantial code data during training [Qwen Team, 2025].

Post-generation selection strategies Selection heuristics have become a popular approach for improving performance without retraining [Wang et al., 2023]. ALPHACODE [Li et al., 2022] popularized selecting generations based on pass rates on ground-truth tests and behavioral similarity under self-generated inputs. In scenarios where external tests are unavailable, heuristics like MBR-EXEC [Shi et al., 2022] and CODET [Chen et al., 2023] select solutions based on agreement groups—sets of codes that pass the same self-generated tests. Many heuristics, such as CODET, rely on "hard" agreement (counting identical outputs). Our work theoretically formalizes these as environment-aware estimators of program correctness and investigates "soft" alternatives that use fuzzy functional similarity to improve accuracy.

Iterative refinement via backprompting Beyond selection, environment feedback is used to iteratively refine code. Frameworks like REFLEXION [Shinn et al., 2023] use verbal reinforcement learning, asking LLMs to reflect on textual feedback to guide repairs. Similarly, SELFREPAIR [Olausson et al., 2024] executes a repair step based on external test failures. While these methods assume valid, discriminative tests provided, our framework addresses the more common setting where assistants rely on self-generated tests. In this setup, we frame backprompting as approximate Thompson sampling [Thompson, 1933, Russo and Roy, 2014]. This allows us to theoretically identify an irreducible regret: a term in the regret bound caused by the inherent task ambiguity.

Bayesian perspectives on in-context learning Recent theoretical work by Xie et al. [2022] and Chlon et al. [2025] demonstrates that in-context learning conducts approximate Bayesian inference. Sutter et al. [2025] argue that pre-training on sequence data with cross-entropy loss naturally yields models capable of Thompson sampling. While Menet et al. [2026b,a] explore Thompson sampling via fine-tuning of LLMs, we study the emergent Thompson sampling capabilities of off-the-shelf LLMs via in-context conditioning. This allows us to develop and apply regret bounds from Bayesian optimization to explain the limitations of backprompting strategies.

B.1 Post-generation selection strategies as estimators of code correctness

Our framework allows us to identify several well-established selection heuristics as maximizing an estimate of the probability of correctness $p(\mathcal{N}_c^\infty)$, see the developments in Section 3. Table 1 provides a summary of the code selection literature based on our formalism. In the experimental Section 5, we focus on boolean test outputs $\mathbb{O}$, i.e., the canonical case of test-driven development.

MBR-EXEC Under the minimum Bayes risk decoding strategy [Shi et al., 2022], a set of codes $c_1, \ldots, c_n \in \mathcal{V}^{\leq L}$ and test suites $t_1, \ldots, t_m \in \mathcal{V}^{\leq L}$ are sampled according to the LLM posteriors $p(c)$ and $p(t)$ given a task description $d \in \mathcal{V}^{\leq L}$. Then, a code $\hat{c} = \arg\min_{c_j \in c_1, \ldots, c_n} \sum_{i \neq j} \ell(c_i, c_j)$ is selected to minimize an empirical estimate of the Bayes risk $\mathbb{E}_{c' \sim p}[\ell(c, c')]$ with risk function ℓ. The risk ℓ is chosen either as $-\widehat{\mathrm{sim}}^{\infty}_{p,e;m}(c, c')$ or $-\widehat{\mathrm{sim}}^{1}_{p,e;m}(c, c')$, resulting in MBR-EXEC HARD and MBR-EXEC SOFT, respectively.

ALPHACODE and FUNCODER Li et al. [2022] and Chen et al. [2024a] do not sample boolean-valued test cases from the language model, but rather execution scripts that invoke the code at arbitrary inputs. In our formalism, they apply generalized test suites t that return a vector of non-boolean outputs $\mathbb{A}^{|t|}$, which is then compared to estimate functional similarity. Whereas ALPHACODE relies on functional equivalence by estimating $\widehat{\mathrm{sim}}^{\infty}_{p,e;m}(c, c')$, FUNCODER relaxes this to functional similarity $\widehat{\mathrm{sim}}^{1}_{p,e;m}(c, c')$.

MAXPASS Lahiri et al. [2022], Rozière et al. [2022], Le et al. [2022], Chen et al. [2024b] consider Monte Carlo estimators of the dual expression of code correctness $\mathbb{P}_{t \sim p}[R(c, t \mid e) = 1]$, albeit only in the setting of single-test test-suites, where the hard and soft estimators coincide. Theoretically, if a model were perfectly code-test calibrated (Definition 3.11), code-based probability of correctness

(MBR-EXEC) and test-based probability of correctness (MAXPASS) would yield the exact same result in expectation. However, as we demonstrate in Section 5.2, MAXPASS consistently outperforms MBR-EXEC, indicating that current LLMs are miscalibrated. Since Monte Carlo estimators of probability of maximality have high variance [Menet et al., 2025], one may instead consider a soft measure of code correctness such as $\hat{\mathbb{E}}_{t_{1:m}}[R(c, t \mid e)]$.

CODET Chen et al. [2023] multiply the objective in the MBR-EXEC HARD strategy with the number of passed test cases. According to our formalism, they take the geometric mean of two dual estimators of self-consistency: $\hat{p}_{n,m}(\mathcal{N}_c^{\infty})$ and $\hat{\mathbb{E}}_{t_{1:m}}[R(c, t|e)]$. With their method, they argue for increased robustness. Based on the insights from Section 3.2, we extend CODET with a soft variant (CODET SOFT) that adopts the smooth estimator $\hat{p}_{n,m}(\mathcal{N}_c^{1})$. As demonstrated in the experimental section, our smooth variant of CODET performs consistently better than the original hard variant.

C Additional experiments

C.1 Improvements of post-generation selection as a function of the number of rounds

We complement Table 2 with Figure S1, which details the accuracy improvements achieved after generation through the considered selection heuristics. We plot the improvement as a function of the number of rounds of code and test suite generations. In several settings, the accuracy improvements do not yet saturate after 10 rounds, but the compute cost of post-generation selection starts to dominate the generation cost (using a Qwen3-235B-A22B-Instruct, even for massive parallelism with 64 AMD EPYC 7763 CPU cores for selection and only a single node with 8 NVIDIA A100 80GB GPUs for generation). See Section C.6 for runtime measurements. This runtime bottleneck stems from the quadratic scaling of post-generation selection, as every code must be executed on every test suite.

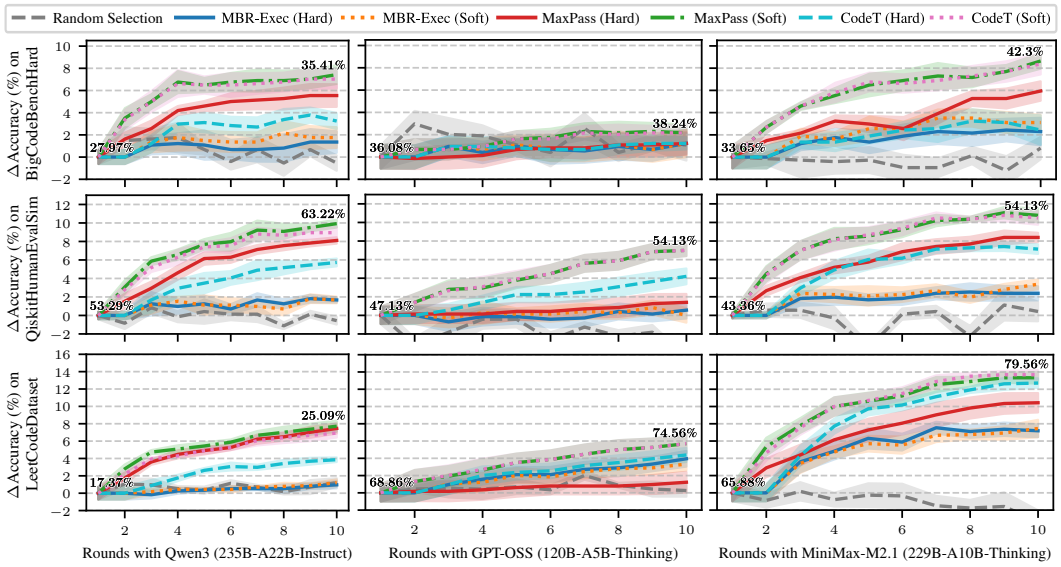


Figure S1: Pass@1 (%) improvements of common post-generation selection heuristics (see Table 1 and Table 2) across BigCodeBenchHard, QiskitHumanEvalSim, and LeetCodeDataset. Qwen3-235B-A22B-Instruct-2507, GPT-OSS-120B, or MiniMax-M2.1 are used to independently generate code and tests across multiple rounds. Results are reported as mean (± standard error) over 5 seeds.

C.2 Approximate in-context Thompson sampling with GPT-OSS-120B and MiniMax-M2.1

Figure S3 reports additional results on GPT-OSS-120B and MiniMax-M2.1. We find that Qwen3-235B-A22B-Instruct is a more effective approximate Thompson sampler. Indeed, given ORACLE-TEST FEEDBACK (purple dotted line), Qwen3-235B-A22B-Instruct yields significantly stronger accuracy improvements than MiniMax-M2.1, whose gains saturate after just one round. Moreover, both GPT-OSS-120B and MiniMax-M2.1 generate test suites that are much worse. In fact, given SELF-TEST FEEDBACK, the SELF-TEST ACCURACY (cyan dashed line) is much lower than the ORACLE-TEST ACCURACY (black solid line). As a result, these two models receive highly noisy environment feedback and thus no accuracy gains over the zero-shot performance can be achieved.

Note that, whereas Figure 3 reports on *summary concatenation* and *test-first* factorization ($p(c,t) = p(c|t) \cdot p(t)$, see Section C.4), Figure S3 reports our initial experiments using *insight reformulation* and *independent* factorization ($p(c,t) = p(c) \cdot p(t)$). This avoids weeks of compute effort. Note that while *summary concatenation* and *test-first* factorization result in superior performance, the optimization trajectory behaves rather similarly, see also Figure S2.

