PREDICTING PERFORMANCE OF SYMBOLIC AND PROMPT PROGRAMS WITH EXAMPLES

Anonymous authors
Paper under double-blind review

ABSTRACT

We study *performance prediction*: given a program—either symbolic (e.g., Python) or a prompt executed on an LLM—and a few labeled examples, predict its success rate on unseen instances from the same task distribution. We model each pass/fail execution as a Bernoulli random variable whose success probability θ is the program's unknown performance. Under this model, the posterior over performance is determined by the observed execution outcomes and a performance prior. We estimate *empirical performance priors* from a corpus of programs and tasks. In this corpus, symbolic-program success rates concentrate near zero and one, whereas prompt programs have a more diffuse prior with many programs that pass most, but not all, evaluated tests. These priors explain why a few passing tests support greater confidence in near-perfect performance for symbolic programs than for prompt programs in the studied setting. We introduce RAP (Retrieved Approximate Prior), which retrieves related labeled examples and prompt programs from a corpus to construct a proxy prior. With few target observations, RAP achieves higher mean posterior density at held-out reference success rates and narrower 95% credible intervals than no-corpus and all-corpus baselines in the evaluated in-domain and out-of-domain settings.

1 INTRODUCTION

Imagine that a bank hires a programmer to build a system that determines whether loan applications satisfy its lending requirements. Before deploying the system, the bank wants to know how reliably it will assess future applications. We define a program's *performance*, as its probability of success under the intended input distribution. To evaluate the system, the bank tests it on a small set of loan applications for which the expected decisions are known. Such labeled examples provide valuable evidence about performance, but obtaining reliable labels often requires expert judgment and is therefore costly (Settles, 2009; Kossen et al., 2021). We therefore study how to accurately predict a program's performance from only a few labeled examples.

Traditionally, a developer implements such a screening system by encoding the bank's lending requirements in executable code. LLMs offer an alternative: the developer describes the intended behavior in a natural-language prompt, which guides the selected model's responses to inputs (Reynolds & McDonell, 2021). We refer to the former as a *symbolic program* (SP) and the latter as a *prompt program* (PP). Suppose both implementations pass the same three tests. The top row of Figure 1 illustrates symbolic and prompt-based implementations of this loan application screening system. Should these identical test outcomes lead to the same prediction of future performance? The bank may still be more cautious about deploying the prompt-based implementation. This intuition suggests that performance predictions may depend not only on test outcomes, but also on what is known about the programs before testing. How can we represent this prior information and combine it with the same test evidence?

Bayesian inference provides a framework to study this question. Inferring future performance from test outcomes can be formulated as updating a prior distribution with observations to obtain a posterior distribution (Gelman et al., 2013). We assess the posterior using the density it assigns to a held-out reference success rate and summarize posterior uncertainty using credible intervals. Formally, for a fixed symbolic or prompt program, we model outcomes on independently sampled target inputs as Bernoulli observations with success probability θ. A performance prior $p(\theta)$ describes the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Symbolic Program: code

```
def screen(incomeK, debtK,
score):
  return (incomeK >= 50
          and debtK <= 20
          and score >= 680)
```

Prompt Program: prompt + LLM

"Determine whether a loan application meets these requirements: annual income at least $50,000, debt at most $20,000, and credit score at least 680. Return only 'True' or 'False'."

Three test cases

Input (I, D, S)	Expected	SP	PP
$(60, 10, 720)$	True	✓	✓
$(45, 8, 710)$	False	✓	✓
$(75, 25, 740)$	False	✓	✓

I, D in thousands of dollars; S is credit score

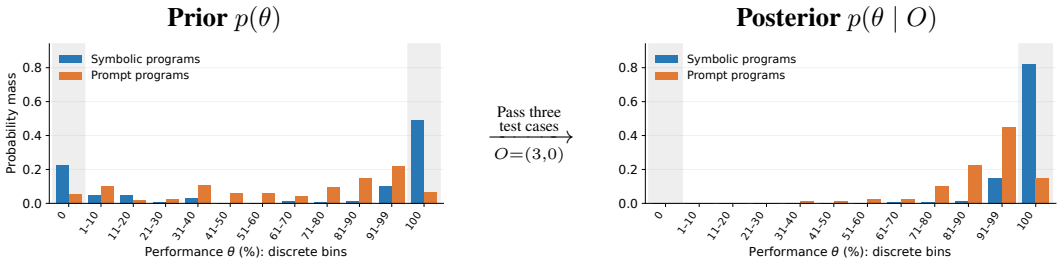


Figure 1: **Same observations, different performance predictions.** Top: a symbolic program (SP) and a prompt program (PP) implement the same loan-screening rule and both pass the three illustrative tests, yielding three successes and no failures, $O = (3, 0)$. Bottom: empirical performance priors (left) and their Bayesian updates (right), where θ denotes the probability of success on a new instance from the target task distribution. Both priors receive the same likelihood factor, θ^3, so the differences between the posteriors arise from the priors rather than the observed outcomes. The SP posterior concentrates near a success rate of one, whereas the PP posterior retains more mass at lower rates. Intuitively, these empirical priors reflect how often sampled programs achieve different success rates: approximately 49% of SPs passed all evaluated instances of their respective tasks, compared with only 6% of PPs. Section 4.1 details the prior construction.

plausible success rates before testing. Given an observation $O = (a, b)$ consisting of a successes and b failures, the likelihood is $p(O \mid \theta) \propto \theta^a(1-\theta)^b$, and Bayes' rule gives $p(\theta \mid O) \propto p(\theta)p(O \mid \theta)$. Thus, when an SP and a PP both pass the same three tests, their likelihoods are identical, and their priors are the only source of difference between their posterior distributions. This leads to our next question: What do the empirical performance priors of SPs and PPs look like?

We estimated these priors by evaluating SPs and PPs across 70 tasks that admit both implementations. This evaluation involved 70,000 code executions and 70,000 LLM inference calls (Section 4.1). The resulting distributions of estimated success rates differ sharply: SP performance concentrates near 0% and 100%, whereas PP performance is more diffuse, with many nearly correct programs. After conditioning on three successes, $O = (3, 0)$, the SP posterior concentrates near perfect performance, while the PP posterior retains more mass at lower success rates (Figure 1). A few successes strongly disfavor very poor programs but do little to distinguish perfect programs from those that occasionally fail. These empirical priors therefore help explain the bank's greater confidence in the symbolic screening system.

Nevertheless, prompt programs offer a flexible way to express tasks in natural language, but deploying them requires reliable performance estimates. A few passing tests may leave substantial uncertainty about their behavior on unseen inputs (Figure 1, lower-right panel). How can we better predict a prompt program's performance when only a few labeled examples are available?

