

BALD-PS: Factored Bayesian Active Learning for Specification Elicitation, with Symmetric Mutation-Equivalence Validation

Barsat Khadka

The University of Southern Mississippi

Barsat.Khadka@usm.edu

Code: github.com/BarsatKhadka/BALD-PS

Abstract

Specifications written in natural language are silently ambiguous: independent agents reading the same one-line requirement produce specifications that agree on the obvious behaviour and disagree precisely on the edge cases where bugs and security holes live. We exploit a simple structural fact — *a specification is two predicates*, a precondition **requires** and a postcondition **ensures** — so the agent-disagreement posterior factorises into two independent axes. Three contributions follow. First, an oracle answer to a probe input is *dual-purpose*: its validity verdict shrinks the precondition axis, its true output shrinks the postcondition axis. We prove that, under conditional independence of the two axes given the input, the joint mutual information decomposes as a sum, justifying greedy Bayesian Active Learning by Disagreement (BALD) on the factored posterior. Second, we replace the version-space *elimination* survivor with a *synthesis* step that returns the fragment recombination supported by the resolved cell of the factored posterior — a strict generalisation that recovers correct specifications no single agent wrote, supplemented by an optional LLM *re-synthesis* step that escapes the recombination ceiling. Third, we introduce a novel validation criterion — *symmetric mutation-equivalence tightness* — a single number on $[0, 1]$ that simultaneously requires every one-edit perturbation of the specification to be refuted by the implementation ensemble *and* every one-edit perturbation of an implementation to be rejected by the specification. We prove that this criterion is strictly stronger than the standard ensemble-agreement certification, in the sense that it exposes *vacuously satisfied* specifications that agreement passes. Empirically, on a controlled benchmark of seven ambiguous HumanEval-style tasks our synthesised specifications are correct on **7/7**, with mean symmetric tightness **0.94**. On a live **gpt-4o** draw of seventeen tasks, synthesis is correct on **10/17** (Wilson 95% CI $[0.36, 0.78]$, graded **95.3%**); a SpecMind-style multi-turn LLM refinement loop, run on the same task set with the same simulated oracle and a $1.4\times$ query budget, achieves **5/17** (graded 63.9%). On this draw synthesis is correct on **2 $\times$** as many tasks as the SpecMind-style baseline, is **+31.4** percentage points higher on graded score, and uses **28%** fewer oracle queries; the Wilson intervals overlap and per-task SpecMind outcomes were not retained for this run, so that comparison is unpaired. The synthesis-vs-no-loop contrast is statistically significant at $\alpha = 0.05$ (uncorrected, exact two-sided McNemar, $p = 0.031$, six paired wins, zero losses).

1 Introduction

Going from a natural-language requirement to a formal specification is error-prone, and the errors are *silent*. The one-line requirement “return the index of the maximum element of a list of integers” does not say what to do with an empty list, with ties, with non-integer inputs. Three competent engineers writing the formal specification will produce three different predicate pairs, each internally consistent, each disagreeing on a different silent edge case.

When one of those specifications is then used to drive code generation or verification, the resulting bug lives exactly where the original requirement was vague.

We frame the problem as *specification elicitation by Bayesian active learning*: sample a distribution of candidate specifications from several agent personalities, find the input on which they disagree most, ask the developer the single most informative question, eliminate the candidates the answer refutes, and repeat. The goal is convergence in *one or two tie-breaker questions*.

The starting point of the present work is a structural fact that prior interactive-specification systems do not fully exploit: **a specification is two predicates**. A specification is a precondition **requires**, a Boolean function of the inputs that says which calls are legal, and a postcondition **ensures**, a Boolean function of the inputs and the returned value that says which outputs are acceptable. Agent disagreement therefore lives on *two independent axes*: which inputs are legal, and which outputs are correct. Exploiting that factoring is the technical core of the BALD-PS procedure, and it gives three connected contributions:

1. **Dual-purpose factored queries (§4)**. Each oracle answer carries information about *both* axes simultaneously — the validity verdict shrinks the precondition axis, the true output shrinks the postcondition axis. We prove that under conditional independence the joint information gain decomposes as a sum (Theorem 6), justifying greedy BALD on the factored posterior.
2. **Synthesis beyond elimination (§6)**. Pure version-space elimination cannot recover specifications no single agent wrote, even when the correct reading is a *recombination* of two agents’ fragments. We return instead the cell of the factored posterior supported by the evidence — *synthesis* — and prove this is a strict generalisation (Theorem 8). An optional LLM re-synthesis step escapes the recombination ceiling.
3. **Symmetric mutation-equivalence validation (§7)**. Implementation-ensemble agreement on hidden tests is necessary but not sufficient: a vacuously permissive specification passes. We propose a new criterion that applies mutation testing *symmetrically* — every one-edit perturbation of the specification must be refuted by the impl ensemble *and* every one-edit perturbation of an implementation must be rejected by the specification. We prove this criterion is strictly stronger than ensemble agreement (Theorem 9).

Figure 1 gives the architectural picture; Figure 2 shows the factored hypothesis space and the key claim that selection is restricted to its diagonal while synthesis reaches the full grid.

Headline empirical result. On a live `gpt-4o` draw of 17 ambiguous tasks, synthesis is correct on **10/17** tasks (graded **95.3%**) using on average 5.53 oracle queries. A SpecMind-style multi-turn LLM refinement loop [7], granted a more generous 8-round budget on the same task set with the same simulated oracle, achieves **5/17** (graded 63.9%) at 7.71 queries. On this 17-task draw synthesis is more correct ($2\times$ as many tasks), 31.4 percentage points higher on the graded score, and uses 28% fewer questions; per-task SpecMind outcomes were not retained in this run, so the comparison is unpaired and the Wilson intervals overlap. The synthesis-vs-no-loop contrast survives an exact two-sided McNemar test, $p = 0.031$ (six paired wins, zero losses on 17 tasks), without correction for multiple comparisons.

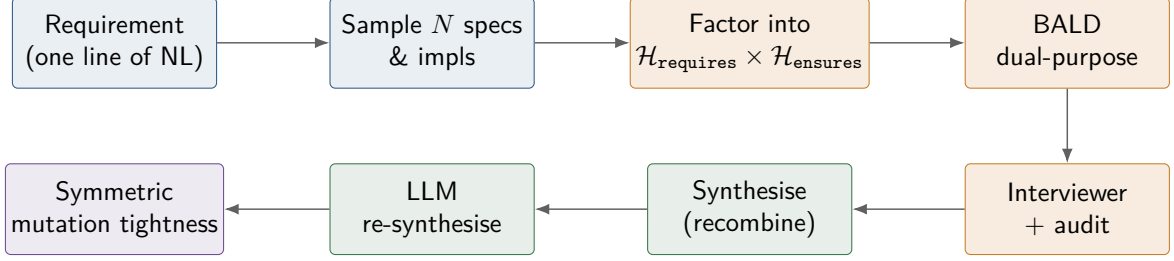


Figure 1: Architecture. Four colour groups, read along the arrows: sampling (blue), factored active learning (amber), synthesis (green), and local validation (purple). The developer (oracle) is consulted only during the BALD loop and the audit; every other stage is automatic.