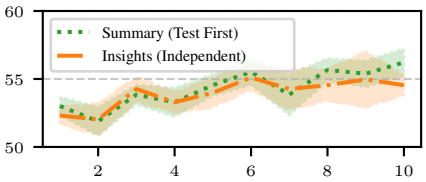
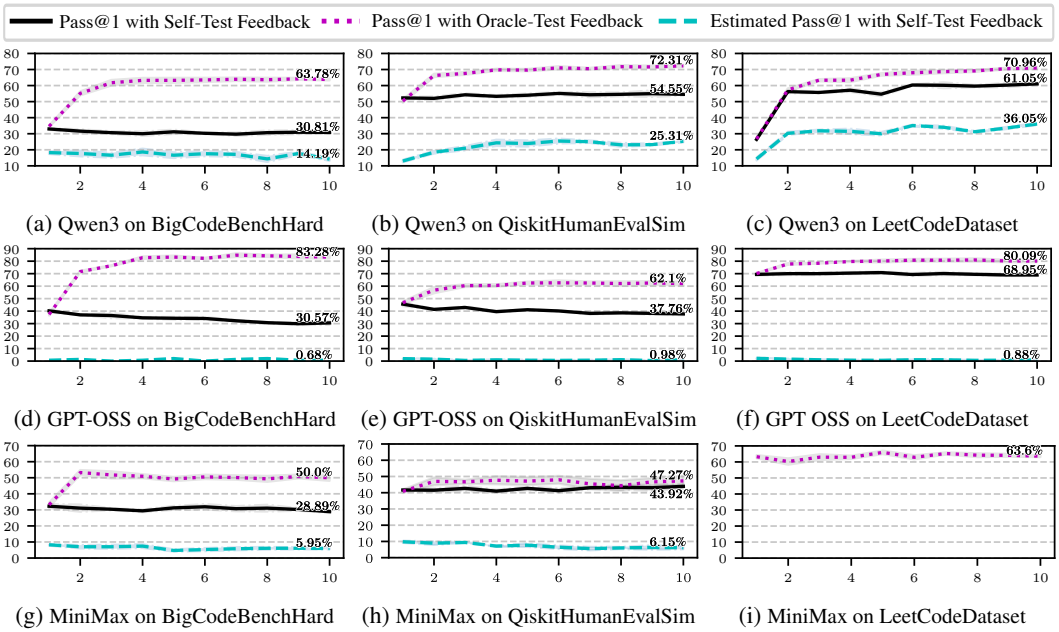


Figure S2: Pass@1 of in-context Thompson sampling via backprompting of self-test feedback using Qwen3-235B-A22B-Instruct-2507 on QiskitHumanEvalSim.



(a) Qwen3 on BigCodeBenchHard (b) Qwen3 on QiskitHumanEvalSim (c) Qwen3 on LeetCodeDataset

(d) GPT-OSS on BigCodeBenchHard (e) GPT-OSS on QiskitHumanEvalSim (f) GPT OSS on LeetCodeDataset

(g) MiniMax on BigCodeBenchHard (h) MiniMax on QiskitHumanEvalSim (i) MiniMax on LeetCodeDataset

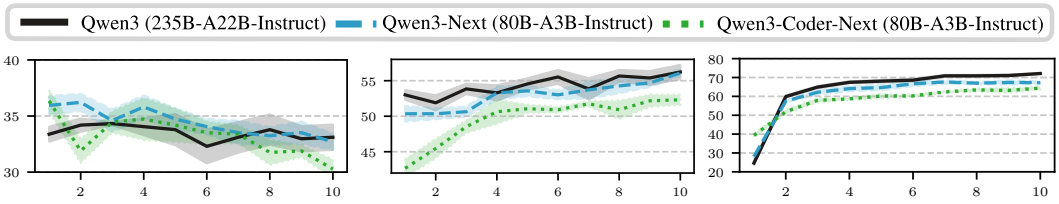
Figure S3: Pass@1 (%) of approximate in-context Thompson sampling via backprompting with Qwen3-235B-A22B-Instruct-2507, GPT-OSS-120B, and MiniMax-M2.1 across BigCodeBenchHard, QiskitHumanEvalSim, and LeetCodeDataset. In the language of Theorem 4.1, the black solid line corresponds to the true reward r, and the cyan dashed line to r_{obs}. The purple dotted line corresponds to the case of known r_{hid}, which according to Corollary 4.2 permits global convergence to x^*. These experiments were conducted with *insight reformulation* and *independent* factorization ($p(c,t) = p(c) \cdot p(t)$). Results are reported as mean ($\pm$ standard error) over 5 random seeds.

C.3 Backprompting with hybrid Qwen3-Next, Qwen3-Coder-Next, and Qwen3.5-397B-A17B

	BigCode	Qiskit	LeetCode
Qwen3-235B-A22B	33.38 -> 33.11	53.0 -> 56.22	24.65 -> 72.11
Qwen3-Next	35.95 -> 32.7	50.35 -> 56.08	28.07 -> 67.28
Qwen3-Coder-Next	36.08 -> 30.27	43.5 -> 52.31	39.3 -> 64.39
Qwen3.5-397B-A17B	37.97 -> 37.3	55.66 -> 59.3	66.96 -> 73.98

Table S1: Initial and final pass@1 (%) of members of the Qwen3(.5) model family during 10 rounds of backprompting self-test feedback. We report the mean across 5 seeds. For detailed trajectories, see Figure S4.

In addition, we evaluate Qwen3-Next, Qwen3-Coder-Next, and Qwen3.5-397B-A17B [Qwen Team, 2026]. The Next variants are 80B parameter models of which only 3B are active, and all three are built on a hybrid DeltaNet-Transformer architecture [Vaswani et al., 2017, Yang et al., 2024b]. As shown in Table S1 and Figure S4, these models achieve results competitive with the Transformer-based Qwen3-235B-A22B-Instruct under backprompting with SELF-TEST FEEDBACK.



(a) BigCodeBenchHard (b) QiskitHumanEvalSim (c) LeetCodeDataset

Figure S4: Pass@1 (%) of backprompting concatenated summaries of self-test feedback using the Qwen3 model family. Results are reported as mean ($\pm$ standard error) over 5 seeds.

Moreover, hybrid models integrate well with *summary-concatenation* history compression, since the cost of the state-space-model part does not grow with the rounds. Overall, these results indicate that model size and architectural design are likely not what hold GPT-OSS and MiniMax-M2.1 back.

C.4 Ablating model factorization and feedback compression / compactification

In Table S2, we investigate two degrees of freedom in the implementation of in-context Thompson sampling. To start, we consider different factorizations of $p(c, t)$: *independent*, *code first*, and *test first*. First generating the test suite, and then a matching implementation conditioned on the suite works best. Next, we consider different LLM-based compressors of test reports $U(c, t \mid e)$. Compression is necessary, since concatenation of the test reports themselves quickly exceeds the available memory. We ablate two compressors: iterative reformulation of past insights in light of a new test report, or concatenation of summaries of each test report. A detailed description of each is provided in Appendix D.4. As our results demonstrate, *summary concatenation* outperforms *insight reformulation*, although, at the cost of additional memory and compute. Unless otherwise mentioned, we always report on *summary concatenation* with the factorization $p(c \mid t) \cdot p(t)$.

Table S2: Pass@1 (%) after 10 rounds of in-context Thompson sampling (with improvements during generation in parenthesis) for different factorization of $p(c, t)$ and different test report compression methods. Results reported as mean ($\pm$ standard error) over 5 random seeds using Qwen3-235B-A22B-Instruct. For each column, the best configuration is emphasized and the second best is underlined.

Compression	Factorization	BigCodeBenchHard	QiskitHumanEvalSim	LeetCodeDataset
Insights Reformulation	$p(c) \cdot p(t)$	$30.81^{\pm 0.28}$ ($-2.16^{\pm 0.32}$)	$54.55^{\pm 0.33}$ ($2.24^{\pm 0.33}$)	$61.05^{\pm 0.32}$ ($34.56^{\pm 0.48}$)
Insights Reformulation	$p(t \mid c) \cdot p(c)$	$31.89^{\pm 0.45}$ ($-0.68^{\pm 0.51}$)	$54.27^{\pm 0.39}$ ($2.94^{\pm 0.90}$)	$58.60^{\pm 0.47}$ ($34.65^{\pm 0.72}$)
Insights Reformulation	$p(c \mid t) \cdot p(t)$	$\mathbf{34.05}^{\pm 0.31}$ ($-0.81^{\pm 0.46}$)	$54.97^{\pm 0.19}$ ($3.08^{\pm 0.40}$)	$60.61^{\pm 0.48}$ ($36.40^{\pm 0.30}$)
Summary Concatenation	$p(c) \cdot p(t)$	$32.84^{\pm 0.45}$ ($-0.27^{\pm 0.49}$)	$55.52^{\pm 0.36}$ ($3.92^{\pm 0.46}$)	$\underline{71.14}^{\pm 0.24}$ ($42.02^{\pm 0.32}$)
Summary Concatenation	$p(t \mid c) \cdot p(c)$	$32.03^{\pm 0.44}$ ($-0.41^{\pm 0.58}$)	$\underline{56.08}^{\pm 0.55}$ ($5.45^{\pm 0.92}$)	$68.42^{\pm 0.31}$ ($44.74^{\pm 1.00}$)
Summary Concatenation	$p(c \mid t) \cdot p(t)$	$\underline{33.11}^{\pm 0.51}$ ($-0.27^{\pm 0.75}$)	$\mathbf{56.22}^{\pm 0.45}$ ($3.22^{\pm 0.70}$)	$\mathbf{72.11}^{\pm 0.08}$ ($47.46^{\pm 0.39}$)

C.5 Combining backprompting with post-generation selection

Consider Table S3, which reports on the MAXPASS selection heuristic applied on top of in-context Thompson sampling. The 10 rounds of code and test suite generations are integrated into post-generation selection as if independently generated. The pass@1 of Thompson sampling still benefits from post-generation selection, but the improvements are less pronounced than with independent generation.