A prior based solely on program type, symbolic or prompt, may offer limited guidance. Prior work on sample-efficient LLM evaluation uses evaluations of other models to select representative benchmark examples and estimate performance (Vivek et al., 2024; Maia Polo et al., 2024). Our goal is instead to construct a prior for a fixed prompt program on its target input distribution. We therefore seek a more informative prior using historical evaluations of related prompt programs on similar examples. A corpus of labeled examples and prompt-program evaluations can supply this evidence, but it also contains evaluations of unrelated programs and tasks. The challenge is to select relevant evaluations, construct an informative prior, and temper its influence when retrieval similarity is low.

We introduce Retrieved Approximate Prior (RAP), which retrieves similar corpus examples and behaviorally similar prompt programs to construct a performance prior. RAP mixes this prior with a uniform prior according to retrieval similarity, then updates it using test examples. We evaluate

RAP on prompt programs spanning different models and prompts. We compare RAP with a no-corpus baseline with a uniform prior and an all-corpus baseline that constructs its prior from the entire corpus. With few target observations, RAP assigns higher posterior density to the reference success rate and produces narrower credible intervals than either baseline, indicating lower posterior uncertainty rather than, by itself, establishing calibration. Our experiments also show benefits from larger corpora. Including target-domain examples in the corpus is particularly helpful when few target test outcomes are available.

Our contributions are threefold:

1. We formulate performance prediction from a few examples as Bayesian inference, combining a performance prior with observed test outcomes to obtain a posterior distribution over success rates.
2. We empirically characterize the performance priors of symbolic and prompt programs, showing how their differences lead to different posterior predictions given identical test outcomes.
3. We introduce and evaluate RAP, which retrieves related examples and prompt programs to construct a proxy performance prior and updates it using test outcomes.

2 Related Work

Predicting benchmark performance. Benchmark compression selects items using cross-model confidence, item response theory (IRT), or source-model representations (Vivek et al., 2024; Maia Polo et al., 2024; Kipnis et al., 2025; Wang et al., 2025; Zhang et al., 2026); Flash-HELM allocates evaluation samples to recover model rankings (Perlitz et al., 2024). Fluid Benchmarking and ATLAS adaptively estimate a model's latent IRT ability (Hofmann et al., 2025; Li et al., 2025). RAP instead predicts a specified prompt program's success probability on a target distribution using retrieved evidence, even without target-domain corpus data.

3 Predicting Performance from Examples

We predict a program's success rate on unseen task instances from a few observed executions. We first introduce the necessary definitions, then present a simple coin-flip model for performance prediction, and finally describe how to evaluate predictions and summarize uncertainty.

3.1 Definitions

Tasks. A task T is a distribution of instances $(x, y) \sim T$ with input x and labeled output y. We assume that instances are sampled independently and identically from T.

Symbolic programs. A symbolic program f is executable code that deterministically maps an input x to an output $f(x)$.

Prompt programs. A prompt program f comprises a prompt, a language model, decoding settings (including temperature), and an execution procedure. Its output on input x follows the distribution $P_f(\cdot \mid x)$, allowing for stochastic execution.

For either program type, we model program construction for task T as sampling $f \sim P(f \mid T)$, for example by asking a person or an LLM to construct a program. This sampling describes variation across programs.

Correctness. Let $c_T(\hat{y}, y) \in \{0, 1\}$ denote the task's correctness checker. It may check exact equality or apply a task-specific test, such as running code tests. On an instance (x, y), an execution produces $\hat{y} \sim P_f(\cdot \mid x)$, and its pass/fail outcome is $\mathbf{Z} = c_T(\hat{y}, y)$.

Performance. Given a program f and a task T, its true performance θ_f^* is the probability that f produces the correct output on an instance drawn from T:

$$\theta_f^*(f, T) = \mathbb{E}_{(x,y)\sim T,\, \hat{y}\sim P_f(\cdot|x)}[c_T(\hat{y}, y)]. \tag{1}$$

Here f is fixed: we average over task instances and the program's possible outputs, not over $P(f \mid T)$. We use θ_f for the unknown success-rate parameter and θ_f^* for its true value, leaving the task T implicit.

Observations. We execute the same program independently on each of the m sampled instances:

$$\mathbf{Z}_i = c_T(\hat{y}_i, y_i), \qquad (x_i, y_i) \sim T, \quad \hat{y}_i \sim P_f(\cdot \mid x_i), \quad i = 1, \ldots, m. \tag{2}$$

We summarize these outcomes by the numbers of successes a and failures b:

$$O = (a, b), \qquad a = \sum_{i=1}^{m} \mathbf{Z}_i, \qquad b = \sum_{i=1}^{m} (1 - \mathbf{Z}_i). \tag{3}$$

3.2 A Simple Coin-flip Model

We use P to denote distributions, $\Pr$ for event probabilities, and p for continuous probability densities. We model each pass/fail outcome $\mathbf{Z}_i$ as a Bernoulli random variable:

$$\mathbf{Z}_i \mid \theta_f \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(\theta_f) \tag{4}$$

For the observed counts $O = (a, b)$, the binomial likelihood is, up to a factor independent of θ_f:

$$\Pr(O \mid \theta_f) \propto \theta_f^a (1 - \theta_f)^b \tag{5}$$

For a continuous prior, Bayes' rule gives the posterior density conditional on the observed counts:

$$p(\theta_f | O) \propto p(\theta_f) \theta_f^a (1 - \theta_f)^b \tag{6}$$

3.3 Prediction and Evaluation

The posterior describes uncertainty about the program's success rate. We evaluate the posterior using a density score and summarize its uncertainty using credible intervals.

Posterior density score (PDS). Ideally, we evaluate the posterior density at the reference success rate, $p(\theta_f^* \mid O)$. In experiments, we substitute an empirical reference success rate estimated from evaluation instances. A higher density score means that the posterior assigns greater density to this reference value.

Credible intervals (CIs). A $(1-\alpha)$ credible interval $[L_\alpha(O), U_\alpha(O)]$ contains posterior probability mass $1 - \alpha$:

$$\Pr(\theta_f \in [L_\alpha(O), U_\alpha(O)] \mid O) = \int_{L_\alpha(O)}^{U_\alpha(O)} p(\theta_f \mid O)\, d\theta_f = 1 - \alpha. \tag{7}$$

Its width summarizes uncertainty under the assumed model and prior.

For the same observed counts, different performance priors can therefore yield different posteriors. The next section estimates these priors empirically for symbolic and prompt programs.

4 The Performance Prior

We use symbolic and prompt programs as a case study of performance priors and Bayesian inference: we construct and compare their empirical performance priors, then examine how these differences affect posterior performance predictions given the same test outcomes.

To estimate prior, we sample a task $T \sim P(T)$, sample a program $f \sim P(f|T)$ written by a person or an LLM, and evaluate its performance on the task. Repeating this process gives success-rate estimates whose distribution approximates the performance prior.

