Making Excluded Capabilities Observable at a Receiving DSL Boundary

Anonymous Author(s)

Abstract

A small domain-specific language can deliberately omit dynamic evaluation, host interoperability, source-generating macros, runtime reflection, and dynamic module loading. Its compiler can still collapse an excluded-facility request and an ordinary misspelling into the same unresolved-name diagnostic. We present a producer-independent receiving boundary that reclassifies exact registered requests on one unresolved-identifier path and emits versioned human and machine-readable policy diagnostics. It does not infer intent or establish runtime confinement.

Study A finds family-correct diagnostics in 15/15 recognized family-by-context cells with zero control misfires and full structured-record conformance. Study B shows all 24 balanced capability mutants reaching the policy classifier with family-correct `L52xx` diagnostics. Study C provides contextual validation of the surrounding staged boundary, where removing the supplemental language package reduces first-draft acceptance by 41.7 points. Study D then measures downstream handling in 288 one-turn cells. Relative to generic `L5002` feedback, policy text increases post-repair `L52xx` absence by 13.9 percentage points with a 95 percent task-resampling interval of $[5.6, 23.6]$; all remaining policy events in this sample were substitutions into another registered family, and the aggregate contrast was driven by the template-assigned tasks. The oracle-correctness gain is positive but uncertain. Raw structured JSON has identical observed task-macro handling rates but is 15.3 points worse on oracle correctness with interval $[-27.8, -4.2]$. The results establish versioned policy classification, its measured reduction of repeated or substituted policy refusals in one-turn output for appended registry-recognizable requests, and the continuing need for semantic evidence.

Keywords: domain-specific languages, language models, capability policy, compiler diagnostics, code repair, mutation testing

1 Introduction

Model-produced source reaches a receiving toolchain as text. The producer may have emitted a clean program, prose around a program, foreign syntax, an unresolved name, an excluded-facility request, a runtime fault, or a program that executes but violates its task contract. An aggregate pass rate records the final disposition, but it hides the stage at which a defect first became observable and the information available to an operator or a repair loop at that point.

This loss of stage information is especially important when a language deliberately omits a facility. Absence from a language surface can be a design decision rather than an implementation gap. Yet ordinary name resolution does not necessarily preserve that distinction. If a compiler reports only that a name is unresolved, a deliberate request for a facility and a typographical error can collapse into the same observable event. The diagnostic is locally correct because neither spelling resolves. It is operationally incomplete because the receiving boundary knows more about one spelling than it reports.

We study this ambiguity in a small domain-specific language called L. L deliberately omits several facilities. In the base compiler R0, both `eval("x")` and the misspelling `evla("x")` receive `L5002`, meaning that the name is not bound. The first spelling names a facility that the boundary has deliberately excluded. The second spelling has no such registered meaning. A generic unresolved-name report makes these cases indistinguishable to a downstream consumer even though they call for different repair behavior.

The distinction matters when model-produced source arrives at a receiving boundary. A model may have inserted a familiar host-language idiom into a smaller language, misspelled a name, or omitted a local definition. When every case receives `L5002`, neither an operator nor a repair model can tell whether the producer requested an excluded family or merely misspelled, so a repair may rebind the name, substitute another excluded name, or rewrite the program without understanding why the request was refused.

The R0+B1 study gate extends that unresolved-reference path with a checked excluded-facility diagnostic registry. An exact registered request can receive `L5201`, while an unregistered spelling retains `L5002`. Resolved local names, strings, comments, unsupported syntax, and arbitrary similar text are not reclassified. The mechanism therefore preserves ordinary resolution as the first decision and introduces a policy distinction only after lookup fails at a supported classifier site.

This receiving-boundary placement is deliberate. The boundary does not require control over the producer, its decoding strategy, its prompt, or its internal representation. It examines persisted source using the same compiler path that will accept or reject that source. Producer-side constraints can reduce malformed output, but they cannot by themselves determine what evidence a later receiving tool exposes when an excluded request reaches it.