Table S3: Pass@1 (%) comparison of backprompting and post-gen selection. Each solution produced 10 code-test pairs. We use a Qwen3-235B-A22B-Instruct-2507 and report mean ($\pm$ standard error) over 5 seeds.

Selection	Backprompting	BigCode	Qiskit	LeetCode
RANDOM	No	$27.43^{\pm 0.31}$	$52.73^{\pm 0.19}$	$18.60^{\pm 0.22}$
LAST	SELF-TEST FB	$\mathbf{33.11}^{\pm 0.51}$	$\mathbf{56.22}^{\pm 0.45}$	$\mathbf{72.11}^{\pm 0.08}$
MP SOFT	No	$35.41^{\pm 0.38}$	$\underline{\mathbf{63.22}}^{\pm 0.19}$	$25.09^{\pm 0.29}$
MP SOFT	SELF-TEST FB	$\underline{\mathbf{36.22}}^{\pm 0.47}$	$60.84^{\pm 0.36}$	$\underline{\mathbf{73.42}}^{\pm 0.17}$

C.6 Computational cost of environment interaction and in-context Thompson sampling

While environment-aware selection and in-context Thompson sampling yield large accuracy improvements, they add substantial compute.

Post-generation selection: Heuristics like MBR-EXEC, MAXPASS, and CODET require cross-executing n generated implementations against m generated test suites, resulting in a quadratic scaling of runtime execution cost, $\mathcal{O}(n \times m)$. In practice, this execution bottleneck becomes quickly apparent: e.g., on QiskitHumanEvalSimX with Qwen3-235B-A22B-Instruct-2507, using a single node with an AMD EPYC 7763 64-Core CPU and 8 NVIDIA A100 (80GB) GPUs, running all implementations on all test suites across the 143 dataset entries for 10 rounds with 5 seeds takes 21 hours and 32 minutes, dominating the 12 hours and 42 minutes required for LLM based code-test generation. This motivates using soft heuristics with lower signal-to-noise ratio (see Theorem 3.10), since they require fewer code/test generations to provide a meaningful selection signal.

Iterative backprompting: Conversely, in-context Thompson sampling avoids the $\mathcal{O}(n \times m)$ runtime bottleneck by only evaluating the current code and test suite, but introduces additional demand on the

LLM-based code-test generation. Given the more performant *summary concatenation* compressor, see Appendix C.4, extending the prompt with historical feedback linearly increases its length with every round. Given the quadratic complexity of attention in standard Transformers, prolonged environment interaction quickly hits a compute and memory ceiling. For instance, across our 10-round Qwen3-235B-A22B-Instruct-2507 experiments, the model processed 119 million total tokens, of which only 17 million were newly generated. This context overhead strongly suggests that model architectures with linear complexity, such as State-Space Models (SSMs) or Linear Transformers, will be critical for scaling iterative coding agents capable of dynamically retaining long execution histories.

C.7 Quality and discriminative power of generated tests

To assess the composition and discriminative power of the self-generated test suites, we empirically analyzed all tests generated by our primary model (Qwen3-235B-A22B-Instruct) over the 10 rounds of backprompting across all datasets. Out of $261,566$ individual test cases, only 0.06% (166 tests) contain a trivial `assert True`. An additional 7.20% ($18,828$ tests) contain no explicit assert statements (relying instead on execution crashes to fail). The vast majority of tests contain programmatic assertions, demonstrating that the model assigns low probability mass to degenerate tests.

We further analyzed the distribution of the average test failure rate across all test cases (with the average computed across all generated programs). The overall mean failure probability is 36.83%. As depicted in Figure S5, the resulting distribution is notably bimodal.

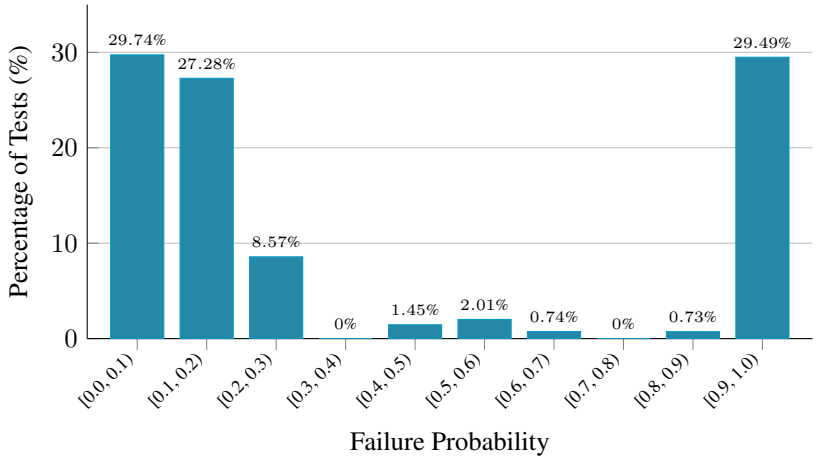


Figure S5: Distribution of average test failure rate (computed across all generated programs) for $261,566$ test cases generated by Qwen3-235B-A22B-Instruct on BigCode, Qiskit, and LeetCode.

This bimodal distribution indicates that the generated tests cover a broad spectrum of difficulty. Rather than providing a uniform evaluation signal, the tests tend to partition into two primary groups: permissive tests that the majority of generated programs pass (serving as basic sanity checks), and stringent tests that the majority of programs fail (e.g., evaluating edge cases).

C.8 Empirical signatures of in-context Thompson sampling

To justify modeling the LLM as an approximate Thompson sampler, we provide indirect empirical evidence of its sampling distribution. First, if the LLM "lazily debugged" the code and test suite to match, we would expect the self-generated tests to drift and become less faithful, e.g., by deleting difficult-to-satisfy test cases. However, as shown in Table S4, the test suite length and precision (positive predictive value) remain remarkably stable across 10 rounds of backprompting.

Next, if the model genuinely executed approximate Thompson sampling, its posterior belief would contract as environment feedback reduces uncertainty, yet remain stochastic due to the irreducible regret. In contrast, simple iterative debugging would be expected to collapse to a single deterministic solution. Table S5 tracks the collision probability of consecutively generated codes (the probability they belong to the same functional equivalence class, estimating $\mathbb{E}_{c^*_{obs,1}, c^*_{obs,2}}[\mathrm{sim}^{\infty}_{p,e}(c^*_{obs,1}, c^*_{obs,2})]$). The model's belief contracts significantly over time, yet always remains stochastic.

Table S4: Average test suite length and precision (positive predictive value) across 10 rounds of in-context Thompson sampling using the Qwen3-235B-A22B-Instruct model.

Dataset	Round	Average Test Suite Length	Precision
BigCodeBenchHard	1	9.86	41.4%
BigCodeBenchHard	5	10.50	41.9%
BigCodeBenchHard	10	10.73	37.8%
QiskitHumanEvalSim	1	7.17	73.1%
QiskitHumanEvalSim	5	7.69	74.2%
QiskitHumanEvalSim	10	7.80	71.2%
LeetCodeDataset	1	11.59	74.1%
LeetCodeDataset	5	11.33	78.9%
LeetCodeDataset	10	11.45	78.6%

Table S5: Collision probability per round of in-context Thompson sampling using Qwen3-235B-A22B-Instruct (averaged across a sliding window of size 3 to account for non-stationarity).

Round	BigCodeBenchHard	QiskitHumanEvalSim	LeetCodeDataset
4	56.40%	77.48%	75.29%
5	62.12%	80.70%	81.37%
6	66.08%	82.84%	83.13%
7	68.29%	84.38%	85.56%
8	71.71%	87.65%	87.02%
9	73.65%	87.69%	88.36%
10	72.93%	88.34%	88.89%

C.9 Rank correlation and signal-to-noise ratio of selection heuristics

To explicitly connect the signal-to-noise ratio (SNR) dominance established in Theorem 3.10 to the empirical pass@1 gains, we analyze estimator variance and rank correlation with the oracle correctness. As Table S6 demonstrates, the hard estimators (MAXPASS HARD) actually match or slightly outperform the soft estimators (MAXPASS SOFT) in terms of rank correlation (Spearman's ρ and Kendall's τ). However, exactly as predicted by our theory, the soft estimators exhibit a vastly superior empirical signal-to-noise ratio. This confirms that the improved pass@1 of soft estimators is genuinely driven by their smoothing bias overcoming the high-variance bottleneck.

Table S6: Rank correlation and SNR analysis at 5 and 10 rounds of independent generation using Qwen3-235B-A22B-Instruct. This is the primary setting investigated for post-generation selection.

Dataset	Strategy	# Test Suites	Spearman's ρ	Kendall's τ	SNR (bootstrapped)
BigCodeBenchHard	MAXPASS HARD	5	**0.500**	**0.494**	3.01
	MAXPASS SOFT	5	0.495	0.481	**151.40**
LeetCodeDataset	MAXPASS HARD	5	**0.864**	**0.861**	5.11
	MAXPASS SOFT	5	0.804	0.793	**553.83**
QiskitHumanEvalSim	MAXPASS HARD	5	**0.787**	**0.783**	2.79
	MAXPASS SOFT	5	0.692	0.680	**53.53**
BigCodeBenchHard	MAXPASS HARD	10	**0.445**	**0.438**	4.69
	MAXPASS SOFT	10	0.420	0.399	**207.08**
LeetCodeDataset	MAXPASS HARD	10	**0.815**	**0.811**	7.12
	MAXPASS SOFT	10	0.757	0.741	**493.67**
QiskitHumanEvalSim	MAXPASS HARD	10	**0.731**	**0.726**	3.96
	MAXPASS SOFT	10	0.703	0.686	**73.35**

C.10 Comparison with B4 baseline

To provide a comparison with recent Bayesian methods for post-generation selection, we benchmarked B4 [Chen et al., 2024b], which computes the MAP probability of an entire consensus set being the true correct solution. Note that in B4 a consensus set is defined via a Monte Carlo estimate of $\mathcal{N}_c^{\infty}$, tying into our theory developed in Section 3. Table S7 shows that MAXPASS SOFT consistently matches or outperforms B4, further illustrating the strength of our considered smooth estimators.