Table 1: Domains used to compare symbolic and prompt programs. We sample 10 programs of each type per task, giving 700 symbolic and 700 prompt programs across 70 tasks, and evaluate each program on 100 instances from its task.

Name	Task given as	Symbolic program type	#Tasks	#Progs/task
Structured regex Ye et al. (2020)	Natural language	Regex expression	20	10
HumanEval Chen et al. (2021)	Natural language	Python script	10	10
ARC Chollet (2019)	Few-shot examples	Python script	20	10
List Rule et al. (2024)	Few-shot examples	Lambda function	20	10

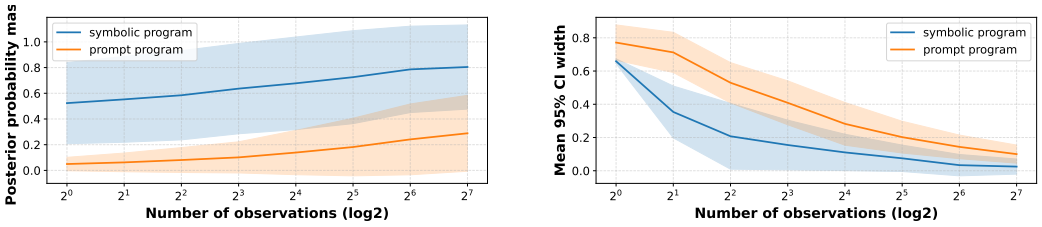
4.1 Empirical Performance Prior

We sampled 70 tasks admitting both symbolic and prompt programs, generated programs of each type, and estimated their success rates on 100 task instances each. Table 1 summarizes the task domains; sampling details are in Appendix A.6.

In Figure 1 (lower-left panel), **symbolic-program estimates cluster near zero and one, whereas prompt-program estimates are more diffuse.** After three successes, the SP posterior concentrates near a success rate of one, whereas the PP posterior retains more probability mass at lower success rates (Figure 1, lower-right panel).

4.2 Comparing Predictions under Empirical

Figure 1 compares the priors and posteriors of symbolic and prompt programs, while Figure 2 quantifies these differences using posterior probability-mass scores and credible interval widths.



(a) Posterior probability mass at the sampled success rate.

(b) Mean width of the 95% equal-tailed credible interval.

Figure 2: Performance prediction under the empirical SP and PP priors as the number of observations increases. Each Monte Carlo trial samples a success rate from the corresponding prior and updates it using simulated test outcomes. Panels show (a) posterior probability mass at the sampled success rate and (b) 95% equal-tailed credible interval width. Lines indicate trial means; shaded bands show $\pm$ one population standard deviation. **Under this prior-matched simulation, SP predictions attain higher probability-mass scores and narrower intervals than PP predictions.**

5 RAP: Retrieved Approximate Prior

To construct a target-specific performance prior, we introduce *Retrieved Approximate Prior (RAP)* to select relevant historical evidence. Following Section 3, we hold the target prompt program f fixed and predict its unknown success-rate parameter θ_f on task distribution T, whose true value is θ_f^*. RAP takes three inputs: a target prompt program f, a set of labeled target examples $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^m$ sampled independently from T, and a corpus $\mathcal{M} = (\mathcal{M}_{\text{examples}}, \mathcal{M}_{\text{programs}})$ of labeled examples and prompt programs. The corpus examples are individual input–label pairs.

RAP retrieves corpus examples similar to the target examples, then selects programs with pass/fail behavior similar to that of f on the retrieved set. Their empirical success rates on that set parameterize a Beta-mixture proxy prior for θ_f (Figure 3). RAP assumes that programs with similar

outcomes on the retrieved examples help predict the target program's success rate. As in Section 3, independent executions of f on $\mathcal{D}$ produce outputs $\hat{y}_j \sim P_f(\cdot \mid x_j)$ and correctness outcomes $\mathbf{Z}_j = c_T(\hat{y}_j, y_j)$. These yield $O = (a, b)$, where $a = \sum_{j=1}^{m} \mathbf{Z}_j$ and $b = m - a$. RAP controls the influence of the retrieved prior by mixing it with a uniform prior according to retrieval similarity, while keeping a fixed base strength for the retrieved components. The target outcomes then update both the component distributions and their mixture weights. This update is conditional on a corpus-derived, target-dependent prior rather than a fixed population prior.

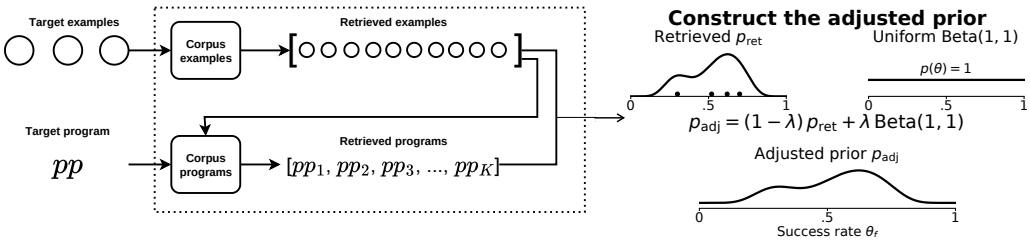


Figure 3: Constructing the adjusted prior. RAP selects corpus examples similar to the target examples and programs with similar pass/fail behavior to the target program. Their success rates (black points) define the equally weighted Beta mixture p_{ret}; similarity sets λ for mixing it with $\text{Beta}(1, 1)$ to obtain p_{adj}.

5.1 Retrieve Similar Examples

For each target input, RAP retrieves the top-n corpus examples by cosine similarity between textual embeddings $e(\cdot)$. It retains their labels for evaluation and takes the union of the retrieval results:

$$\mathcal{D}' = \bigcup_{(x',y')\in\mathcal{D}} \text{Top-n}\left(\{(x, y) \in \mathcal{M}_{\text{examples}}\},\ \cos\big(e(x), e(x')\big)\right). \tag{8}$$

Thus, n is the retrieval budget per target example, not the size of the union. The retrieved set serves as a proxy for the target task distribution T, but textual similarity alone does not ensure similar program performance.

5.2 Retrieve Similar Prompt Programs

RAP next evaluates the target program f on $\mathcal{D}'$ and compares its outcomes with those of corpus programs $f_i \in \mathcal{M}_{\text{programs}}$. Each corpus example is evaluated using the correctness checker of its source task, with the same checker applied to the target and corpus programs. The agreement score $s_{\mathcal{D}'}(f, f_i)$ counts examples on which both programs succeed or both fail. RAP selects the top-K programs by this score:

$$\mathcal{F}' := \text{Top-K}\left(\{f_i \in \mathcal{M}_{\text{programs}}\},\ s_{\mathcal{D}'(f,f_i)}\right). \tag{9}$$

Behavioral retrieval therefore requires additional target-program executions on corpus examples beyond the m target observations. Corpus-program evaluations can be reused when available.