The distinction gives receiving tools a versioned policy category and gives a model more specific repair feedback. It remains narrower than intent inference or confinement. A refusal reports that a recognized source form matches a declared exclusion, nothing about why the producer emitted it, about authority used inside allowed modules, or about semantic correctness.

We ask four questions.

RQ1 Observability Which predeclared source contexts can be distinguished from ordinary unresolved names, and do their structured records conform?

RQ2 Gate sensitivity Where are controlled extraction, syntax, binding, type, capability, runtime, and semantic defects first detected?

RQ3 Contextual validation Under one fixed endpoint, corpus, and gate, how does supplied language information change first-draft outcomes?

RQ4 Diagnostic value Does policy-specific feedback change one-turn handling relative to rejection-only or generic unresolved-name feedback?

The paper makes four contributions.

1. A receiving-boundary policy-classification mechanism that preserves ordinary resolution and reclassifies only exact registered names on supported unresolved-identifier paths, with Study A testing the human diagnostic and every structured field.
2. A staged-detector characterization with controlled seeded defects, where Study B's frozen schedule balances capability families and recognized source contexts and records the earliest detector reached.
3. Contextual validation under natural first drafts, where Study C varies the supplied language-information bundle with everything else fixed and separates static acceptance from execution and hidden-oracle success.
4. A measured account of diagnostic value, where Study D compares rejection-only, generic-diagnostic, policy-text, and policy-JSON feedback and finds that policy text improves handling of appended registry-recognizable requests relative to generic feedback.

The primary contribution is therefore a receiving-boundary mechanism for versioned policy classification, validated through conformance and seeded detector characterization, and a measured account of its diagnostic value for one-turn model handling. Study C provides contextual validation of the broader staged outcome model under natural model output. The contribution is not a general restricted-API configuration mechanism. Existing mechanisms classify uses of resolved APIs. R0+B1 measures a different ambiguity at a producer-independent boundary, where a deliberately absent facility and an ordinary misspelling otherwise share an unresolved-name diagnostic.

The paper does not claim to infer producer intent, to confine execution or track authority, or that static acceptance establishes task correctness. These limits shape the design. Positive classifier cases are paired with negative controls, seeded static outcomes are separated from hidden-oracle outcomes, and downstream success is measured both as handling of the policy request and as preservation of intended task behavior.

2 Boundary and Study Gate

2.1 Policy classification

R0+B1 first performs ordinary lexical resolution. A resolved local name continues through normal checking, while an unresolved bare identifier is matched exactly against a checked registry before the generic binding diagnostic is finalized. A match yields its family-specific `L52xx` code and a non-match yields `L5002`. Direct calls, alias-value references, and expressions inside supported templates traverse this one recognition mechanism, which recovers no finer producer intent.

This ordering prevents the registry from overriding valid lexical meaning. A local declaration shadows the registered spelling because successful lookup ends the policy-classification question, text in comments and strings never reaches name resolution, unsupported syntax is rejected before the resolver, and qualified members and method idioms proceed through their established namespace or member paths.

The implementation exposes four hook sites in resolution order. These are `value-reference` for an unresolved value read, `direct-call` for an unresolved callee, `member-access` where an unresolved reference participates in member handling, and `forward-reference` when deferred lexical reconciliation still leaves a reference unresolved. Each hook is reached only after its ordinary lookup opportunity has failed and before a generic unresolved-name record is finalized, so the hook annotates where classification occurred without asserting what the producer meant.

The five evaluated families are dynamic evaluation, host interoperability or FFI, macro source generation, runtime reflection, and dynamic module loading. The exact diagnostic codes are `L5201` through `L5205`. Codes `L5298` and `L5299` guard malformed registry data and contradictory interface metadata at gate construction.

Checked data has an operational meaning. Gate construction validates the registry's schema, family identifiers, numeric code assignments, registered spellings, and interface metadata before compilation is available. A malformed entry receives `L5298` and contradictory metadata receives `L5299`, either of which prevents the gate from being built. Candidates are therefore classified against a registry instance that has already passed the build guards rather than against unchecked configuration read at diagnostic time.