Table S7: Pass@1 (%) performance comparing B4 against MAXPASS SOFT as post-generation selection mechanisms after 10 rounds of independent generation using Qwen3-235B-A22B-Instruct.

Dataset	B4	MAXPASS SOFT
BigCodeBenchHard	33.24% $\pm$ 0.48%	**35.41%** $\pm$ 0.75%
LeetCodeDataset	**25.18%** $\pm$ 0.42%	25.09% $\pm$ 0.57%
QiskitHumanEvalSim	60.56% $\pm$ 0.32%	**63.22%** $\pm$ 0.38%

D Experimental details

To ensure reproducibility, we complement Section 5 with a more detailed description of the experimental setup. All experiments were conducted on compute nodes with an AMD EPYC 7763 64-Core CPU, 2TB RAM, and eight NVIDIA A100-SXM4 GPUs (80GB), running on Red Hat Enterprise Linux 9.4 with CUDA 12.4. Our code includes configuration files for each experiment and environment files specifying the necessary Python packages and their versions, and is publicly available at `https://github.com/IBM/theory-of-coding-agents` under a permissive license.

D.1 Datasets

D.1.1 BigCodeBenchHard

The partition of the 148/1140 most challenging tasks in BigCodeBench [Zhuo et al., 2025]. BigCodeBench, released under the Apache 2.0 License, is designed for function-level code generation, adopting the format of HumanEval [Chen et al., 2021]. The tasks in BigCodeBench require multiple external libraries across 7 domains including networking, data analysis, and visualization. The benchmark tests compositional reasoning and the ability to implement complex instructions that span 139 libraries with an average of 2.8 libraries per task.

D.1.2 QiskitHumanEvalSim

A novel subset of QiskitHumanEval [Vishwakarma et al., 2024], that we introduce. It consists of the subset of the 143/151 questions (excluding task ids 43, 97, 98, 122, 129, 133, 134, 146 of which 6 are classified as basic, 2 as intermediate, and none as difficult) that can be solved without access to a real-world quantum computer (i.e., simulated), democratizing the dataset. Due to the small fraction of training data being dedicated to Qiskit during typical LLM training, and since Qiskit's API has undergone multiple backward-incompatible changes, QiskitHumanEvalSim is a great testbed for runtime environment interaction. Note that QiskitHumanEval adopts the Apache 2.0 License.

D.1.3 LeetCodeDataset v0.3.1

228 diverse puzzle-style questions from LeetCode released after 2024-07-01 and collected by Xia et al. [2025]. LeetCode typically includes example input-output pairs in natural language, significantly reducing the irreducible regret Δ identified in Corollary 4.2 and, as we show, enabling more effective in-context Thompson sampling. The dataset was released under the MIT license.

D.1.4 QISKITHUMANEVALSIMX

QISKITHUMANEVALSIMX is a novel benchmark that we introduce in order to corroborate our theory, in particular the insight that effective in-context Thompson sampling requires a task description $d \in \mathcal{V}^{\leq L}$ with small uncertainty on the latent algorithm $a \in \mathcal{A}$ (see Theorem 4.1 and its Corollary 4.2). We improve the specification of the algorithm in d by describing its target behavior in more detail, including input-output examples. More precisely, to create QISKITHUMANEVALSIMX from QiskitHumanEvalSim, we query Google Gemini 3 Pro (January 10, 2026) using the following prompt:

````
Consider the following dataset entry:
```
{"task_id": "qiskitHumanEval/35", "prompt": "from
qiskit.circuit.library [...]
```
Improve the docstring in the prompt by appending a natural
language description of the test without leaking the canonical
solution. The natural language description must be detailed,
but must not contain code. If the test contains an
input-output example, provide it. Don't explicitly mention the
presence of tests. Don't change anything else. Return the
improved dataset entry in the jsonl format.
````

In case the instructions were not followed or invalid symbols were returned, the query was repeated. The resulting dataset was verified by running all canonical solutions against the oracle test suite. Moreover, to ensure that the prompt constraints were followed and no oracle-test leakage occurred, we manually inspected the generated dataset entries to verify that the canonical solutions and oracle tests were not directly revealed.

D.1.5 SWE-bench Verified

A human-validated subset of the SWE-bench evaluation framework [Jimenez et al., 2024], consisting of real-world software engineering issues collected from open-source GitHub repositories. The benchmark transitions our evaluation from isolated function-level synthesis to a fully agentic coding paradigm, requiring language model agents to autonomously navigate complex multi-file codebases, execute bash tools, and implement code modifications. The dataset's irreducible regret Δ (see Corollary 4.2) is quite small, making it an excellent testing ground for multi-subaction contextual Thompson sampling. The dataset is released under the MIT license.

D.2 Models

We consider five frontier open-weight LLMs and two smaller hybrid LLMs, with an emphasis on qualitative differences such as reasoning vs instruct and general-purpose vs coding. We sample directly from the model with temperature 1.0 and top-p set to 0.95. After exceeding $32\,768$ output tokens, generation is stopped. In case invalid output is generated (e.g., if no code or observations can be extracted) we repeat up to 9 more times the same query to the LLM. The considered models are as follows:

Qwen3-235B-A22B-Instruct-2507 is the largest publicly available instruction-tuned model in the general-purpose Qwen3 model family [Qwen Team, 2025]. Adopting the mixture of experts (MoE) architecture [Shazeer et al., 2017], only 22 out of the 235 billion parameters are active at inference. The model is **not trained for chain-of-thought** [Wei et al., 2022]. It was released under Apache 2.0.

GPT-OSS-120B is the largest publicly available model released by OpenAI [OpenAI, 2025]. It is a general-purpose reasoning model trained for long **chain-of-thoughts**. Tallying a total of 120 billion parameters, but with only 5B active, it is optimized for high throughput. It follows Apache 2.0.

MiniMax-M2.1 and MiniMax-M2.7 are state-of-the-art **coding** models released on the 23rd of December 2025 and 18th of March 2026, building on the MiniMax-M1 architecture [MiniMax, 2025]. They are MoE with 229 billion total parameters, of which only 10B are active. They are trained for and **utilize chain-of-thought** for reasoning. The models are released under a modified MIT license.

Qwen3-Next and Qwen3-Coder-Next are MoE models introduced by Qwen Team [2026]. Of a total of 80B parameters only 3B are active. Building on a hybrid DeltaNet-Transformer architecture [Vaswani et al., 2017, Yang et al., 2024b], they are particularly efficient when handling long contexts. We consider the instruct variants **without chain-of-thought**. They follow Apache 2.0.

Qwen3.5-397B-A17B is the largest hybrid Transformer-SSM model in our evaluation suite, introduced by Qwen Team in 2026. Tallying a total of 397 billion parameters with 17B active at inference, it expands on the hybrid architecture to efficiently manage large historical contexts. Crucially, and in contrast to its smaller siblings considered here, the model is fully trained for and **utilizes chain-of-thought** for complex algorithmic reasoning. It is released under the Apache 2.0 license.

D.3 Implementation and test suite generation prompts

In the following, we list the system prompts used for implementation and test suite generation, adopting the factorization $p(c, t) = p(c \mid t)p(t)$. The prompts for the alternative factorizations are adjusted accordingly and listed in the code base.

Test suite generation

```
You are an expert Python programmer.
```

```
Your task is to write pytests for an implementation that obeys
the provided task description. You are provided observations
from previous rounds of comparing potential implementations
against potential test suites.

**1. Analyze and Plan**
 - Carefully reflect on the provided observations.
 - Your new test suite must address these observations while
still covering the program's main functionality and common
edge cases.

**2. Write the test suite**
 - Include all necessary imports for external libraries. The
candidate implementation is available without imports.
 - Each pytest must be represented by a separate function
starting with `test_`. In particular, do not define test
classes.
 - Add a brief docstring to each pytest function explaining
its purpose.
 - Do not number the pytests.
 - Provide all of the pytests in a single Python code block
marked with triple backticks (```).
```

Implementation generation

```
You are an expert Python programmer.
Your task is to write an implementation that obeys the
provided task description. You are provided observations from
previous rounds of comparing potential implementations against
potential test suites, as well as a novel test suite.

**1. Analyze and Plan**
 - Carefully reflect on the provided observations.
 - Review the provided test suite.
 - Your new implementation must address the observations, pass
all test cases, and adhere to the original task description.

**2. Write the implementation**
 - Include all necessary imports for external libraries.
 - Do not include a docstring.
 - Write the program in a Python code block marked with triple
backticks (```).
```

D.4 Feedback compression

We consider two strategies for compressing the test reports. With *insight reformulation*, we not only compress each report into actionable insights, but also reformulate old insights to avoid insight duplication and to ensure the context size only increases if new information is acquired. In contrast, with *summary concatenation*, we simply ask the LLM to compress the test report of each round into a summary, which are then concatenated in the prompt. This ensures no information is accidentally lost, but results in a context that grows linearly with the number of rounds of in-context Thompson sampling.