5.3 Construct the Prior and Approximate Posterior

Let a_i and b_i be the success and failure counts of retrieved program $f_i \in \mathcal{F}'$ on $\mathcal{D}'$. For a nonempty retrieved set, define its empirical success rate $r_i = a_i/(a_i + b_i)$ and use a positive base strength κ_0, selected as described in Appendix B.1:

$$\alpha_i = 1 + \kappa_0 r_i, \qquad \beta_i = 1 + \kappa_0(1 - r_i). \tag{10}$$

The unit pseudocounts keep the shape parameters positive, and each component has concentration $\alpha_i + \beta_i = 2 + \kappa_0$. Thus, retrieval determines the component's success-rate estimate, but neither the number of retrieved examples nor a target-outcome discrepancy rescales its concentration.

Using the success-rate parameter θ_f defined in Section 3, the retrieved prior density is

$$p_{\text{ret}}(\theta_f) = \frac{1}{K}\sum_{i=1}^{K} \text{Beta}(\theta_f \mid \alpha_i, \beta_i). \tag{11}$$

Adjusting Prior Strength. We adjust the influence of the retrieved prior through a uniform-mixture weight, rather than an EMD-based change to component concentration. Let $\mathcal{R}$ contain the retained query–corpus input pairs after global deduplication: each retrieved corpus example is paired with the first target input that retrieves it. Define the absolute retrieval-similarity score

$$s_{\text{ret}} = \frac{1}{|\mathcal{R}|}\sum_{(x',x)\in\mathcal{R}} \cos\big(e(x'), e(x)\big). \tag{12}$$

For fixed thresholds $s_{\text{low}} < s_{\text{high}}$, set

$$\lambda = \frac{1}{2}\,\text{clip}\left(\frac{s_{\text{high}} - s_{\text{ret}}}{s_{\text{high}} - s_{\text{low}}},\, 0,\, 1\right), \qquad 0 \le \lambda \le \frac{1}{2}, \tag{13}$$

where $\text{clip}(u, 0, 1) = \min\{1, \max\{0, u\}\}$. High similarity gives $\lambda = 0$, low similarity gives $\lambda = 1/2$, and intermediate scores interpolate linearly. The adjusted prior is

$$p_{\text{adj}}(\theta_f) = (1-\lambda)p_{\text{ret}}(\theta_f) + \lambda\,\text{Beta}(\theta_f \mid 1, 1). \tag{14}$$

Here λ is a prior mixture weight.

6 Experiments

We evaluate whether corpus-derived priors improve performance prediction from few observations. We first describe the experimental setup, then present the main results, examine ablation studies, and discuss the findings.

6.1 Experimental Setup

Tasks and programs. We select tasks from the benchmark domains described in Appendix A.5, focusing on knowledge and reasoning-based multiple-choice questions, Python code generation and embodied agent. A prompt program comprises its prompt, language model, and execution procedure representative prompt program and sample strategy are described in Appendix A.7.

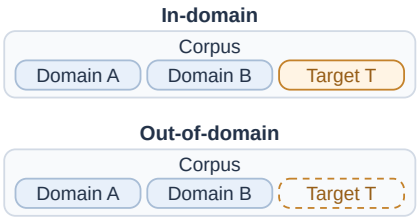


Figure 4: Corpus settings for target domain T. Solid and dashed orange boxes indicate inclusion and exclusion, respectively.

Corpus settings. Figure 4 illustrates the two corpus settings. The in-domain corpus includes the target domain, whereas the out-of-domain corpus excludes it. This comparison assesses the benefit of target-domain historical evidence and RAP's ability to leverage evidence from other domains when such evidence is unavailable. In both settings, we sample labeled examples from the target domain to obtain target observations.

Methods. We compare three approaches:

- **No-corpus** uses a uniform $\text{Beta}(1, 1)$ prior. With a target successes and b failures, its posterior is $\text{Beta}(a + 1, b + 1)$.
- **All-corpus** uses all corpus programs without retrieval and updates its components with target outcomes while retaining equal mixture weights; its formula is given in Appendix A.4.
- **RAP** retrieves related examples and behaviorally similar programs to construct a target-specific prior (Section 5). It adjusts the prior according to retrieval similarity and updates it with target outcomes.

Evaluation measures. We evaluate predictions against a held-out reference success rate. We report two complementary quantities (Section 3):

- **Posterior density score (PDS)** is the posterior density at the reference success rate. Higher values indicate greater density at that rate, not necessarily better calibrated uncertainty.
- **95% credible interval width** measures posterior uncertainty using the difference between the 97.5th and 2.5th percentiles of the full posterior mixture.

6.2 Main Results

Qualitative Analysis: The Role of Priors in Posterior Updating. Figure 5 illustrates how a relevant prior can help interpret limited target observations. As observations increase from two to eight, RAP's posterior mean moves closer to the reference success rate, whereas the no-corpus posterior shifts toward one and away from the reference. In this example, RAP's retrieved prior moderates the overly optimistic estimate induced by an all-success observation sequence. Meanwhile, the all-corpus posterior remains broader and multimodal, reflecting heterogeneous historical evidence. This contrast illustrates the potential benefit of selecting relevant evidence rather than relying solely on target observations or pooling the entire corpus.

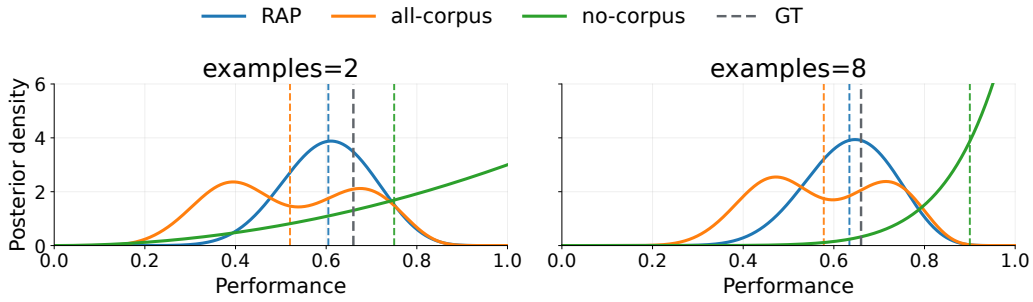


Figure 5: Posterior performance distributions for an example target program at increasing observation counts. Dashed vertical lines show posterior means and the held-out reference success rate.

Quantitative Analysis: Comparison with baselines. Figure 6 compares RAP with both baselines in the in-domain and out-of-domain settings. With few target observations, RAP achieves higher mean PDS and narrower credible intervals than either baseline, indicating that its posterior places more density at the reference success rate while expressing less uncertainty. RAP performs better when the corpus includes the target domain, particularly at low observation counts, highlighting the value of relevant historical evidence.