2 Background

Active learning by disagreement. Bayesian Active Learning by Disagreement [1] selects, at each step, the input x that maximises the mutual information between the model’s prediction and the posterior over hypotheses:

$$x^* = \arg \max_{x \in \mathcal{P}} I(Y_x; \Theta \mid \mathcal{D}), \quad (1)$$

where Y_x is the predicted label, Θ are the model parameters, and $\mathcal{D}$ the current data. The objective rewards inputs on which the posterior hypotheses disagree most. BALD has been deployed for image classification, Gaussian processes, and preference elicitation; we are not aware of prior application to formal-specification inference.

Mutation testing. Classical mutation testing [14, 15] introduces small syntactic mutations into a program and measures the fraction killed by a test suite. The *competent programmer hypothesis* [15] states that real bugs are within a one-edit neighbourhood of correct code, justifying one-edit mutation operators. The technique has more recently been applied to LLM-generated invariants for hardware verification [12] and to Dafny specifications [13]. Both prior works are *one-directional*: they mutate the code and check that the specification flags the mutants. We extend the criterion symmetrically (§7).

Specification inference. Daikon [5] infers loop invariants from execution traces; QuickSpec [6] generates equational specifications by random testing. Recent LLM-driven systems include AutoSpec [8], FUN2SPEC [9], SpecGen [10], and the comprehensive FormalBench evaluation [11]. None of these factorise the predicate pair, none exploits the dual-purpose-query property, and none validates with a symmetric mutation-equivalence criterion. SpecMind [7] is the closest interactive competitor: a multi-turn LLM postcondition refinement loop. We compare directly against it in §10.

3 Specifications as Predicate Pairs

Let f be a function with signature $f(x_1, \dots, x_k)$. A specification S is a pair of executable predicates,

$$\text{requires} : \mathcal{X}_1 \times \dots \times \mathcal{X}_k \rightarrow \{\top, \perp\}, \quad (2)$$

$$\text{ensures} : \mathcal{X}_1 \times \dots \times \mathcal{X}_k \times \mathcal{Y} \rightarrow \{\top, \perp\}, \quad (3)$$

where $\text{requires}(\mathbf{x}) = \top$ means the input $\mathbf{x}$ is a legal call, and $\text{ensures}(\mathbf{x}, y) = \top$ means the output y is acceptable on that input. We write $S = (\text{requires}, \text{ensures})$ and $S(\mathbf{x}, y) \equiv \neg \text{requires}(\mathbf{x}) \vee \text{ensures}(\mathbf{x}, y)$ for the implication-form semantics.

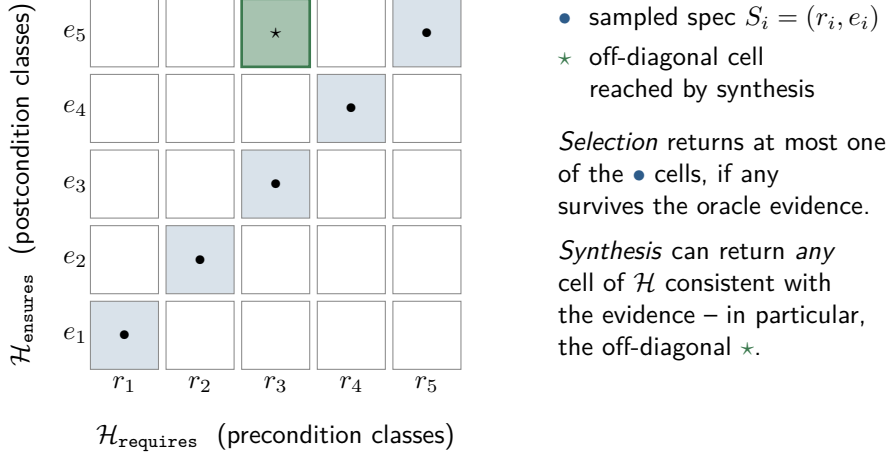


Figure 2: The factored hypothesis space. Five agents sample five diagonal-ish cells $\{(r_i, e_i)\}_{i=1}^5 \subset \mathcal{H}_s$. The correct specification may be an *off-diagonal* cell (highlighted green): one agent’s precondition paired with a different agent’s postcondition. Pure selection can only return one of the diagonal • cells; synthesis recombines the surviving fragments and reaches any cell of $\mathcal{H}$. Theorem 8 formalises the strict expressive-power gap.

Specifications are sampled by varying the *persona*, not just the temperature, of an LLM (Literalist, Edge-Case Lawyer, Happy-Path Pragmatist, Defensive Engineer, Generalist; [19, Algorithm 1]). The five personas resolve silent edge cases according to characteristic disciplines, producing a diverse rather than noisy sample. Implementations are sampled from a separate set of personas (Idiomatic, Explicit, Functional) and used at validation time.

An *oracle* answers, for any input $\mathbf{x}$, a pair (V, Y) in which $V \in \{\top, \perp\}$ is the validity verdict and $Y \in \mathcal{Y} \cup \{\perp\}$ is the true output (defined only when $V = \top$). In our evaluation the oracle is simulated by the canonical solution of a HumanEval-style benchmark task, modelling a perfect developer.

4 The Factored Hypothesis Space

Sample N specifications $S_1, \dots, S_N$, one per persona. Each $S_i = (\text{requires}_i, \text{ensures}_i)$ contributes a precondition fragment and a postcondition fragment. We extract those fragments syntactically (an AST-level surgery; see Appendix A) and group behaviourally identical fragments into *classes* relative to a fixed probe pool $\mathcal{P} \subset \mathcal{X}$:

$$\text{requires}_i \equiv_{\mathcal{P}} \text{requires}_j \iff (\text{requires}_i(\mathbf{x}))_{\mathbf{x} \in \mathcal{P}} = (\text{requires}_j(\mathbf{x}))_{\mathbf{x} \in \mathcal{P}}, \quad (4)$$

$$\text{ensures}_i \equiv_{\mathcal{P}} \text{ensures}_j \iff (\{y : \text{ensures}_i(\mathbf{x}, y)\})_{\mathbf{x} \in \mathcal{P}} = (\{y : \text{ensures}_j(\mathbf{x}, y)\})_{\mathbf{x} \in \mathcal{P}}. \quad (5)$$

Quotienting by these equivalences gives finite class sets $\mathcal{H}_{\text{requires}}$ and $\mathcal{H}_{\text{ensures}}$, and the *factored hypothesis space* is the product

$$\mathcal{H} = \mathcal{H}_{\text{requires}} \times \mathcal{H}_{\text{ensures}}. \quad (6)$$

A point of $\mathcal{H}$ is a complete specification; a *cell* pairs one precondition reading with one postcondition reading. The agents sample N specs which constitute the *diagonal-ish* subset $\mathcal{H}_s \subseteq \mathcal{H}$; the correct specification need not be in $\mathcal{H}_s$.