Insight reformulation

```
You are an expert Python programmer.
You are provided a task description, a candidate program
implementation, a candidate test suite with feedback on the
implementation, and additional past observations.
Your task is to first analyze the test feedback and then state
an exhaustive list of insights that acts as a compressed
history of all observations.
```

```
**1. Analyze Feedback**
 - Carefully review the program, tests, and feedback. This is
your new data.
 - Identify errors in the program implementation, as well as
weaknesses in the tests themselves.
 - Summarize all your findings. Be precise and do not
speculate.

**2. State Insights**
 - Restate all insights from past observations as well as
additional insights from the feedback analysis.
 - Mark each insight with `<observation> </observation>` tags.
 - Ensure each insight is self-contained, i.e., informative
without considering the provided program and test suite.
```

Summary concatenation

```
You are an expert Python programmer.
You are provided a task description, a candidate program
implementation, and a candidate test suite with feedback on
the implementation. Your task is to first analyze the test
feedback and then summarize the implementation, tests, and
feedback.

**1. Analyze Feedback**
 - Carefully review the program, tests, and feedback. This is
your data.
 - Identify errors in the program implementation, as well as
weaknesses in the tests themselves.

**2. State Summary**
 - Summarize the provided implementation, test suite, and all
your findings.
 - Mark the summary with `<observation> </observation>` tags.
```

E Proofs

E.1 Discussion of assumptions

Across this work, several assumptions are adopted. Here, we reassure the reader of their validity.

Non-empty test suites In Definition 3.1, we tacitly assume all test suites from the test manifold to contain at least one test. This is without loss of generality, since if no test can be extracted from the language model output, we just consider the test suite to consist of a single test which always fails.

Almost surely unique correct behavior In Proposition 3.12, we assume that the true test suite is such that only one equivalence class of codes passes it. This is a desired property and necessary for program well-specification, and can thus be assumed true for the underlying generative process of Figure 2. However, it may not hold for LLM generated tests. As such, Proposition 3.12 should be understood as holding for the true joint code-test manifold, not for the LLM approximation. Note that Proposition 3.12 stands by itself and does not affect any of our other results.

LLM training approximately learns in-context Thompson sampling Since pre-training teaches the LLM to approximate the Thompson sampling policy, the regret bound of Theorem 4.1 directly maps to in-context backprompting of execution feedback. We remark, however, that post-training reshapes the learned code-test manifold and may result in behavior that deviates from the theory. Consequently, we suggest future work to investigate and contrast the in-context learning capabilities and black-box optimization capabilities of pre-trained models with those of post-trained models.

Subgaussian rewards and finite LLM response length Theorem 4.1 assumes subgaussian r_{obs} and finite $|\mathrm{dom}(r)|$. The former is trivially satisfied since test suite pass-rates are within $[0, 1]$ and since all bounded random variables are subgaussian. The latter holds as, due to memory limits in the Transformer architecture Vaswani et al. [2017], the length of LLM output responses is always limited.

Strong assumptions for predicting the reward curves At the end of Section 4.2, we estimate the shape of empirical reward curves under backprompting. We stress that this is not a formal statement and the predicted curves require strong assumptions, in particular, tight regret bounds, a saturated cumulative uncertainty $\Gamma_T \approx \Gamma$, and an accurate truncated Taylor expansion $\sqrt{T}\text{-}\sqrt{T\text{-}1} \approx T^{\text{-}1/2}/2$.

E.2 Proof sketches

To improve readability, we provide brief sketches of the intuition behind our main theoretical results before presenting all formal proofs.

Proof sketch of Proposition 3.3 (Properties of s-similarity). To prove that s-similarity is a valid kernel, we first show it holds for $s = 1$ by representing the test outputs as one-hot vectors; the proportion of passed tests is then just the expectation of their inner products. Because the class of positive semi-definite kernels is closed under point-wise multiplication, multiplying the $s = 1$ kernel by itself s times ensures it remains a valid kernel for any $s \in \mathbb{N}$. Finally, the limit as $s \to \infty$ drives any fractional similarity strictly to 0, leaving only perfect matches (value 1), which aligns exactly with the definition of functional equivalence.

Proof sketch of Proposition 3.5 (Functional equivalence classes). Strict functional equivalence ($\mathrm{sim}^{\infty}_{p,e}(c_1, c_2) = 1$) requires two implementations c_1, c_2 to yield identical outputs on all test cases of non-zero measure. Because equality is an equivalence relation (it is reflexive, symmetric, and transitive), the $\mathrm{sim}^{\infty}_{p,e}$ kernel inherits these properties. Consequently, it partitions the space of all possible programs $\mathcal{V}^{\leq L}$ into disjoint equivalence classes.

Proof sketch of Theorem 3.8 (Measure smoothing inductive bias). Because s-similarity is a positive definite kernel, we can embed the implementations into a Hilbert space using a feature map. The probability of a fuzzy neighborhood can then be expressed as the inner product between the code's feature representation and the mean embedding of the distribution. By applying the Cauchy-Schwarz inequality, the difference in probability mass between two highly similar codes is bounded by their geometric distance in the feature space, which directly links to the s-similarity.

Proof sketch of Theorem 3.9 (Consistency and bias). For strong consistency, we invoke Kolmogorov's strong law of large numbers on the bounded, independent test suite evaluations, followed by the continuous mapping theorem to handle the exponent s. For bias, we rely on the linearity of expectation to show the estimator is perfectly unbiased for $s = 1$. However, because $g(x) = x^s$ is strictly convex for $s > 1$, Jensen's inequality guarantees the estimator is biased upwards for all $s > 1$.

Proof sketch of Theorem 3.10 (SNR dominance). We compare the variances of the two estimators. The soft estimator ($s = 1$) is an average of independent bounded variables, so its variance shrinks inversely with the number of test suites m. In contrast, the hard estimator ($s \to \infty$) acts as a strict boolean AND across all sampled test suites, meaning its probability of success decays exponentially with m. Taking the ratio of their signal-to-noise ratios and applying a geometric series bound yields the m^2 dominance factor.

Proof sketch of Proposition 3.12 (Additivity of probability of correctness). Under the assumption that there is almost surely only one functionally correct equivalence class, the success events for distinct classes are mutually exclusive. Therefore, the probability of the union of their success events (pass@k) is simply the sum of their individual success probabilities (pass@1). Maximizing this sum over k selections trivially reduces to greedily picking the k classes with the highest pass@1.

Proof sketch of Theorem 4.1 (Bayesian regret bound). We first decompose the reward into an observable component (execution results) and an unobservable component (task relevancy). By taking conditional expectations, we map the regret on the total reward function to a regret purely on the observable component. We then apply standard subgaussian tail bounds and union bounds (via Lemma E.1) to limit the expected maximum deviation. Finally, applying Cauchy-Schwarz over the T steps bounds the cumulative regret by the sum of the squared subgaussian parameters.

Proof sketch of Lemma E.1 relies on subgaussian tail integration and union bounds to limit the expected maximum of a set of random variables, producing the stated β constant for the regret bound.

Proof sketch of Lemma E.2 utilizes the law of total variance. Strict subgaussianity combined with the assumption of non-zero correlation forces the variance of the conditional mean to be strictly positive, proving the marginal subgaussian parameter strictly bounds the expected conditional one.

Proof sketch of Lemma E.3 bounds the exact noiseless predictive variance by comparing it to a virtual process with noisy observations. This stochastic relaxation allows us to directly apply standard maximal information gain bounds for noisy Gaussian processes to upper-bound the cumulative variance.

E.3 Main results

Proposition 3.3. *The similarity* $\mathrm{sim}^s_{p,e}(c_1, c_2)$ *is a positive semi-definite kernel with* $\mathrm{sim}^s_{p,e} \geq 0$ *and* $\mathrm{sim}^s_{p,e}(c, c) = 1$. *Moreover,* $\mathrm{sim}^\infty_{p,e}(c_1, c_2) = \lim_{s\to\infty} \mathrm{sim}^s_{p,e}(c_1, c_2)$.

Proof. We first show that $\mathrm{sim}^1_{p,e}$ is a positive semi-definite kernel. Recall Definition 3.1:

$$\mathrm{sim}^1_{p,e}(c_1, c_2) := \mathbb{E}_{t\sim p}[\tfrac{1}{|t|}\textstyle\sum_{k=1}^{|t|} \mathbb{1}_{O_k(c_1,t|e)=O_k(c_2,t|e)}]$$

Let $\mathbb{O}$ be the finite set of possible test outputs, i.e., $O_k(c, t|e) \in \mathbb{O}\ \forall c, t, k$ under the specified environment $e \in \mathcal{E}$. For a fixed test t and index k, define the feature map $\phi_{t,k} : \mathcal{V}^{\leq L} \to \{0, 1\}^{|\mathbb{O}|}$ such that $\phi_{t,k}(c)$ is a one-hot vector corresponding to the output $O_k(c, t|e)$. The indicator function can then be written as an inner product:

$$\mathbb{1}_{O_k(c_1,t|e)=O_k(c_2,t|e)} = \langle \phi_{t,k}(c_1), \phi_{t,k}(c_2)\rangle.$$

Since the inner product is a valid kernel, and the class of kernels is closed under addition and scaling by non-negative constants (linearity of expectation), $\mathrm{sim}^1_{p,e}(c_1, c_2)$ is a valid kernel. Concerning non-negativity: The indicator function is non-negative, and the probability measure $p(t)$ is non-negative, thus $\mathrm{sim}^1_{p,e}(c_1, c_2) \geq 0$. Finally, the kernel is normalized since

$$\mathrm{sim}^1_{p,e}(c, c) = \mathbb{E}_{t\sim p}[\tfrac{1}{|t|}\textstyle\sum_{k=1}^{|t|} \mathbb{1}_{O_k(c,t|e)=O_k(c,t|e)}] = \mathbb{E}_{t\sim p}[1] = 1.$$

To extend these results to $\mathrm{sim}^s_{p,e}$ for $s \neq 1$, recall that the class of positive semi-definite kernels is closed under point-wise multiplication (as a consequence of the Schur Product Theorem). Since

$\text{sim}^s_{p,e}(c_1, c_2) = (\text{sim}^1_{p,e}(c_1, c_2))^s$ is the point-wise product of the kernel $\text{sim}^1_{p,e}$ with itself s times, $\text{sim}^s_{p,e}$ is a kernel for any $s \in \mathbb{N}$.