Listing 1 shows an abbreviated sanitized entry. The omitted family entries use the same shape. The reserved arrays

remain present even when empty so that consumers do not infer support for an omitted field from its accidental absence.

Listing 1. Abbreviated sanitized registry entry

```
{
  "schema": "excluded-capabilities.v1",
  "languageRelease": "R0",
  "registryVersion": 1,
  "families": [
    {
      "id": "dynamic-evaluation",
      "diagnostic": "L5201",
      "rationale": "dynamic source evaluation",
      "unboundNames": ["eval"],
      "qualifiedSelectors": [],
      "importRoots": [],
      "authorityTags": []
    }
  ]
}
```

The registry schema also contains `qualifiedSelectors`, `importRoots`, and `authorityTags`. These fields are empty in R0+B1 and unevaluated. No result depends on object-capability propagation, qualified-selector policy, or import-root policy. This is an explicit non-claim. Method idioms, unbound qualified roots, and host-import syntax instead retain ordinary member, binding, or parser diagnostics.

The spellings `typeof` and `load` remain outside the registry. Their surface meanings are shared with ordinary operations or plausible user-defined vocabulary, so registering them would turn an ordinary unresolved spelling into a policy assertion that failed lookup alone cannot justify. The registry is deliberately not a list of every word associated with reflection or loading.

B1 attaches a `policy` object only to boundary-refusal JSON records. Adding the policy object to B1's JSON serialization does not alter the corresponding B1 human rendering and leaves non-policy JSON records byte-identical. The complete compiler test suite reports zero new failures. Listing 2 is a real sanitized record shape.

Listing 2. Sanitized R0+B1 refusal record

```
{
  "code": "L5201",
  "severity": "error",
  "message": "`eval` requests dynamic source evaluation,
     which this authoring boundary excludes (family dynamic-
     evaluation, registry v1)",
  "primary": {
    "file": "candidate.l",
    "line": 1,
    "col": 1,
    "endLine": 1,
    "endCol": 5,
    "lo": 0,
    "hi": 4,
    "message": ""
  },
  "secondary": [],
  "notes": [],
  "policy": {
    "family": "dynamic-evaluation",
    "requestForm": "direct-call",
    "registryVersion": 1
  }
}
```

The site-derived `requestForm` vocabulary has exactly the four hook-site values above. They describe classifier sites rather than inferred producer intent, and the field name is part of the serialized interface rather than a redefinition of the experiments' source-context terminology.

Numeric diagnostic stability follows a versioning contract. Within registry version 1 a code retains its declared family meaning, and an incompatible reassignment requires a registry-version change rather than silent reuse. Stability is a versioning contract, not a longitudinal empirical result, so the paper claims versioned numeric diagnostics and tests conformance at registry version 1.

2.2 Corpus and regression population

Studies B, C, and D use a frozen 24-task corpus with four tasks in each of six strata covering scalar and record transformations, collections, closed variants and guarded matching, strings and templates, multi-function composition, and localized edits. Each task provides a language-neutral contract, a required L interface, three visible examples, and five hidden checks.

The tasks were frozen, covering task text, interface, visible examples, hidden inputs, expected outputs, reference programs, and schedule assignments, before any seeded-defect or downstream outcome was observed, so no study outcome can revise the population another study evaluates. Independent L and general-purpose reference implementations were frozen only after producing byte-identical outputs on all eight cases for all 24 tasks. The L implementation provides the clean program from which Study B mutates, and cross-implementation agreement detects contract disagreements before either implementation becomes the hidden oracle.

Hidden-oracle discipline separates visible guidance from evaluation evidence: prompts expose the contract, interface, and visible examples but never the five hidden checks, so a candidate that imitates visible examples without satisfying the wider frozen contract can pass the gate and still receive an oracle mismatch.