6.3 Ablation Study

Hyperparameter selection. We select prior strength and similarity thresholds using mean posterior-estimate error across $m \in \{1, 2, 4, 8, 16, 32, 64\}$ (Figure 7 in Appendix B.1).

Point Prediction Accuracy. Using posterior means on the same in-domain records, RAP achieves lower mean absolute error than both baselines at every plotted observation count ($m = 1$ through 64); its advantage over no-corpus narrows as observations accumulate (Appendix B.4).

Corpus size. With the selected hyperparameters, larger corpora improve in-domain mean PDS most clearly when target observations are few (Figure 8).

Corpus-example selection at a matched execution budget. In a retrospective paired ablation on six MMLU domains, we compare eight RAP-selected corpus examples with eight examples sampled uniformly from the same candidate pool. Both arms share four target observations and the same inference procedure, requiring 12 archived target-program executions per record (3,600 pairs per corpus setting). RAP versus Random obtains mean PDS 2.580 versus 2.230 when the target

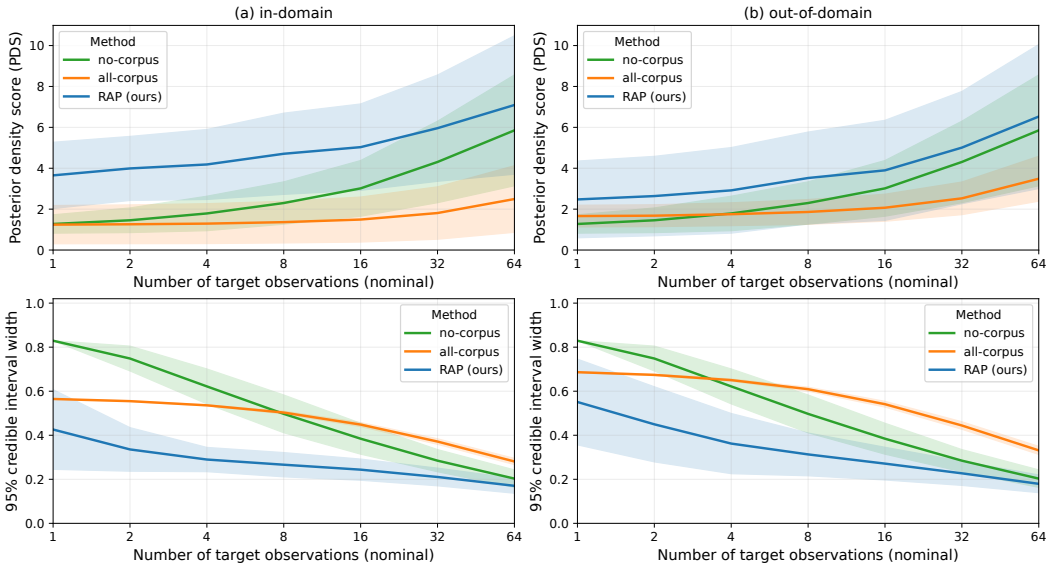


Figure 6: Performance prediction with (left) and without (right) target-domain corpus examples. Top: PDS. Bottom: 95% equal-tailed credible interval width. Lines and bands show means ± one population standard deviation. See Appendices A.2 for evaluation details.

domain is included in the pool and 2.298 versus 2.094 when it is excluded (Figure 9; details in Appendix B.3). This result suggests that RAP's retrieved prior can improve predictive performance over the random-selection baseline at the same target-program execution budget.

7 Conclusion, Limitations, and Future Work

We formulate program performance prediction as inference over a fixed program's success rate on a task distribution. Empirical priors for the sampled symbolic and prompt programs show why identical test outcomes can support different predictions. RAP constructs a target-specific prior from retrieved examples and programs, mixes it with a uniform prior according to retrieval similarity, and updates it with target observations. With few observations, RAP yields higher mean PDS and narrower credible intervals than the no-corpus and all-corpus baselines in the evaluated settings. A retrospective paired ablation also shows higher mean PDS than random corpus-example selection at the same target-program execution budget.

Limitations. First, our findings are empirical and limited to the evaluated programs, tasks, and corpus settings. Second, RAP estimates a program's average success rate to inform developer decisions, but does not identify which inputs cause failures or explain why they occur. A high predicted success rate therefore does not guarantee reliable behavior on every input. Third, RAP depends on the quality and relevance of the available corpus. Inaccurate labels, unrepresentative examples, or unrelated programs can produce misleading prior evidence, and textual similarity alone does not ensure that this evidence transfers to the target task.

Future work. Independent validation and simulations under prior mismatch could assess generalization and uncertainty calibration. Further ablations could isolate behavioral program selection and uniform-prior mixing. Active selection of corpus evaluations and low-rank approximations of the evaluation matrix could reduce computation, while input-specific performance models could identify where failures occur. Deployment decisions would additionally require explicit quality thresholds and evaluation of false acceptance and false rejection.

AI USE STATEMENT

We used generative AI tools to assist with writing code and polishing the manuscript for clarity and readability. The authors reviewed the AI-assisted code and text and take responsibility for the final content, results, and claims.

REFERENCES

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

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 3rd edition, 2013. doi: 10.1201/b16018. URL https://doi.org/10.1201/b16018.

Valentin Hofmann, David Heineman, Ian Magnusson, Kyle Lo, Jesse Dodge, Maarten Sap, Pang Wei Koh, Chun Wang, Hannaneh Hajishirzi, and Noah A. Smith. Fluid language model benchmarking. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2509.11106.

Alex Kipnis, Konstantinos Voudouris, Luca M. Schulze Buschoff, and Eric Schulz. metabench - a sparse benchmark of reasoning and knowledge in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4T33izzFpK.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5753–5763. PMLR, 2021. URL https://proceedings.mlr.press/v139/kossen21a.html.

Peiyu Li, Xiuxiu Tang, Si Chen, Ying Cheng, Ronald Metoyer, Ting Hua, and Nitesh V. Chawla. Adaptive testing for LLM evaluation: A psychometric alternative to static benchmarks. *arXiv preprint arXiv:2511.04689*, 2025. URL https://arxiv.org/abs/2511.04689.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: evaluating LLMs with fewer examples. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34303–34326. PMLR, 2024. URL https://proceedings.mlr.press/v235/maia-polo24a.html.

Yotam Perlitz, Elron Bandel, Ariel Gera, Ofir Arviv, Liat Ein-Dor, Eyal Shnarch, Noam Slonim, Michal Shmueli-Scheuer, and Leshem Choshen. Efficient benchmarking (of language models). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2519–2536. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.139. URL https://aclanthology.org/2024.naacl-long.139/.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–7. Association for Computing Machinery, 2021. doi: 10.1145/3411763.3451760. URL https://doi.org/10.1145/3411763.3451760.