Finally, to prove convergence to functional equivalence, consider two cases: If $\text{sim}^\infty_{p,e}(c_1, c_2) = 1$ then $O_k(c_1, t|e) = O_k(c_2, t|e)\ \forall k \in [|t|], t \in \mathcal{V}^{\leq L} : p(t) > 0$. Therefore, $\text{sim}^1_{p,e}(c, c) = 1$ and thus $\text{sim}^s_{p,e}(c, c) = 1\ \forall s \in \mathbb{N}$, including in the limit. In contrast, if $\text{sim}^\infty_{p,e}(c_1, c_2) \neq 1$, then $\text{sim}^\infty_{p,e}(c_1, c_2) = 0$ and $\exists t \in \mathcal{V}^{\leq L} : p(t) > 0$ and $O_k(c_1, t|e) \neq O_k(c_2, t|e)$ for some $k \in [|t|]$. But then $\text{sim}^1_{p,e}(c_1, c_2) \in [0, 1)$ and thus $\lim_{s\to\infty} \text{sim}^s_{p,e}(c_1, c_2) = \lim_{s\to\infty}(\text{sim}^1_{p,e}(c_1, c_2))^s = 0$. In either case, functional s-similarity converges to functional equivalence as $s \to \infty$. □

Proposition 3.5. *Define* $\mathcal{N}^\infty_c = \{c' \in \mathcal{V}^{\leq L} : \text{sim}^\infty_{p,e}(c, c') = 1\}$. *Then* $\{\mathcal{N}^\infty_c : c \in \mathcal{V}^{\leq L}\}$ *is a set of equivalence classes partitioning* $\mathcal{V}^{\leq L}$.

Proof. By Definition 3.2, $\text{sim}^\infty_{p,e}(c, c') = 1$ if and only if c, c' produce identical outputs on all test cases with non-zero support. Since the equality of outputs is reflexive, symmetric, and transitive, $\text{sim}^\infty_{p,e}$ inherits these properties, partitioning the code space into disjoint equivalence classes $\mathcal{N}^\infty_c$. □

Theorem 3.8. *Let* $\text{sim}^s_{p,e}(c, c') \geq 1 - \varepsilon$ *for* $\varepsilon \in (0, 1)$. *Then*

$$|p(\mathcal{N}^s_c) - p(\mathcal{N}^s_{c'})| \leq \sqrt{2\varepsilon}.$$

Proof. Since $\text{sim}^s_{p,e}$ is a positive definite kernel with $\text{sim}^s_{p,e}(c, c) = 1$ (see Proposition 3.3), there is a feature map $\phi : \mathcal{V}^{\leq L} \to \mathcal{H}$ into a Hilbert space s.t. $\text{sim}^s_{p,e}(x, y) = \langle\phi(x), \phi(y)\rangle$ and $\|\phi(x)\| = 1$.

As a result, we can express the probability of the fuzzy neighborhood $p(\mathcal{N}^s_c)$ as an inner product with the *mean embedding* of the distribution p, which is defined as $\mu_p := \mathbb{E}_{x\sim p}[\phi(x)]$. It holds that

$$\begin{aligned} p(\mathcal{N}^s_c) &= \mathbb{E}_{x\sim p}[\text{sim}^s_{p,e}(c, x)] \\ &= \mathbb{E}_{x\sim p}[\langle\phi(c), \phi(x)\rangle] = \langle\phi(c), \mu_p\rangle. \end{aligned}$$

The difference in probability mass is then governed by the distance in feature space, i.e., we have that

$$\begin{aligned} |p(\mathcal{N}^s_c) - p(\mathcal{N}^s_{c'})| &= |\langle\phi(c), \mu_p\rangle - \langle\phi(c'), \mu_p\rangle| \\ &= |\langle\phi(c) - \phi(c'), \mu_p\rangle|. \end{aligned}$$

Applying the Cauchy-Schwarz inequality then yields the following inequality:

$$|\langle\phi(c) - \phi(c'), \mu_p\rangle| \leq \|\phi(c) - \phi(c')\| \cdot \|\mu_p\|.$$

We bound the first term by quadratic expansion of the squared norm, using that $\|\phi(c)\| = 1$ and that by assumption $\text{sim}^s_{p,e}(c, c') \geq 1 - \varepsilon$. We get

$$\begin{aligned} \|\phi(c) - \phi(c')\|^2 &= \langle\phi(c) - \phi(c'), \phi(c) - \phi(c')\rangle \\ &= \|\phi(c)\|^2 + \|\phi(c')\|^2 - 2\langle\phi(c), \phi(c')\rangle \\ &= 2 - 2\text{sim}^s_{p,e}(c, c') \leq 2 - 2(1 - \varepsilon) = 2\varepsilon. \end{aligned}$$

Thus, $\|\phi(c) - \phi(c')\| \leq \sqrt{2\varepsilon}$. For the second term, by Jensen's inequality, the norm of the mean embedding is bounded by the expected norm:

$$\|\mu_p\| = \|\mathbb{E}_x[\phi(x)]\| \leq \mathbb{E}_x[\|\phi(x)\|] = 1.$$

Multiplying these bounds gives $|p(\mathcal{N}^s_c) - p(\mathcal{N}^s_{c'})| \leq \sqrt{2\varepsilon} \cdot 1$. □

Theorem 3.9. $\widehat{\text{sim}}^s_{p,e;m}$ *is a strongly consistent estimator of* $\text{sim}^s_{p,e}$, *i.e.,*

$$\mathbb{P}[\lim_{m\to\infty} \widehat{\text{sim}}^s_{p,e;m}(c_1, c_2) = \text{sim}^s_{p,e}(c_1, c_2)] = 1\ \forall c_1, c_2 \in \mathcal{V}^{\leq L}.$$

Yet, it is biased unless $s = 1$.

Proof. Let us first define the proportion of passed test cases on a single sampled test suite t_j as a random variable Z_j:

$$Z_j := \frac{1}{|t_j|} \sum_k \mathbb{1}_{O_k(c_1,t_j|e)=O_k(c_2,t_j|e)}$$

Because the test suites $t_j \sim p(t)$ are sampled i.i.d., the terms $Z_1, \ldots, Z_m$ are i.i.d. bounded random variables on $[0, 1]$. By definition, their expected value is the true 1-similarity:

$$\mu := \mathbb{E}[Z_j] = \mathrm{sim}^1_{p,e}(c_1, c_2)$$

We can rewrite the true s-similarity and its Monte Carlo estimator in terms of Z:

- True value: $\mathrm{sim}^s_{p,e}(c_1, c_2) = \mu^s$
- Estimator: $\widehat{\mathrm{sim}}^s_{p,e;m}(c_1, c_2) = (\bar{Z}_m)^s$, where $\bar{Z}_m = \frac{1}{m} \sum_{j=1}^m Z_j$ is the sample mean.

Part 1: Strong Consistency

To prove strong consistency, we must show that the estimator converges almost surely to the true value as the sample size $m \to \infty$.

1. Since Z_j are i.i.d. with finite expectation μ, we can apply Kolmogorov's strong law of large numbers, which dictates that the sample mean converges almost surely to the true mean:
$$\mathbb{P}\left[\lim_{m\to\infty} \bar{Z}_m = \mu\right] = 1$$
2. The function $g(x) = x^s$ is continuous for all $x \in [0, 1]$ and $s \in \mathbb{N}$.
3. By the continuous mapping theorem, if a sequence of random variables $\bar{Z}_m \xrightarrow{a.s.} \mu$, then $g(\bar{Z}_m) \xrightarrow{a.s.} g(\mu)$. Therefore:
$$\mathbb{P}\left[\lim_{m\to\infty} (\bar{Z}_m)^s = \mu^s\right] = 1$$

Substituting back our definitions yields:

$$\mathbb{P}\left[\lim_{m\to\infty} \widehat{\mathrm{sim}}^s_{p,e;m}(c_1, c_2) = \mathrm{sim}^s_{p,e}(c_1, c_2)\right] = 1$$

This concludes the proof of strong consistency.

Part 2: Bias

To evaluate bias, we must check if the expected value of the estimator equals the true value, i.e., for which $s \in \mathbb{N}$ it holds that $\mathbb{E}[(\bar{Z}_m)^s] = \mu^s$.

Case 1: $s = 1$

By the linearity of expectation, the expected value of the sample mean is exactly the population mean:

$$\mathbb{E}[(\bar{Z}_m)^1] = \mathbb{E}[\bar{Z}_m] = \mu = \mu^1$$

Thus, for $s = 1$, the estimator is perfectly unbiased.