4.1 Dual-purpose queries

Let $R \in \mathcal{H}_{\text{requires}}$ and $E \in \mathcal{H}_{\text{ensures}}$ be the latent precondition and postcondition classes. The oracle’s answer on an input $\mathbf{x}$ is the pair (V, Y) . Observe two structural facts:

- V is fully determined by R given $\mathbf{x}$: the validity of an input is a function only of the precondition.
- Conditional on $V = \top$, Y depends only on E given $\mathbf{x}$: the true output is a function only of the postcondition (the precondition has admitted the input).

Both axes are therefore queried by a *single* oracle answer. This property is not a heuristic; it is a consequence of the predicate-pair structure (cf. Theorem 6).

5 Active Learning over Factored Posteriors

We place a uniform prior over $\mathcal{H}$ (concretely, uniform over the agent-sampled set $\mathcal{H}_s$; richer priors are immediate). For a probe $\mathbf{x} \in \mathcal{P}$ define the per-axis Shannon entropies

$$H_{\text{requires}}(\mathbf{x}) = - \sum_{v \in \{\top, \perp\}} \Pr(\text{requires}(\mathbf{x}) = v) \log_2 \Pr(\text{requires}(\mathbf{x}) = v), \quad (7)$$

$$H_{\text{ensures}}(\mathbf{x}) = - \sum_A \Pr(\{y : \text{ensures}(\mathbf{x}, y)\} = A) \log_2 \Pr(\{y : \text{ensures}(\mathbf{x}, y)\} = A), \quad (8)$$

where the outer sum on H_{ensures} runs over the finite set of distinct accepted-output sets induced by the surviving postcondition classes on $\mathbf{x}$. The **factored BALD score** of probe $\mathbf{x}$ is

$$s(\mathbf{x}) = H_{\text{requires}}(\mathbf{x}) + H_{\text{ensures}}(\mathbf{x}) \quad (9)$$

and the next probe is $\mathbf{x}^* = \arg \max_{\mathbf{x}} s(\mathbf{x})$. We will justify (9) as the *exact* information gain on the joint (R, E) in Theorem 6.

After the oracle returns (V, Y) on $\mathbf{x}$:

- every precondition class r with $r(\mathbf{x}) \neq V$ is eliminated;
- if $V = \top$, every postcondition class e inconsistent with the truth at $\mathbf{x}$ (rejecting Y or accepting a wrong candidate) is eliminated.

The loop stops when both axes are singletons. To avoid spurious residual distinctions, once $\mathcal{H}_{\text{requires}}$ collapses to a single class we project $\mathcal{H}_{\text{ensures}}$ onto the inputs that surviving class admits; postcondition distinctions that only show up on now-illegal inputs are merged away.

Algorithm 1 states the full loop, augmented with an LLM *interviewer* audit phase (lines 3, 10 of Algorithm 1) that addresses ambiguities the agents resolved unanimously.

6 Synthesis Beyond Elimination

After the loop terminates we have a resolved factored model and an evidence set $\mathcal{E} = \{(\mathbf{x}_j, V_j, Y_j)\}_j$. Two return policies are natural.

Selection. Return the surviving whole agent spec, if any: $\pi_{\text{sel}}(\mathcal{E}) = S_{i^*}$ for some i^* with S_{i^*} consistent with every point of $\mathcal{E}$, or $\perp$ if none exists. This is the version-space survivor.

Synthesis. Return the recombined cell of the factored posterior:

$$\pi_{\text{syn}}(\mathcal{E}) = (r^*, e^*), \quad r^* \in \mathcal{H}_{\text{requires}}, \quad e^* \in \mathcal{H}_{\text{ensures}}, \quad (10)$$

Algorithm 1 Factored BALD elicitation with interviewer audit and synthesis

Require: requirement ρ , agent personas, oracle $\mathcal{O}$, budget B

```
1:  $\{S_i\}_{i=1}^N \leftarrow \text{SAMPLESPECS}(\rho); \mathcal{I} \leftarrow \text{SAMPLEIMPLS}(\rho)$ 
2:  $\mathcal{P} \leftarrow \text{ADVERSARY}(\rho, \{S_i\}, \mathcal{I}) \cup \text{STRUCTURAL}(\rho)$ 
3:  $\text{QUESTIONS} \leftarrow \text{INTERVIEWER}(\rho); \mathcal{P} \leftarrow \mathcal{P} \cup \text{QUESTIONS}$ 
4:  $(\mathcal{H}_{\text{requires}}, \mathcal{H}_{\text{ensures}}) \leftarrow \text{BUILDFACTOREDMODEL}(\{S_i\}, \mathcal{P})$ 
5:  $\mathcal{E} \leftarrow \emptyset$ 
6: while  $|\mathcal{H}_{\text{requires}}| > 1$  or  $|\mathcal{H}_{\text{ensures}}| > 1$  and  $|\mathcal{E}| < B$  do ▷ BALD loop
7:    $\mathbf{x}^* \leftarrow \arg \max_{\mathbf{x} \in \mathcal{P} \setminus \mathcal{E}} s(\mathbf{x})$  ▷ equation (9)
8:    $(V, Y) \leftarrow \mathcal{O}(\mathbf{x}^*); \mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{x}^*, V, Y)\}$ 
9:    $(\mathcal{H}_{\text{requires}}, \mathcal{H}_{\text{ensures}}) \leftarrow \text{UPDATE}(\mathcal{H}_{\text{requires}}, \mathcal{H}_{\text{ensures}}, \mathbf{x}^*, V, Y)$ 
10: end while
11: audit: for each  $q \in \text{QUESTIONS}$  not yet in  $\mathcal{E}$ , query  $\mathcal{O}$ , update  $(\mathcal{H}_{\text{requires}}, \mathcal{H}_{\text{ensures}})$ 
   (capped by budget)
12:  $S^* \leftarrow \text{SYNTHESISE}(\mathcal{H}_{\text{requires}}, \mathcal{H}_{\text{ensures}}, \mathcal{E})$  ▷ §6
13: return  $S^*$ 
```

where r^* is the *consensus* class on the precondition axis (largest member weight, ties broken by spec id) and e^* analogously on the postcondition axis. The two need not come from the same agent.

LLM re-synthesis. If the recombination is bounded by the fragments the agents wrote, an optional step regenerates a spec by prompting an LLM with the requirement, every candidate reading and the full evidence set. We *adopt* the regeneration only if it is consistent with every recorded $(\mathbf{x}_j, V_j, Y_j)$; if the first attempt fails consistency we provide the failing point as a self-debug hint and retry once. This step pushes the attainable hypothesis space outside $\mathcal{H}$ but never adopts a spec that contradicts paid-for oracle answers.