Joshua S Rule, Steven T Piantadosi, Andrew Cropper, Kevin Ellis, Maxwell Nye, and Joshua B Tenenbaum. Symbolic metaprogram search improves learning efficiency and explains rule learning in humans. *Nature Communications*, 15(1):6847, 2024.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. URL https://burrsettles.com/pub/settles.activelearning.pdf.

Rajan Vivek, Kawin Ethayarajh, Diyi Yang, and Douwe Kiela. Anchor points: Benchmarking models with much fewer examples. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1576–1601. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.eacl-long.95. URL https://aclanthology.org/2024.eacl-long.95/.

Shaobo Wang, Cong Wang, Wenjie Fu, Yue Min, Mingquan Feng, Isabel Guan, Xuming Hu, Conghui He, Cunxiang Wang, Kexin Yang, et al. Rethinking llm evaluation: Can we evaluate llms with 200x less data? *arXiv preprint arXiv:2510.10457*, 2025.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Benchmarking multimodal regex synthesis with complex structures, 2020. URL https://arxiv.org/abs/2005.00663.

Yueqi Zhang, Jin Hu, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Yiwei Li, Jiayi Shi, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, and Kan Li. Learning more from less: Unlocking internal representations for benchmark compression. *arXiv preprint arXiv:2602.00710*, 2026. URL https://arxiv.org/abs/2602.00710.

A Experimental Details

A.1 Hyperparameter Settings

We use the following hyperparameter settings for RAP in our main experiments:

- Base prior strength: $\kappa_0 = 30$.
- Similarity thresholds: $s_{\text{low}} = 0.1$, $s_{\text{high}} = 0.4$.
- Retrievaled prompt program number: $K = 5$.

A.2 Main Comparison Details

Figure 6 uses $s_{\text{low}} = 0.1$, $s_{\text{high}} = 0.4$, and base strength 30, with evidence-updated mixture weights. Both models and all domains, are pooled at corpus size 160 per domain; historical baseline conventions are retained. PDS is evaluated at the held-out reference success rate.

A.3 Computational Cost

Prior construction across all experiments costs approximately $300 in API usage, covering corpus and target-observation executions.

A.4 The All-Corpus Baseline

Using the notation of Section 5, the all-corpus baseline evaluates each corpus program $f_i \in \mathcal{M}_{\text{programs}}$ on all examples in $\mathcal{M}_{\text{examples}}$, without retrieval. Let a_i and b_i denote its success and failure counts, and let $\alpha_i^{\text{all}} > 0$ and $\beta_i^{\text{all}} > 0$ denote the corresponding baseline Beta shape parameters. These parameters are distinct from RAP's fixed-strength construction. For the target program f, let θ_f be its unknown success-rate parameter on task T and $O = (a, b)$ its target observation counts. The baseline updates each component with O while retaining equal mixture weights:

$$p_{\text{all-corpus}}(\theta_f \mid O) = \frac{1}{|\mathcal{M}_{\text{programs}}|} \sum_{f_i \in \mathcal{M}_{\text{programs}}} \text{Beta}\big(\theta_f \mid \alpha_i^{\text{all}} + a,\ \beta_i^{\text{all}} + b\big)\,. \tag{15}$$

The label *all-prior* in the plots refers to this all-corpus baseline. Unlike RAP, this baseline retains equal component weights after updating their parameters; the displayed mixture is not an exact Bayesian update of a mixture prior.

A.5 Benchmark Details

Our benchmarks cover three broad classes of prompt-programming tasks: knowledge-intensive question answering, code generation, and embodied agentic tasks. Specifically, we use six MMLU-PRO domains, two code-generation benchmarks, and two ALFWorld task types.

A.6 Part 1: Comparison Between Symbolic Programs and Prompt Programs

We sample 70 tasks across domains ranging from inductive reasoning to natural-language code generation (Table 1). Each task admits both program types: for example, phone-number validation can use a regular expression (*sp*) or an LLM prompt (*pp*). For each task, we prompt an LLM to generate 10 programs of each type. Human-written programs and prompts are not included.

Each task supplies an instance sampler with oracle-labeled outputs. For a program f, we estimate its success rate by averaging pass/fail indicators over $N = 100$ sampled instances:

$$\tilde{\theta}_f = \frac{1}{N} \sum_{i=1}^{N} \mathbf{Z}_i, \qquad (x_i, y_i) \sim T. \tag{16}$$

Here $\mathbf{Z}_i$ indicates whether execution i passes the task checker (Section 3). The stated sampling design entails 70,000 symbolic-program executions and 70,000 LLM calls. These observed rates approximate, but do not establish, true performance.

Table 2: Benchmark domains and example counts for prompt-program evaluation. Most domains provide 200 held-out test examples, 128 candidate target observations, and 400 corpus examples. HumanEval is used only as a corpus pool, and ALFWorld uses smaller corpus pools. See Appendix A.5 and the following evaluation details.

Type	Domain	#Examples
Knowledge (MMLU-PRO)	math	728
	physics	728
	chemistry	728
	economics	728
	law	728
	engineering	728
Code Generation	MBPP	728
	HumanEval	168
Agentic (ALFWorld)	pick and place	448
	cool then place	448

We compare symbolic and prompt programs across the domains listed in Table 3. Where the available instances are insufficient, we augment them to support evaluation on 100 instances per task. These finite-sample success rates approximate the programs' true performance.

Table 3 lists the task specifications, prompt-program execution models, and augmentation settings for this comparison.

Table 3: Benchmark tasks used in Part 1, including task specification formats and prompt-program execution models.

Benchmark	Task Specification	Execution Model	Data Augmented
Structured Regex	Natural language	GPT-4o-mini	Yes
HumanEval	Natural language	GPT-4o-mini	Yes
ARC	Few-shot examples	Llama-3.1 (fine-tuned)	No
List	Few-shot examples	GPT-5-mini	Yes

Simulation under the empirical priors. For each Monte Carlo trial, we draw a success rate from the discrete empirical prior, then sample Bernoulli observations and update its probability masses. Figure 2 reports the posterior mass at the sampled rate and the mean width of the 95% equal-tailed credible interval. For the interval analysis, we use 5,000 trials per program type and observation count. The symbolic-program prior yields higher mass scores and narrower intervals over the tested counts. Empirical coverage ranges from 96.5% to 99.1% across these matched-prior simulations.

A.7 Part 2: Prompt-Program Evaluation

In the second part of the experiments, we evaluate 60 prompt programs, including 24 from multiple-choice question answering, 24 from code generation, and 12 from embodied AI tasks. We consider two language models, GPT-4o-mini and GPT-5-mini, and evaluate two prompting strategies: chain-of-thought (CoT), which explicitly elicits intermediate reasoning before producing an answer, and a vanilla setting, which does not allow the model to output reasoning content.

For each domain, we use the official checker to verify correctness.