Case 2: $s > 1$

For any $s > 1$, the function $g(x) = x^s$ is strictly convex on $[0, 1]$. Now, for Z_j not almost surely constant, we can apply Jensen's inequality for strictly convex functions, which states that the expectation of a convex function is strictly greater than the convex function of the expectation:

$$\mathbb{E}[(\bar{Z}_m)^s] > (\mathbb{E}[\bar{Z}_m])^s = \mu^s$$

Because the expected value of the estimator is strictly greater than the true parameter for all non-degenerate test distributions, the estimator is biased upwards for all $s > 1$. □

Theorem 3.10. *Let* $\mathrm{sim}^1_{p,e}(c_1, c_2) = \mu \in (0, 1)$ *be the true functional similarity. We define the Signal-to-Noise Ratio (SNR) of an estimator* $\widehat{X}$ *as* $\mathrm{SNR}(\widehat{X}) := (\mathbb{E}[\widehat{X}])^2/\mathrm{Var}(\widehat{X})$. *For* $m \geq 1$*, the SNR of the soft estimator dominates the hard estimator by a factor of at least* m^2*:*

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^1_{p,e;m}(c_1, c_2)) \,/\, \mathrm{SNR}(\widehat{\mathrm{sim}}^\infty_{p,e;m}(c_1, c_2)) \geq m(1/\mu)^{m-1} \cdot \tfrac{1-\mu^m}{1-\mu} \geq m^2$$

Proof. Let $Z := \widehat{\mathrm{sim}}^{1}_{p,e;1}(c_1, c_2 \mid e)$ be the random variable representing the similarity score on a single test suite, i.e., $Z \in [0,1]$ and $\mathbb{E}[Z] = \mu \in (0,1)$. The soft estimator is the sample mean of m i.i.d. draws of Z. Its SNR depends on the variance of Z:

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^{1}_{p,e;m}) = \frac{\mathbb{E}[\widehat{\mathrm{sim}}^{1}_{p,e;m}]^2}{\mathrm{Var}(\widehat{\mathrm{sim}}^{1}_{p,e;m})} = \frac{\mu^2}{\mathrm{Var}(Z)/m} = \frac{m\mu^2}{\mathrm{Var}(Z)}.$$

Since Z is bounded in $[0,1]$, $\mathrm{Var}(Z) = \mathbb{E}[Z^2] - \mu^2 \leq \mathbb{E}[Z] - \mu^2 = \mu(1-\mu)$. Thus:

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^{1}_{p,e;m}) \geq \frac{m\mu^2}{\mu(1-\mu)} = \frac{m\mu}{1-\mu}.$$

The hard estimator is an indicator that is 1 if and only if all sampled suites are perfect matches, i.e., $\widehat{\mathrm{sim}}^{\infty}_{p,e;m} = \prod_{j=1}^{m} \mathbb{1}_{Z_j=1}$. Consequently, it is a Bernoulli variable with parameter p_1^m, where $p_1 = \mathbb{P}[Z=1]$. As a result,

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^{\infty}_{p,e;m}) = \frac{(p_1^m)^2}{p_1^m(1-p_1^m)} = \frac{p_1^m}{1-p_1^m}.$$

Now, since $Z \geq 0$, $p_1 \leq \mathbb{E}[Z] = \mu$. Moreover, for $m \geq 1$, $f(x) = \frac{x}{1-x}$ and $g(x) = x^m$ are both monotonously increasing between 0 and 1. As a result,

$$\mathrm{SNR}(\widehat{\mathrm{sim}}^{\infty}_{p,e;m}) \leq \frac{\mu^m}{1-\mu^m}.$$

Taking the ratio of the lower bound of the smooth SNR to the upper bound of the sharp SNR and applying the formula for a finite geometric series in reverse yields

$$\frac{\mathrm{SNR}(\widehat{\mathrm{sim}}^{1}_{p,e;m})}{\mathrm{SNR}(\widehat{\mathrm{sim}}^{\infty}_{p,e;m})} \geq \frac{\frac{m\mu}{1-\mu}}{\frac{\mu^m}{1-\mu^m}} = \frac{m}{\mu^{m-1}} \frac{1-\mu^m}{1-\mu} = \frac{m}{\mu^{m-1}} \sum_{k=0}^{m-1} \mu^k = m \sum_{k=0}^{m-1} \mu^{k-(m-1)}.$$

Finally, we observe that since $\mu \in (0,1)$, every term $\mu^{k-(m-1)} \geq 1$. As there are m such terms:

$$\frac{\mathrm{SNR}(\widehat{\mathrm{sim}}^{1}_{p,e;m})}{\mathrm{SNR}(\widehat{\mathrm{sim}}^{\infty}_{p,e;m})} \geq m \cdot m = m^2.$$

□

Proposition 3.12. *Suppose that the true test suite $t \in \mathcal{V}^{\leq L}$ verifying an implementation $c \in \mathcal{V}^{\leq L}$ in environment $e \in \mathcal{E}$ is almost surely such that $\exists!$ correct $\mathcal{N}_c^{\infty}$. Define* pass@k$(\{\mathcal{N}_{c_i}^{\infty}\}_{i=1}^{k}) := \mathbb{P}_{t\sim p}[\exists i \in \{1,\ldots,k\} : R(c_i, t \mid e) = 1]$. *Then,* pass@k *is maximized by greedily selecting the k code classes $\{\mathcal{N}_{c_i}^{\infty}\}_{i=1}^{k}$ with largest* pass@1.

Proof. Let $S = \{\mathcal{N}_{c_1}^{\infty}, \ldots, \mathcal{N}_{c_k}^{\infty}\}$ be the set of k selected equivalence classes. We aim to maximize pass@k:

$$\text{pass@k}(S) := \mathbb{P}_{t\sim p}\big[\exists \mathcal{N}_c^{\infty} \in S : R(c, t \mid e) = 1\big].$$

Define the success event for a class as $E_c := \{t \in \mathcal{V}^{\leq L} : R(c, t \mid e) = 1\}$. Since by assumption there exists almost surely one correct equivalence class, for any two distinct classes $\mathcal{N}_{c_i}^{\infty}, \mathcal{N}_{c_j}^{\infty} \in S$, the intersection of their success events has zero measure ($\mathbb{P}[E_{c_i} \cap E_{c_j}] = 0$).

This mutual exclusivity allows us to decompose the probability of the union into a sum:

$$\text{pass@k}(S) = \mathbb{P}_{t\sim p}\big[\bigcup_{\mathcal{N}_c^{\infty} \in S} E_c\big] = \sum_{\mathcal{N}_c^{\infty} \in S} \mathbb{P}_{t\sim p}[E_c] = \sum_{\mathcal{N}_c^{\infty} \in S} \text{pass@1}(\{\mathcal{N}_c^{\infty}\}).$$

Maximizing this sum subject to the constraint $|S| = k$ is a standard selection problem, for which the optimal solution is the greedy strategy: selecting the k classes with the largest pass@1. □

Theorem 4.1. *Decompose the reward into independent components $r := r_{obs} + r_{hid}$, where $\mathbb{E}[r_{hid}]$ is finite, and $r_{obs}(x)$ is $\sigma_n(x)$-subgaussian given observation history $\mathcal{H}_n := (x_i, r_{obs}(x_i))_{i=1}^{n-1}$. Define the Thompson sampling policies $x^*_{obs} \sim \mathbb{P}[x^* \mid r_{obs}]$ and $x_n \sim \mathbb{P}[x^* \mid \mathcal{H}_n]$ for $x^* := \arg\max_x r(x)$. Then*

$$\mathbb{E}[\textstyle\sum_{n=1}^{T} r(x^*_{obs}) - r(x_n)] \leq \beta\sqrt{T \cdot \Gamma_T}$$

for $\beta := \sqrt{2\log(|\mathrm{dom}(r)|)} + 3$ and cumulative uncertainty $\Gamma_T := \mathbb{E}[\sum_{n=1}^{T} \sigma_n^2(x_n)]$.

Proof. Without loss of generality $\mathbb{E}[r_{hid}] = 0$, since nonzero expectations can be absorbed into a shifted r_{obs}. Since $r_{hid} \perp r_{obs}$, it follows that $r_{hid} \perp x^*_{obs}, x_n$ and therefore

$$\mathbb{E}[r(x^*_{obs}) - r(x_n) \mid \mathcal{H}_n] = \mathbb{E}[r_{obs}(x^*_{obs}) - r_{obs}(x_n) \mid \mathcal{H}_n].$$

Conditioned on $\mathcal{H}_n$, define $(r'_{obs}, r'_{hid}) \overset{d}{:=} (r_{obs}, r_{hid})$ as an independent copy of the true reward tuple and $r''_{hid} \mid r_{obs} \overset{d}{:=} r_{hid} \mid r_{obs}$ as an independent copy of r_{hid} given full knowledge of r_{obs}. Then it holds that $x^*_{obs} := \arg\max_x r_{obs}(x) + r''_{hid}(x)$ and $x_n := \arg\max_x r'_{obs}(x) + r'_{hid}(x)$, and therefore $\mathbb{E}[r_{obs}(x^*_{obs}) \mid \mathcal{H}_n] = \mathbb{E}[r'_{obs}(x_n) \mid \mathcal{H}_n]$. As a result,

$$\mathbb{E}[r_{obs}(x^*_{obs}) - r_{obs}(x_n) \mid \mathcal{H}_n] = \mathbb{E}[r'_{obs}(x_n) - r_{obs}(x_n) \mid \mathcal{H}_n].$$

Next, we take expectations w.r.t. $\mathcal{H}_n$ and apply Lemma E.1 and Jensen's inequality to obtain

$$\begin{aligned}\mathbb{E}[r(x^*_{obs}) - r(x_n)] = \mathbb{E}[r'_{obs}(x_n) - r_{obs}(x_n)] &= \mathbb{E}[\mathbb{E}[r'_{obs}(x_n) - r_{obs}(x_n)|\mathcal{H}_n]] \\ &\leq \mathbb{E}[\beta\sqrt{\mathbb{E}[\sigma_n^2(x_n)|\mathcal{H}_n]}] \leq \beta\sqrt{\mathbb{E}[\sigma_n^2(x_n)]}.\end{aligned}$$

Finally, we apply Cauchy-Schwarz on the full regret sequence

$$\mathbb{E}[\textstyle\sum_{n=1}^{T} r(x^*_{obs}) - r(x_n)] \leq \beta \sum_{n=1}^{T} \sqrt{\mathbb{E}[\sigma_n^2(x_n)]} \leq \beta\sqrt{T \sum_{n=1}^{T} \mathbb{E}[\sigma_n^2(x_n)]}.$$

□

E.4 Lemmas

Lemma E.1. *Let r, r' be i.i.d., element-wise σ_x-subgaussian, and let $x = f(r) \in \{1, \ldots, |\mathrm{dom}(r)|\}$ for any function f. Then*

$$\begin{aligned}&\mathbb{E}[|r_x - \mu_x|] \leq \sqrt{\mathbb{E}[\sigma_x^2] \cdot (2\log(2|\mathrm{dom}(r)|) + 2)} \text{ and} \\ &\mathbb{E}[|r_x - r'_x|] \leq \beta \cdot \sqrt{\mathbb{E}[\sigma_x^2]}.\end{aligned}$$

for $\beta := \sqrt{2\log(|\mathrm{dom}(r)|)} + 3$.