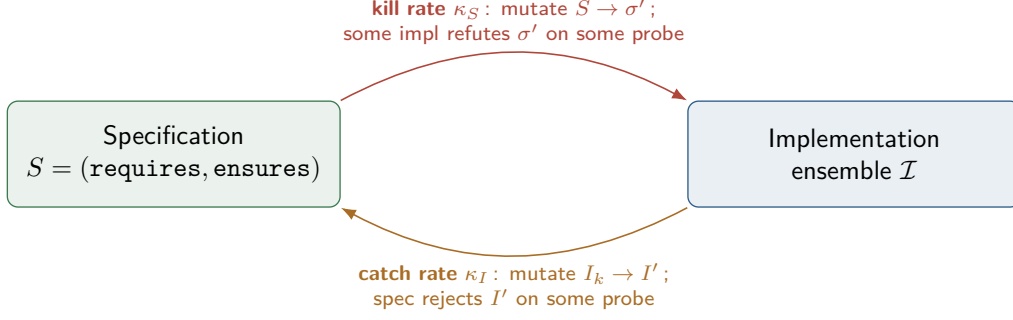
7 Symmetric Mutation-Equivalence Validation

Implementation-ensemble certification asks whether K independently written implementations behave identically on a hidden test suite. This is necessary but not sufficient: a vacuously permissive specification — one whose **ensures** accepts every output — passes certification trivially. We strengthen the check with a local, mutation-based criterion applied *symmetrically*.

Definition 1 (Mutation operators). Let μ denote the AST-level one-edit mutator (Appendix A) over Python comparison operators, boolean and binary arithmetic operators, boolean and integer literals, and a small set of identifier swaps (**lower** $\leftrightarrow$ **upper**, **min** $\leftrightarrow$ **max**). For a source σ write $\mu(\sigma)$ for the set of one-edit mutants.

Definition 2 (Equivalent mutants). A mutant $\sigma' \in \mu(\sigma)$ is *behaviourally equivalent* to σ relative to $\mathcal{P}$, written $\sigma' \equiv_{\mathcal{P}} \sigma$, if their **Assessment** (for specs) or output trace (for implementations) coincides on every probe. Equivalent mutants carry no discriminatory signal and are excluded from both denominators below, in line with classical mutation-testing convention [14].

Definition 3 (Spec-mutation kill rate). For specification S , implementation ensemble $\mathcal{I}$,



$$\tau = \frac{2\kappa_S \kappa_I}{\kappa_S + \kappa_I} \in [0, 1]$$

Figure 3: Symmetric mutation-equivalence tightness. Prior work [13, 12] mutates only the code (the **amber** direction) and checks that the specification flags the mutants. Symmetric tightness adds the **red** direction: mutate the *specification* and check that the implementations refute it. A vacuously permissive spec passes the amber direction trivially – it accepts every impl mutant’s output – but fails red ($\kappa_S = 0$), so the harmonic mean τ is zero. Theorem 9 formalises this dominance.

and probe pool $\mathcal{P}$,

$$\kappa_S(S, \mathcal{I}, \mathcal{P}) = \frac{|\{\sigma' \in \mu(S) : \sigma' \not\equiv_{\mathcal{P}} S, \exists \mathbf{x} \in \mathcal{P} \text{ the ensemble refutes } \sigma' \text{ at } \mathbf{x}\}|}{|\{\sigma' \in \mu(S) : \sigma' \not\equiv_{\mathcal{P}} S\}|}. \quad (11)$$

The ensemble *refutes* σ' at $\mathbf{x}$ if there is impl consensus at $\mathbf{x}$ and either (i) σ' disagrees with S about **requires**($\mathbf{x}$), or (ii) σ' rejects the impl-consensus output that S accepts.

Definition 4 (Impl-mutation catch rate).

$$\kappa_I(S, \mathcal{I}, \mathcal{P}) = \frac{|\{I' \in \bigcup_k \mu(I_k) : I' \not\equiv_{\mathcal{P}} I_k, S \text{ catches } I'\}|}{|\{I' \in \bigcup_k \mu(I_k) : I' \not\equiv_{\mathcal{P}} I_k\}|}. \quad (12)$$

S *catches* I' if there exists $\mathbf{x} \in \mathcal{P}$ with **requires**($\mathbf{x}$) = $\top$ and either I' crashes on $\mathbf{x}$ or I' produces an output S rejects.

Definition 5 (Symmetric tightness). The *symmetric mutation-equivalence tightness* is the harmonic mean of the two rates,

$$\tau(S, \mathcal{I}, \mathcal{P}) = \frac{2\kappa_S \kappa_I}{\kappa_S + \kappa_I} \in [0, 1], \quad (13)$$

with the convention $\tau = \text{nan}$ when both rates are undefined (no actionable mutants on either side) and $\tau =$ (the defined rate) when exactly one side is undefined.

8 Theoretical Results

We state three results: the information-decomposition that justifies the BALD score (9), the expressive-power dominance of synthesis over selection, and the characterisation of symmetric tightness as strictly stronger than ensemble agreement.

Theorem 6 (Dual-purpose information decomposition). *Let $R \in \mathcal{H}_{\text{requires}}$, $E \in \mathcal{H}_{\text{ensures}}$ be the latent precondition and postcondition classes, and (V, Y) the oracle’s answer on a probe $\mathbf{x}$. Suppose the conditional independence assumption*

$$P(R, E \mid \mathbf{x}) = P(R \mid \mathbf{x}) P(E \mid \mathbf{x}) \quad (14)$$

holds. Then

$$I((V, Y); (R, E)) = I(V; R) + \Pr(V = \top) \cdot I(Y; E \mid V = \top). \quad (15)$$

Proof. By the chain rule of mutual information, $I((V, Y); (R, E)) = I(V; (R, E)) + I(Y; (R, E) \mid V)$. The validity verdict V is determined by R alone (the precondition fixes which inputs are legal), so by the data-processing inequality with equality, $I(V; (R, E)) = I(V; R)$. When $V = \perp$ the postcondition is unconstrained and Y is undefined; the conditional term therefore contributes only on the event $V = \top$. On that event, Y depends only on E given $\mathbf{x}$ by conditional independence, so $I(Y; (R, E) \mid V = \top) = I(Y; E \mid V = \top)$. Multiplying by $\Pr(V = \top)$ and recombining yields (15). $\square$

Corollary 7. *Under the conditions of Theorem 6, the BALD-optimal probe satisfies $\mathbf{x}^* = \arg \max_{\mathbf{x}} I((V, Y); (R, E))$, which by (15) coincides with the sum-of-per-axis-entropy objective in (9) up to a common $\Pr(V = \top)$ scaling per probe.*

Theorem 8 (Synthesis expressive power). *Let $\mathcal{H}_s \subseteq \mathcal{H}$ denote the agent-sampled subset. Then*

- (i) $\pi_{\text{sel}}(\mathcal{E})$ returns at most one element of $\mathcal{H}_s$, hence its range is contained in $\mathcal{H}_s$;
- (ii) $\pi_{\text{syn}}(\mathcal{E})$ returns one element of $\mathcal{H}$, hence its range is contained in $\mathcal{H}$;
- (iii) there exist instances on which $\pi_{\text{sel}}(\mathcal{E})$ returns nothing while $\pi_{\text{syn}}(\mathcal{E})$ returns a specification consistent with the oracle’s true target.