We use three sets of examples: a held-out test set to estimate reference performance, a target observation set from which the predictor receives examples, and a corpus pool available for retrieval. HumanEval is used only as a corpus pool because it has few examples. For agentic domains, we reduce the corpus pool size because multi-turn environment executions are expensive.

We use DSPy to implement instruction templates for multiple-choice questions, code generation, and embodied-agent tasks. The representative instructions below omit fixed framework formatting.

MMLU-PRO. For multiple-choice question answering, the prompt encourages careful reasoning over the question and answer options:

> Analyze the multiple-choice question carefully. Reason step by step, identify the key concept, evaluate each option, eliminate weaker choices, and select the single best answer. Return only the final answer in the required format.

MBPP. For code-generation tasks, the prompt instructs the model to produce a correct and readable Python function:

> Write a clean, efficient Python function that solves the problem and passes all tests. Read the description, constraints, examples, and assertions carefully, handle edge cases, and ensure correctness and readability.

ALFWorld. For embodied agentic tasks, the prompt specifies the task context and asks the model to select the next action:

> You are an embodied household agent in an interactive environment. Your goal is to complete the given task description by selecting the best next action based on the task description, current observation, previous trajectory, and possible actions.

B Ablation Study

B.1 Sensitivity to Prior Strength and Similarity Thresholds

We vary $\kappa_0 \in \{20, 30, 40\}$, $s_{\text{low}} \in \{0.1, 0.2, 0.3\}$, and $s_{\text{high}} \in \{0.4, 0.5, 0.6\}$. Each of the 27 combinations is evaluated on the same 2,400 validation records at each $m \in \{1, 2, 4, 8, 16, 32, 64\}$. Reference success rates are estimated from 200 outcomes per program and domain.

We select the configuration minimizing the arithmetic mean, equally weighted across the seven counts $m \in \{1, 2, 4, 8, 16, 32, 64\}$, of the mean absolute error between the posterior mean and the empirical reference rate.

Hyperparameters are selected on a separate validation set that is disjoint from the final evaluation set.

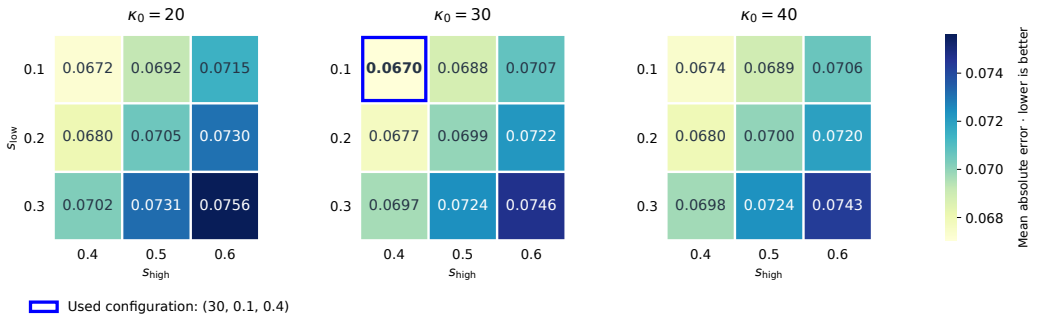


Figure 7: Mean absolute posterior-mean error against empirical reference rates, averaged equally over seven observation counts ($m = 1$ through 64; lower is better). Columns vary κ_0; rows and columns within each heatmap vary the lower and upper similarity thresholds. Values pool both corpus settings on the same retrospective validation programs; all panels share a color scale.

B.2 Effects of Corpus Size

We vary the number of corpus examples per domain in the in-domain setting in Figure 8. We recompute RAP posteriors for corpus sizes 20, 40, 80, and 160 using the selected $(\kappa_0, s_{\text{low}}, s_{\text{high}}) = (30, 0.1, 0.4)$ and each size's archived retrieval similarities and component success rates. Each point pools 4,200 records from both models and all seven domains; the size-160 predictions match the main comparison exactly. Larger corpora improve in-domain PDS most clearly at small observation counts. At $m = 4$, increasing the corpus size from 20 to 160 raises mean PDS from 3.184 to 4.185.

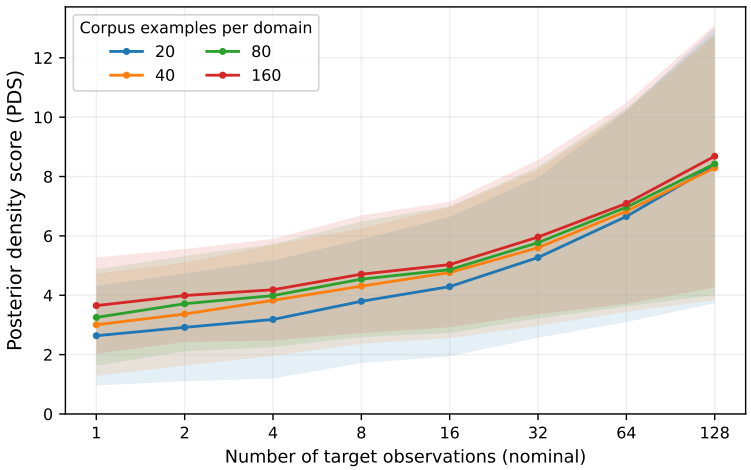


Figure 8: Posterior density score (PDS) for RAP in the in-domain setting as the number of corpus examples per domain varies. Posteriors are recomputed from archived retrieval results with $(\kappa_0, s_{\text{low}}, s_{\text{high}}) = (30, 0.1, 0.4)$, pooling both models and all seven domains. Lines show means across evaluation records; shaded bands show $\pm$ one population standard deviation, not confidence intervals. Larger corpora improve mean PDS most clearly with few target observations.

B.3 Corpus-Example Selection at a Matched Execution Budget

Figure 9 isolates the selection of eight corpus inputs: RAP chooses them by cosine retrieval, whereas Random samples eight uniformly without replacement from the *same* candidate pool. Both arms use the same four target-domain observations, archived 24-program cohort, held-out performance reference, similarity-gated prior construction, and posterior update; only the eight selected corpus inputs change. Thus each arm uses 12 target-program executions per record (four target observations and eight corpus evaluations).

We evaluate six verified MMLU target domains, five example seeds, and five corpus seeds, yielding 3,600 paired records per corpus setting. The in-domain candidate pool contains 160 inputs from each of six domains, *including* the target; the out-of-domain pool contains 160 inputs from each of the other five domains. RAP takes the two nearest inputs per target observation, globally deduplicates them, and backfills from the nearest unused inputs to reach eight; Random draws eight distinct inputs using a reproducible seed independent of cosine similarity. Selected inputs are disjoint from the four target observations, and their identities and binary outcomes are verified against archived evaluations. In both arms, the top five behavioral programs form a strength-30 Beta-mixture prior, gated by the selected examples' cosine similarity with thresholds $(s_{\text{low}}, s_{\text{high}}) = (0.1, 0.4)$, before updating on the four shared target observations.