Proof. Define $Z_z = (r_z - \mu_z)/\sigma_z$. Since $r_z - \mu_z$ is σ_z-subgaussian, Z_z is 1-subgaussian. Cauchy-Schwarz then gives, $\mathbb{E}[|r_x - \mu_x|] = \mathbb{E}[\sigma_x|Z_x|] \leq \sqrt{\mathbb{E}[\sigma_x^2]\mathbb{E}[Z_x^2]}$. Now, $Z_x^2 \leq \max_{1\leq j\leq|\mathrm{dom}(r)|} Z_j^2$, so the tail integral formula for expectations with union bound gives $\mathbb{E}[Z_x^2] \leq \mathbb{E}[\max_{1\leq j\leq|\mathrm{dom}(r)|} Z_j^2] = \int_0^\infty \mathbb{P}[\max_{1\leq j\leq|\mathrm{dom}(r)|} Z_j^2 \geq t]dt \leq \int_0^\infty \min(1, 2|\mathrm{dom}(r)|e^{-t/2})dt = 2\log(2|\mathrm{dom}(r)|) + 2$. The first result then follows immediately. For the second result, apply the triangle inequality $\mathbb{E}[|r_x - r'_x|] \leq \mathbb{E}[|r_x - \mu_x|] + \mathbb{E}[|r'_x - \mu_x|]$, bound the first term as above, and the second by $\sqrt{\mathbb{E}[\sigma_x^2]}$, using that $r' \perp x = f(r)$ and thus $\mathbb{E}[(Z'_x)^2] \leq 1$. This yields an overall factor of $1 + \sqrt{2\log(2|\mathrm{dom}(r)|) + 2}$. Applying the subadditivity of the square root ($\sqrt{A+B} \leq \sqrt{A} + \sqrt{B}$), we can bound this factor by $1 + \sqrt{2\log 2 + 2} + \sqrt{2\log(|\mathrm{dom}(r)|)}$. Since $1 + \sqrt{2\log 2 + 2} \approx 2.84 \leq 3$, we safely obtain the upper bound using β. □

Lemma E.2. *Let X and Y be correlated random variables where X and $X \mid Y$ are strictly subgaussian. Let σ_X^2 denote the optimal subgaussian parameter of X, and $\sigma_{X|Y}^2$ denote the optimal conditional subgaussian parameter of X given Y. Then,*

$$\sigma_X^2 > \mathbb{E}_Y[\sigma_{X|Y}^2].$$

Proof. By strict subgaussianity, the optimal subgaussian parameters are equal to the variance of the distributions. Therefore, the marginal parameter is $\sigma_X^2 = \mathrm{Var}(X)$, and the conditional parameter satisfies $\sigma_{X|Y}^2 = \mathrm{Var}(X \mid Y)$. By the Law of Total Variance, the marginal variance decomposes as:

$$\mathrm{Var}(X) = \mathbb{E}_Y[\mathrm{Var}(X \mid Y)] + \mathrm{Var}(\mathbb{E}[X \mid Y]).$$

Next, note that the conditional mean $\mathbb{E}[X|Y]$ cannot be almost surely constant, otherwise $\mathbb{E}[X|Y] = \mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X]$ a.s., and thus $\mathrm{Cov}(X, Y) := \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[Y(\mathbb{E}[X|Y] - \mathbb{E}[X])] = 0$, contradicting the correlation assumption. As a result, we can apply the strict form of Jensen's inequality to infer that the variance of the conditional mean is strictly positive: $\mathrm{Var}(\mathbb{E}[X|Y]) = \mathbb{E}\left[(\mathbb{E}[X|Y])^2\right] - (\mathbb{E}[\mathbb{E}[X|Y]])^2 > 0$. Substituting the subgaussian parameters into the variance decomposition and applying this strict positivity yields the final bound:

$$\sigma_X^2 = \mathbb{E}_Y[\sigma_{X|Y}^2] + \mathrm{Var}(\mathbb{E}[X \mid Y]) > \mathbb{E}_Y[\sigma_{X|Y}^2].$$

□

Lemma E.3. *Let $r \sim \mathcal{N}(\mu, \Sigma)$ be a multivariate Gaussian with uniformly bounded standard deviations $\sigma_1(x) \leq 1 \; \forall x$ that is consecutively evaluated at locations $(x_t)_{t=1}^T$ with noiseless observations $y_t = r(x_t)$. For any artificial noise level $\eta^2 > 0$, define the noisy maximal information gain $\gamma_T(\eta) := \max_{x_{1:T}} I(\tilde{y}_{1:T}; r)$, where $\tilde{y}_t = r(x_t) + \varepsilon_t$ and $\varepsilon_t \sim \mathcal{N}(0, \eta^2)$. Then, the aggregated predictive variances $\sigma_t^2(x_t) := \mathrm{Var}[r(x_t) \mid y_1, \ldots, y_{t-1}]$ satisfy*

$$\sum_{t=1}^{T} \sigma_t^2(x_t) \leq \frac{2\gamma_T(\eta)}{\log(1 + \eta^{-2})}.$$

Proof. We consider a virtual process that observes the same locations $(x_t)_{t=1}^T$ but receives noisy observations $\tilde{y}_t = r(x_t) + \varepsilon_t$, with i.i.d. noise $\varepsilon_t \sim \mathcal{N}(0, \eta^2)$. Let $\tilde{\sigma}_t^2(x_t) := \mathrm{Var}[r(x_t) \mid \tilde{y}_1, \ldots, \tilde{y}_{t-1}]$ denote the predictive variance under this noisy process.

Let k_{t-1} be the prior covariance vector between y_t and the previously evaluated locations $y_{1:t-1}$ (and also $\tilde{y}_t$ and $\tilde{y}_{1:t-1}$), and let K_{t-1} be the prior covariance matrix of $y_{1:t-1}$. According to the closed-form Gaussian update rules, the exact, noiseless predictive variance is given by

$$\sigma_t^2(x_t) = \sigma_1^2(x_t) - k_{t-1}^\top K_{t-1}^{-1} k_{t-1}.$$

In contrast, for the noisy virtual process, the covariance of the observations incorporates the independent noise variance η^2, yielding

$$\tilde{\sigma}_t^2(x_t) = \sigma_1^2(x_t) - k_{t-1}^\top (K_{t-1} + \eta^2 I)^{-1} k_{t-1}.$$

Because $\eta^2 I$ is strictly positive definite, it follows that $(K_{t-1} + \eta^2 I)^{-1} \preceq K_{t-1}^{-1}$ in the positive semi-definite order. This implies $k_{t-1}^\top (K_{t-1} + \eta^2 I)^{-1} k_{t-1} \leq k_{t-1}^\top K_{t-1}^{-1} k_{t-1}$, meaning the subtracted reduction in variance is strictly smaller in the noisy case. Consequently, the noiseless predictive variance is upper-bounded by the noisy predictive variance:

$$\sigma_t^2(x_t) \leq \tilde{\sigma}_t^2(x_t) \quad \forall t.$$

Intuitively, this confirms that conditioning on the noiseless values provides at least as much information as conditioning on the noisy ones.

Because the standard deviation remains bounded ($\tilde{\sigma}_t(x_t) \leq \sigma_1(x_t) \leq 1$), we can directly apply the standard information gain bound for noisy Gaussian processes to the virtual process (Srinivas et al., 2010, with the relevant statements combined into Lemma 4 of Menet et al., 2026b). We obtain:

$$\sum_{t=1}^{T} \sigma_t^2(x_t) \leq \sum_{t=1}^{T} \tilde{\sigma}_t^2(x_t) \leq \frac{2 \cdot I(\tilde{y}_{1:T}; r)}{\log(1 + \eta^{-2})}.$$

□

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's scope, emphasizing the theoretical analysis of post-generation selection and in-context Thompson sampling.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper includes a dedicated and exhaustive "Limitations" section in the appendix.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Full sets of assumptions and complete proofs for all theoretical claims (e.g., Propositions 3.3, 3.5, 3.12 and Theorems 3.8, 3.9, 3.10, 4.1) are provided in Appendix E

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The paper fully discloses models, datasets, system prompts, and feedback compression techniques in Appendix D. Moreover, the entire code base is made available.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The code was provided in the supplementary materials and is released publicly at `https://github.com/IBM/theory-of-coding-agents` under a permissive license.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: Hyperparameters (e.g., temperature 1.0, top-p 0.95) and other evaluation settings are clearly specified in Appendix D.2 and follow standard practices. Moreover, the code base has configuration files that do not leave any parameters unspecified.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: The experimental tables and figures correctly report the mean and standard error over 5 random seeds. The latter is the correct statistical measure to determine statistical significance of our results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The compute environment (AMD EPYC 7763 64-Core CPU, 8 NVIDIA A100 GPUs) and specific time requirements are detailed in Section C.6 and Appendix D.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research utilizes standard benchmarking datasets and open-weight models, conforming to standard ethical guidelines for machine learning research.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: There is a dedicated impact statement at the beginning of the Appendix.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release high-risk models or scraped datasets that require specific safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: All creators and owners of assets are properly credited. The license and terms of use are respected and listed in Section D.1 and Section D.2 in the Appendix.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: The paper introduces QISKITHUMANEVALSIMX, detailing the specific prompt template used to generate it via Gemini 3 Pro. The dataset is provided in the code as part of the supplementary materials and will be released under a permissive license.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing or human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The research does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: The use of LLMs is the core subject of the paper, and the specific models used (Qwen3, GPT-OSS, MiniMax, Gemini 3 Pro) are thoroughly documented.

    Guidelines:

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