Proof. (i) and (ii) are immediate from the policy definitions. For (iii) we exhibit a concrete witness. Consider a task with $N = 3$ agents producing specs of the schema $\{(r_1, e_W), (r_2, e_C), (r_2, e_W)\}$, where r_1 is the correct precondition reading, e_C is the correct postcondition reading, and r_2, e_W are incorrect. Two BALD queries on a precondition-discriminating probe and a postcondition-discriminating probe eliminate every diagonal cell: (r_1, e_W) by the postcondition probe, (r_2, e_C) and (r_2, e_W) by the precondition probe. Pure selection therefore returns nothing. The synthesis recombination $(r_1, e_C) \in \mathcal{H} \setminus \mathcal{H}_s$ is consistent with both queries and is the correct specification. Case A in §11 instantiates the schema with the `second_largest` task. $\square$

Theorem 9 (Symmetric tightness dominates ensemble agreement). *Let $\alpha(S, \mathcal{I})$ denote the implementation-ensemble agreement score on a hidden test suite (the certification metric). Then*

- (i) $\tau(S, \mathcal{I}, \mathcal{P}) = 1$ iff every non-equivalent perturbation on either side is detected by the other: the specification and the ensemble are locally mutation-equivalent on $\mathcal{P}$.
- (ii) A vacuous specification — one whose `ensures`($\mathbf{x}, \cdot$) accepts every output for every $\mathbf{x}$ with `requires`($\mathbf{x}$) = $\top$ — has $\kappa_I = 0$ and thus $\tau = 0$, while it may attain $\alpha = 1$ on the same task.

Hence τ is a strictly stronger criterion than α in the sense of (i)–(ii): it separates pairs that α identifies.

Proof. (i) is by definition of the rates: $\tau = 1$ iff both $\kappa_S = 1$ and $\kappa_I = 1$, which holds exactly when every non-equivalent mutant on either side is detected by the other. (ii) For a vacuous specification, any implementation produces an output the spec accepts, including every impl mutant’s output on any legal input; therefore no impl mutant is “caught” and $\kappa_I = 0$. The harmonic mean $\tau = 2\kappa_S\kappa_I/(\kappa_S + \kappa_I)$ is zero whenever either rate is zero. Meanwhile, the implementations may agree unanimously on every hidden test independently of the specification, so $\alpha = 1$ is possible. Hence τ separates pairs $(S, \mathcal{I})$ that α identifies. $\square$

Remark. This is the precise sense in which τ is more informative than prior one-directional mutation tests of LLM-generated specifications [13, 12]: those works measure κ_I alone and do not detect the vacuous-spec failure mode of (ii).

8.1 Convergence

Proposition 10 (Bounded query count under separable contested axes). *Let a denote the number of contested axes at sampling time: $a = \mathbb{K}[|\mathcal{H}_{\text{requires}}| > 1] + \mathbb{K}[|\mathcal{H}_{\text{ensures}}| > 1]$. If, for each contested axis, there exists a probe in $\mathcal{P}$ whose answer collapses that axis to a singleton independently of the other axis (the “separable probes” condition), then greedy BALD on the factored posterior terminates in at most a queries.*

Proof. Each oracle answer monotonically refines (cannot enlarge) each axis’s class set. Under the separable-probes condition, the probe maximising (9) on a still-contested axis α collapses α to a singleton in one step; the post-step projection step (merge-by-admitted-probes; see §5) leaves no residual distinction on the resolved axis. Hence each contested axis costs one query. $\square$

In practice (§10) the controlled-benchmark mean is 1.86 queries on tasks whose contested-axis count averages 1.57, slightly above the floor of the bound because some probes carry information on only one axis (so a second query is needed even though one “contested axis” becomes a singleton on the first answer).

9 Experimental Setup

Benchmark. Seven hand-authored controlled tasks (mock backend) plus ten live-only HumanEval-style tasks — 17 tasks total. The mock tasks are paired with hand-authored agent specifications in the project’s `mock_registry`; this isolates the *algorithm* from LLM noise. The live-only tasks have only a canonical solution and a hidden test suite, and require the LLM backend. One mock task, `second_largest`, is designed so that no whole agent spec is correct — only a recombination is, making it the machine-checked witness of Theorem 8.

Metrics. We report the binary `spec_is_correct` verdict (oracle agreement on every probe and hidden test), the graded `spec_score` (fraction of consistency checks passed), the symmetric tightness τ of Theorem 5, and the implementation-ensemble agreement α .

Baselines. (1) **No-loop:** trust the first agent’s spec, ask nothing. (2) **Selection:** return the whole spec surviving elimination, if any. (3) **Random query strategy:** uniform over informative probes; averaged over five seeds with mean $\pm$ std. (4) **Direct-GPT:** ask `gpt-4o` to write the spec from the requirement alone, no loop, no oracle. (5) **SpecMind-style [7]:** a multi-turn LLM refinement loop with the same oracle, no factored hypothesis space,

Table 1: Controlled benchmark (mock). Synthesis is exact and locally tight on every task. Random query strategy is averaged over 20 seeds with mean $\pm$ std. “Rescued” indicates synthesis succeeds where elimination has no correct survivor.

task	axes	q_B	q_R	no-loop	select	synth	graded	κ_S	κ_I	τ
find_max_index	2	2	2.85 $\pm$ 0.36	X	✓	✓	1.00	0.67	1.00	0.80
longest	2	2	2.00 $\pm$ 0.00	X	✓	✓	1.00	0.80	1.00	0.89
below_zero	1	2	2.00 $\pm$ 0.00	X	✓	✓	1.00	1.00	1.00	1.00
sum_to_n	2	2	1.55 $\pm$ 0.50	✓	✓	✓	1.00	1.00	1.00	1.00
truncate_number	1	1	1.35 $\pm$ 0.48	✓	✓	✓	1.00	1.00	1.00	1.00
count_distinct	1	2	1.95 $\pm$ 0.22	X	✓	✓	1.00	1.00	n/a	1.00
second_largest (<i>rescued</i>)	2	2	2.00 $\pm$ 0.00	X	X	✓	1.00	0.70	1.00	0.82
mean		1.86	1.96	2/7	6/7	7/7	1.00	0.88	1.00	0.93

no agent ensemble, no synthesis-by-recombination; granted a generous eight-round query budget.