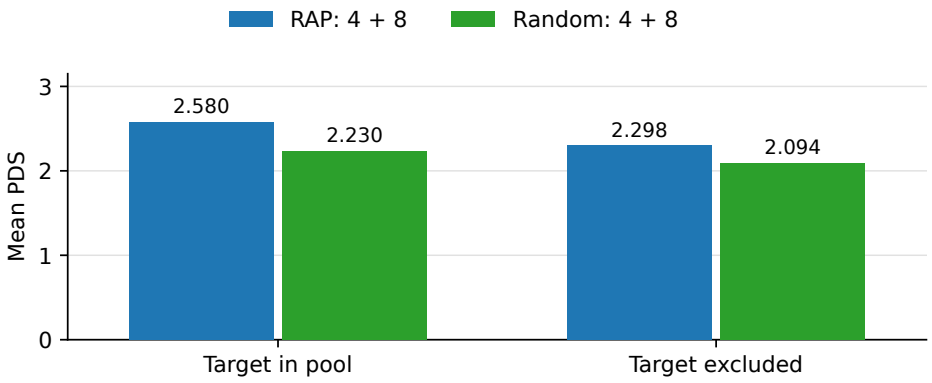


Figure 9: Mean posterior density score (PDS) for RAP-selected versus randomly selected corpus inputs, paired on four target observations (3,600 records per setting across six MMLU domains).

B.4 In-Domain Absolute Error

We supplement the distributional evaluation with the absolute error (AE) of the posterior mean. For each method, we compute $\mathrm{AE} = |\mathbb{E}[\theta_f \mid O] - \hat{\theta}_{\mathrm{ref}}|$, where $\hat{\theta}_{\mathrm{ref}}$ is the held-out empirical reference success rate. All methods use posterior means, including no-corpus, whose point prediction is $(a+1)/(a+b+2)$. We reuse the in-domain main-comparison predictions with $(\kappa_0, s_{\mathrm{low}}, s_{\mathrm{high}}) = (30, 0.1, 0.4)$ and corpus size 160 per domain, without further tuning.

RAP has lower mean AE than both baselines at every plotted observation count (Figure 10). At $m = 4$, mean AE is 0.0542 for RAP, 0.1335 for no-corpus, and 0.2166 for all-corpus. At $m = 64$, these values decrease to 0.0337, 0.0408, and 0.0990, respectively, and RAP's advantage over no-corpus narrows. These results support improved point predictions on the evaluated records, but do not establish uncertainty calibration. AE is measured against an estimated reference rate rather than true performance.

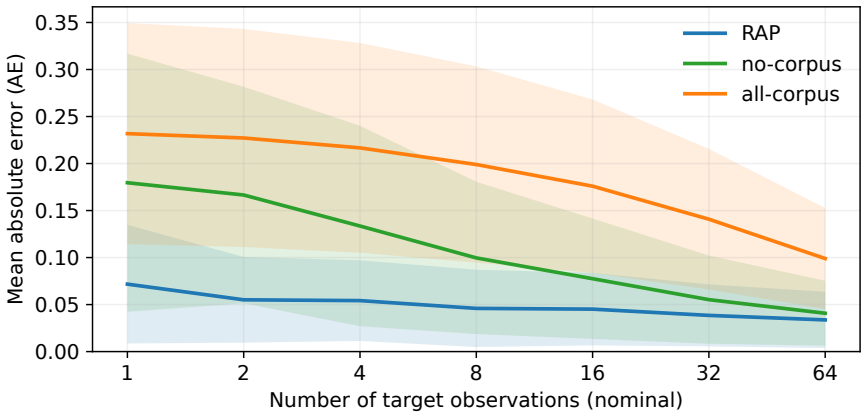


Figure 10: In-domain absolute error of posterior-mean predictions (lower is better). Lines show means across evaluation records; shaded bands show ± one population standard deviation.

C Reference Coverage

Reference coverage is the fraction of records whose 95% credible intervals contain the held-out reference success rate.

Figure 11 reports in-domain reference coverage using the same fixed-threshold configuration and evaluation records as the PDS and interval-width results in Figure 6. At nominal $m = 4$, RAP achieves 97.69% coverage, compared with 96.31% for no-corpus and 63.81% for all-corpus. RAP's coverage is 95.90% at $m = 64$.

Neither higher PDS nor narrower intervals establishes calibrated uncertainty. Moreover, the reference success rates are estimated from finite evaluation sets, so reference coverage is only a proxy for coverage of true performance.

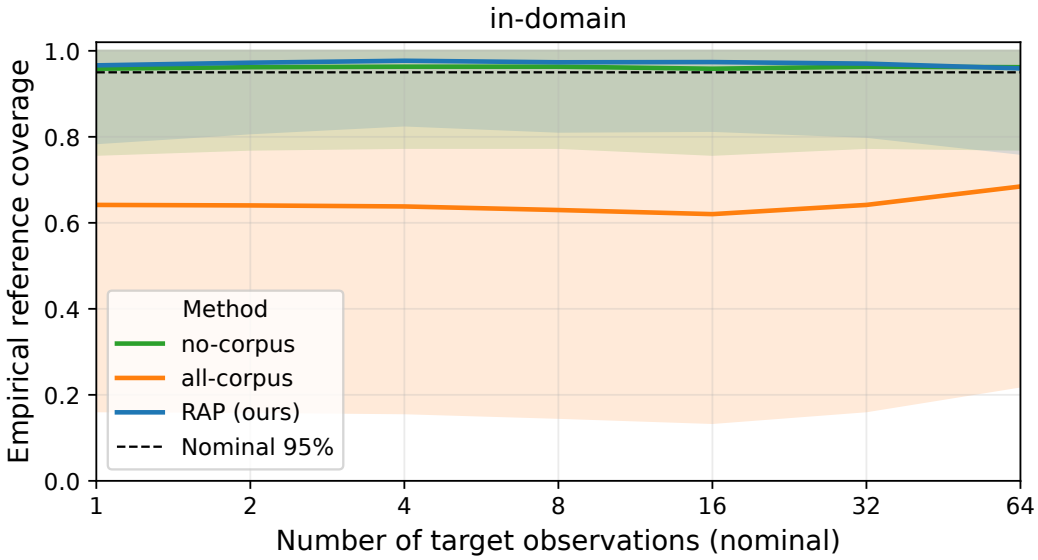


Figure 11: In-domain empirical reference coverage of 95% equal-tailed credible intervals, shown only through $m = 64$. Lines show the fraction of intervals containing the held-out reference success rate; the dashed line marks 95%. Matching-color bands show $\pm$ one population standard deviation of binary coverage indicators, clipped to $[0, 1]$, not confidence intervals.