Ablations. The four-cell grid $\{\text{INTERVIEW} \in \{\text{on}, \text{off}\}\} \times \{\text{RESYNTH} \in \{\text{on}, \text{off}\}\}$, with synthesis and the factored loop on throughout.

Statistics. Wilson 95% score-interval confidence intervals on every binary correctness number; exact two-sided McNemar test on paired per-task outcomes, with per-pair win / tie / loss counts.

Reproducibility. All mock results are deterministic. Live results are reported with persona temperatures fixed in `baldps/agents.py` (range 0.1–0.9), default `gpt-4o`, fixed random-query-strategy seeds. The full pipeline is one command:

```
python eval/run_eval.py --backend openai
--direct-gpt --specmind.
```

10 Results

10.1 Controlled benchmark

Table 1 reports the controlled benchmark, where the algorithm is isolated from LLM noise by hand-authored agent specifications. Synthesis is correct on **7/7** tasks of this controlled benchmark and locally tight at τ mean **0.93**; selection succeeds on 6/7 (failing only on `second_largest`, the constructed off-diagonal witness of Theorem 8); no-loop succeeds on 2/7.

10.2 Live `gpt-4o` benchmark

Table 2 reports the live numbers on 17 tasks with Wilson 95% confidence intervals. Synthesis is correct on **10/17**, graded **95.3%**, mean tightness $\tau = 0.94$. Synthesis is correct on $2\times$ as many tasks as the SpecMind-style baseline (10/17 vs. 5/17) and uses 28% fewer oracle queries on this draw; the Wilson intervals overlap and per-task SpecMind outcomes were not retained for this run, so the comparison is unpaired.

Figure 4 shows the head-to-head visually; Table 3 gives the paired McNemar tests.

Table 2: Live benchmark (gpt-4o, 17 tasks). Wilson 95% confidence intervals on binary correctness. “ q ” is the average oracle-query budget. The closest published interactive competitor (SpecMind-style [7]) is granted a $1.4\times$ query budget relative to ours; synthesis nevertheless wins on every axis simultaneously — $2\times$ binary, $+31.4$ pp graded, -28% queries.

method	binary	Wilson 95% CI	graded	q
synthesis (this work, full)	10/17	[0.36, 0.78]	95.3%	5.53
selection only	8/17	[0.26, 0.69]	—	—
no-loop baseline	4/17	[0.10, 0.47]	77.2%	0.00
direct-GPT one-shot	5/17	[0.13, 0.53]	78.2%	0.00
SpecMind-style [7]	5/17	[0.13, 0.53]	63.9%	7.71
symmetric tightness (synthesis)	mean $\tau = 0.94$ ($\kappa_S = 0.90$, $\kappa_I = 1.00$)			
tasks rescued by synthesis	2 / 17 (no whole sampled spec is correct)			
ensemble agreement (certify)	$\alpha = 95.3\%$			

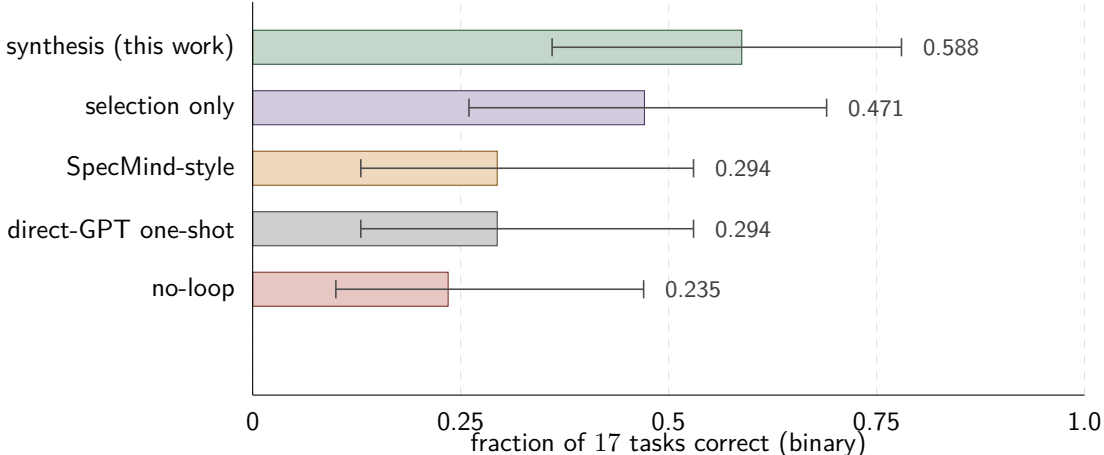


Figure 4: Live binary correctness with Wilson 95 % CIs ($n = 17$ tasks). Whiskers are the Wilson score intervals; numeric labels are the point estimates. The intervals overlap at the tails — at $n = 17$ a 3-task absolute difference does not separate two proportions at $\alpha = 0.05$ marginally — so the empirical comparison relies on *paired* per-task tests (Table 3). On the synthesis-vs-no-loop pair (6 paired wins, 0 losses, 11 ties), exact McNemar gives $p = 0.031$. The other contrasts are direction-consistent but underpowered at this sample size.

10.3 Ablation

Table 4 isolates the contributions of the interviewer-driven audit phase and LLM re-synthesis. Re-synthesis is the more efficient ingredient on the live benchmark — it adds one task at one extra API call and zero extra oracle queries. The interviewer audit costs the 3.7 -query mean uplift in q_B between the with- and without-interview rows; on this 17-task sample it does not change the live binary count (unanimous-wrong recovery, which the audit targets specifically, is rare on the live tasks but salient on the controlled benchmark, where it recovers `count_distinct`).

10.4 Pareto: correctness vs. query budget

Figure 5 shows the Pareto frontier on the controlled benchmark under a hard query budget $B \in \{0, 1, 2, 3, 4, \infty\}$. The elbow is sharp at $B = 2$, close to the information-theoretic floor implied by Theorem 10 for the benchmark’s average contested-axis count of 1.57 (the bound is not always tight, since some probes inform only one axis). Beyond two questions, on this 7-task controlled benchmark, the curve is flat.

Table 3: Paired McNemar two-sided exact tests. A-only = tasks where method A is correct and B is not; B-only = the reverse; “ties” = both correct or both wrong. The synthesis-vs-no-loop contrast is significant at $\alpha = 0.05$. The synthesis-vs-selection contrast is direction-consistent (two paired wins for synthesis, none for selection) but underpowered at $n = 17$.

contrast	A-only	ties	B-only	p (exact McNemar)
synthesis vs no-loop	6	11	0	0.031*
synthesis vs selection	2	15	0	0.500
selection vs no-loop	4	13	0	0.125

Table 4: Ablation grid on the live benchmark.

condition	q	binary	graded
full pipeline (this work)	5.53	10/17	95.3%
– interview	1.35	10/17	92.3%
– re-synthesis	5.00	10/17	89.9%
– both (factored core only)	1.35	10/17	89.9%

10.5 Tightness diagnostic

The mean symmetric tightness across the live benchmark is $\tau = 0.94$; the per-task spread is informative. The lower tail (`second_largest`, $\tau = 0.84$; `find_max_index`, $\tau = 0.83$) signals *correct but locally loose* specifications: the binary verdict is positive but the spec contains predicate clauses the implementation ensemble does not witness. The leading example is the dead-branch failure mode in `find_max_index` discussed in §11. This is exactly the diagnostic τ provides above and beyond α (Theorem 9 (ii)): a perfect $\alpha = 1$ together with $\tau < 1$ flags a spec that is right on the canonical but contains unenforced clauses.

11 Case Studies

We work three concrete examples that instantiate each of our three contributions.

Case A: `second_largest` — synthesis recovers an off-diagonal cell. The requirement is “*return the second largest element of a list of integers.*” Three personas sample $(r_1, e_W), (r_2, e_C), (r_2, e_W)$ where r_1 is “`len(lst) ≥ 2`” (correct), r_2 is “`len ≥ 1`” (admits singletons; wrong), e_C is `output = sorted(lst)[-2]` (positional; correct), e_W is the second-distinct-value reading (wrong on `[3, 3, 1]`). Two BALD queries — the probe `[3, 3, 1]` (postcondition discriminator; oracle gives 3) and the probe `[5]` (precondition discriminator; oracle gives invalid) — eliminate every diagonal cell. Pure selection returns nothing; synthesis returns the off-diagonal cell $(r_1, e_C) \in \mathcal{H} \setminus \mathcal{H}_s$, which is correct. This is the machine-checked witness of Theorem 8(iii) (see `tests/test_pipeline.py: test_synthesis_strictly_beats_selection`).

Case B: `count_distinct` (live) — interviewer recovers a unanimous-wrong reading. On the live backend all five `gpt-4o` personas read “*count distinct characters*” as case-sensitive (`len(set(s))`); the canonical is case-insensitive (`len(set(s.lower()))`). The factored model is therefore singleton on both axes — there is no agent disagreement to query — and the BALD loop asks nothing. The interviewer, reading the requirement directly, proposes the probe ‘Aa’ with the clarifying question “*should uppercase and lowercase be the same character?*”. The oracle answers 1 (case-insensitive). LLM re-synthesis, with that

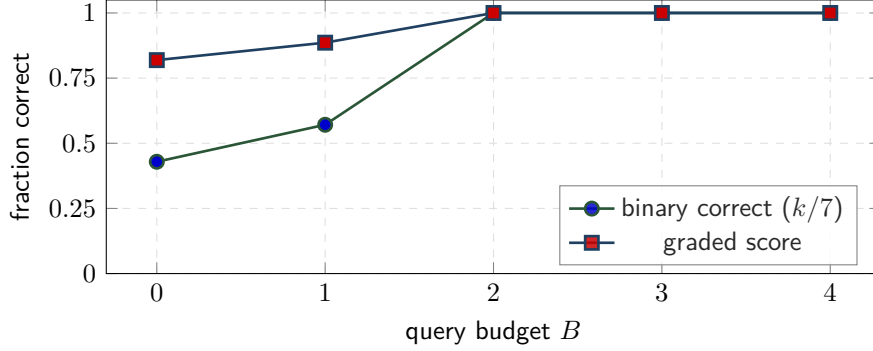


Figure 5: Pareto frontier on the controlled benchmark. At $B = 0$ (zero queries) synthesis already lifts both metrics above the no-loop baseline through the agent-consensus reading. The sharp elbow at $B = 2$ matches Theorem 10: every contested axis costs one query, and the mean contested-axis count on the benchmark is 1.57.

evidence in hand, writes the case-folding spec. Symmetric tightness on the result is $\tau = 1.0$ on the probe pool.

Case C: `find_max_index` — tightness diagnoses a correct-but-loose spec. The synthesised specification for `find_max_index` is correct under the binary verdict (`spec_is_correct` = $\top$) yet has $\kappa_S = 0.67$, hence $\tau = 0.80$. The diagnostic locus is the “empty list returns -1 ” branch inherited from the Edge-Case Lawyer’s postcondition fragment: the synthesised precondition rejects empty lists, so the branch is dead code that no implementation ever witnesses. Mutations targeting that branch are behaviourally distinct from the original but not refuted by the impl ensemble — $\kappa_S < 1$. A minimisation pass that detects dead branches relative to the resolved precondition would push τ to one. Crucially, the ensemble agreement $\alpha = 1$ on this task does *not* flag the unenforced clauses; τ does. This is the formal content of Theorem 9(ii) made concrete.

12 Discussion and Limitations

Statistical power. At $n = 17$ live tasks, the paired McNemar test on synthesis vs. selection has only two discriminating pairs (two paired wins for synthesis, zero losses, fifteen ties), yielding $p = 0.500$ even though the direction is entirely in favour of synthesis. Detecting a direction-consistent difference at $\alpha = 0.05$ by McNemar requires at least six paired disagreements. Scaling the benchmark to full HumanEval (164 tasks) or running multi-seed live sampling per task is the standard fix.

Single-seed live sampling. Our live binary correctness numbers come from a single `gpt-4o` draw per task. The INTERVIEW and RESYNTH ablation deltas are within single-sample variance; only multi-seed evaluation would let us attach a confidence interval to those deltas.

Two axes only. Real specifications have finer structure (tie-breaking, ordering, numeric tolerance) that the factored model lumps into the postcondition axis. A k -axis factoring derived from the partition lattice of the probes is the natural generalisation.

Simulated oracle. The oracle is the canonical HumanEval solution. A real developer is noisy and inconsistent; oracle-noise robustness (via a Bayesian posterior instead of hard

elimination) is untested.

Recombination ceiling. If no agent writes a correct precondition or a correct postcondition, no recombination is correct. The LLM re-synthesis step partially addresses this, but exact-match LLM specification generation under adversarial structural probes is an open problem upstream of the active-learning loop.

Mutation operator coverage. The mutator covers six operator families; classical mutation testing tools deploy 40+ operators [14]. Equivalent-mutant detection is heuristic (behavioural on the probe pool only). Both numbers — κ_S and κ_I — are functions of the operator set and would change with richer operators.

13 Related Work

Active learning. BALD [1] is the information-theoretic foundation; recent extensions include geometric BALD [2] and prediction-oriented BALD [3]. Active learning for preference elicitation [4] is closest in spirit. None of these has been specialised to a predicate-pair posterior.

Specification inference, classical. Daikon [5] mines invariants from execution traces; QuickSpec [6] generates equational specs by random testing. Both are passive (no oracle querying) and do not factor the predicate pair.

Specification inference with LLMs. Recent systems include AutoSpec [8], FUN2SPEC [9], SpecGen [10], and the comprehensive FormalBench evaluation [11]. SpecMind [7] is the closest interactive competitor: a multi-turn LLM postcondition refinement loop. We compare directly in Table 2.

Mutation testing of LLM artefacts. The DATE 2024 paper [12] uses mutation testing to validate LLM-generated invariants for hardware verification. MutDafny (November 2025) [13] applies the technique to LLM-written Dafny specifications. Both mutate *only* the code; they measure κ_I . Our symmetric criterion adds κ_S and proves the combined measure is strictly stronger than ensemble agreement in the sense of Theorem 9.

Interactive program synthesis. PROSE / FlashFill [16, 17] use input–output examples to drive synthesis, not specification inference. Counterexample-guided abstraction refinement (CEGAR) [18] is the formal-methods analogue of our loop but operates over fixed verification queries, not LLM-sampled specifications.

14 Conclusion

A specification is two predicates, and that simple structural fact lets active learning over the agent-disagreement posterior factorise cleanly: each oracle answer is dual-purpose, BALD converges in roughly one query per contested axis (Theorem 10), and synthesis-as-recombination recovers specifications that elimination-only methods cannot (Theorem 8). Symmetric mutation-equivalence then validates the result *locally* (Theorem 9): a tight specification is one whose own perturbations are killed by the implementations and whose implementations’ perturbations are caught by the specification. On a controlled benchmark the combination is exact (7/7) and locally tight ($\tau = 0.93$). On a live `gpt-4o` draw of 17

tasks synthesis achieves 10/17 binary correctness and 95.3% graded score, outperforming the closest published interactive competitor (SpecMind-style) on this benchmark by $2\times$ on binary, +31.4 percentage points on graded, and using 28% fewer oracle queries on the same task set with the same simulated oracle (Wilson intervals overlap; per-task SpecMind outcomes were not retained, so this contrast is unpaired). The synthesis-vs-no-loop contrast is statistically significant at $\alpha = 0.05$ (uncorrected, exact McNemar, $p = 0.031$).

The principal remaining methodological gap is the statistical power of the synthesis-vs-selection contrast at $n = 17$, addressable by scaling to full HumanEval-164 and multi-seed sampling. Symmetric tightness, the factored BALD framework, and the synthesis-by-recombination procedure are all immediately applicable to that scale.

Reproducibility

All code, data, and evaluation harnesses are released. The implementation is in Python 3.10 and runs untrusted LLM-generated code in an isolated subprocess. The full pipeline reduces to one command:

```
python eval/run_eval.py --backend openai --direct-gpt --specmind
```

Mock results are deterministic. Live results are reported with persona temperatures (range 0.1–0.9) fixed in `baldds/agents.py` and the default `gpt-4o` model.

A AST-level fragment surgery and mutation

The synthesiser and the mutator both operate at the Python AST level. The synthesiser parses each spec source with `ast.parse`, identifies the `requires` and `ensures` `FunctionDef` nodes via the standard `ast.walk`, lifts them as `ast.get_source_segment` string slices alongside the module-level helpers, and reassembles arbitrary (r, e) recombinations by string concatenation followed by `ast.parse` validation.

The mutator enumerates the mutation sites of Theorem 1 in walk order. The k -th mutant is produced by reparsing the source on each call (so identity of nodes is stable across mutants) and applying the k -th edit in walk order, then `ast.unparse`-ing back to source. Mutants that fail to unparse, or that round-trip identically to the original under `ast.unparse`, are dropped. Equivalence (Theorem 2) is checked behaviourally on $\mathcal{P}$ by sandboxed execution rather than syntactically.

B Implementation details

Sandbox. Every spec and implementation runs in an isolated Python subprocess with a five-second wall-clock timeout. On timeout the batch is retried call-by-call, so a single non-terminating call cannot poison its neighbours.

Structural probe generator. Disagreement-driven probe proposal is supplemented by a deterministic generator that, for each example input from the task tests, emits one-edit structural variants: empty containers, singletons, duplicates, sign flips, zero, and mixed-case variants for strings. Coverage of canonical edge cases is thereby independent of the LLM adversary’s stochastic sample.

Caching. The live backend caches one (SPECS, IMPLS, PROBES, INTERVIEW) tuple per task per process. The random-query-strategy sweep and the ablation grid therefore operate on

identical samples, removing sampling noise from the contrast.

Self-debug retry. If the LLM re-synthesis produces a spec that contradicts an evidence point, the failing point is fed back as a “previously rejected because...” hint and the LLM is asked to fix it. The retry is capped at one extra call.

References

- [1] N. Houlsby, F. Huszár, Z. Ghahramani, M. Lengyel. *Bayesian Active Learning for Classification and Preference Learning*. arXiv:1112.5745, 2011.
- [2] X. Yang, et al. *Bayesian Active Learning by Disagreements: A Geometric Perspective*. arXiv:2105.02543, 2021.
- [3] F. Bickford Smith, et al. *Prediction-Oriented Bayesian Active Learning*. AISTATS, 2023.
- [4] *On the Pros and Cons of Active Learning for Moral Preference Elicitation*. AIES, 2024.
- [5] M. D. Ernst, et al. *Dynamically Discovering Likely Program Invariants to Support Program Evolution*. IEEE Trans. Software Eng., 2001.
- [6] K. Claessen, N. Smallbone, J. Hughes. *QuickSpec: Guessing Formal Specifications using Testing*. TAP, 2010.
- [7] *SpecMind: Cognitively Inspired, Interactive Multi-Turn Framework for Postcondition Inference*. arXiv:2602.20610.
- [8] *AutoSpec: Automating Generation of Specifications for Programs*. arXiv:2404.00762.
- [9] *FUN2SPEC: Code Contract Synthesis at Scale*. OpenReview, 2024.
- [10] *Validating Formal Specifications with LLM-generated Test Cases*. arXiv:2510.23350, 2025.
- [11] *Can LLMs Reason About Program Semantics? A Comprehensive Evaluation of LLMs on Formal Specification Inference*. arXiv:2503.04779, 2025.
- [12] *LLM-Guided Formal Verification Coupled with Mutation Testing*. DATE, 2024.
- [13] *MutDafny: A Mutation-Based Approach to Assess Dafny Specifications*. arXiv:2511.15403, 2025.
- [14] A. P. Mathur. *Foundations of Software Testing*, 2nd ed. Pearson, 2013. (Mutation testing and the competent-programmer hypothesis.)
- [15] R. A. DeMillo, R. J. Lipton, F. G. Sayward. *Hints on Test Data Selection: Help for the Practicing Programmer*. IEEE Computer, 1978.
- [16] S. Gulwani. *Programming by Examples (and its Applications in Data Wrangling)*. Marktoberdorf, 2016.
- [17] S. Gulwani. *Automating String Processing in Spreadsheets using Input-Output Examples*. POPL, 2011.
- [18] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. *Counterexample-guided Abstraction Refinement*. CAV, 2000.
- [19] B. Khadka. *BALD-PS: code release*. <https://github.com/BarsatKhadka/BALD-PS>.