The corpus is producer-independent. Its task contracts do not mention Endpoint A, and its acceptance, execution, and oracle procedures operate on persisted candidate text, so the same corpus and gate can receive a human implementation or output from a different model without changing the task definition. Study B mutates the frozen L references to ask which detector receives a controlled defect, Study C requests natural first drafts to ask how they move through the complete staged boundary, and Study D repairs Study B's

Table 1. Common detector and outcome mapping

Earliest outcome	Stored label	Study D terminal
Extraction	1	Extraction failure
Parse band	2	Parse rejection
Resolution or type	3O	Other static rejection
Capability refusal	3C	`L52xx` refusal
Runtime	4	Runtime failure
Oracle mismatch	5	Oracle mismatch
Oracle success	6	Correct completion

capability mutants to ask how feedback changes immediate handling.

R0 and R0+B1 agree on the acceptance decision for 41/41 frozen reference and example programs, with zero decision divergences. Registry-matched rejected inputs differ in diagnostic classification only. This is population-specific accept parity, not a proof of compiler equivalence. It establishes that the added classifier did not change acceptance for the clean regression population used to construct the studies.

2.3 Common outcome model

Table 1 aligns the detector taxonomy with Study C's stored six-label format. Capability refusal is subtype 3C rather than a hidden seventh numeric label. Study D reports 3C as its own terminal state when a repaired program still emits `L52xx`. Other static rejection is 3O.

An outcome is assigned to the earliest detector with evidence. After extraction and parsing, ordinary static processing begins; a failed lookup is classified against the registry before the generic unresolved-name outcome is finalized, and resolved programs continue through the remaining type checks, execution, and hidden-oracle comparison. This order prevents a later observation from replacing an earlier demonstrated failure.

Gate acceptance means zero static diagnostics. Oracle correctness additionally requires successful execution and exact hidden-oracle output. These predicates answer different operational questions. Acceptance reports whether the receiving compiler admits the program. Oracle correctness reports whether the accepted program behaves as required on the frozen hidden cases.

3 Study A Refusal and Record Conformance

3.1 Method

Study A contains 50 disjoint cases. Thirty predeclared family-by-context matrix cells cross five families with six source contexts, and each family appears once in each context, which makes family correctness observable separately from the fact of receiving some `L52xx` code. Direct call, value alias, and supported template contexts share the unresolved bare-identifier recognition path with different surrounding syntax. These positions are callee position, a value read propagated through an ordinary local binding, and an expression position inside L's template syntax. In every case the classifier sees an exact registered spelling only after ordinary lookup has failed.

Qualified member, host import, and method idiom cases exercise the negative observation surface. Method idioms resolve through member handling, qualified members through namespace handling, and host-import syntax is rejected in the parser band before bare-identifier resolution. Preserving these outcomes tests that the registry does not pull unrelated language mechanisms into one policy category.

Five nested cases contain an alias chain, nested template, string assembly, record-field holder, and allowed wrapper. The alias chain and nested template classify at the policy-matched originating reference. They do not classify later uses of a local alias or the containing expression. Both recognized nested cases use dynamic evaluation, so they demonstrate that the originating reference remains identifiable through those two enclosing structures. They do not establish nested-context behavior for every family.

The remaining nested cases test related non-events. These are string assembly whose fragments could form a policy word only at runtime, a record-field holder storing ordinary data, and an allowed wrapper invoking an admitted local abstraction. They keep textual resemblance from being mistaken for exact policy matching.

Fifteen controls contain five misspellings, five policy words in strings or comments, and five exact registered names legitimately bound as local functions. The misspellings test exactness because each remains an unresolved name but does not equal its registry spelling. The textual controls test that non-code text never enters the classifier. The local-binding controls use an exact-shadow design. Each registered spelling is declared as a legitimate local function in lexical scope and is then used normally. Ordinary lookup succeeds, so the program is correctly accepted even though its token text equals a registry entry.

Expected codes and structured fields were frozen independently of the registry instance under test. The conformance checker validates all 17 recognized inputs against the frozen expectation. It checks observed code, structured policy fields, primary token span, required field types, record count, and agreement between the human and JSON code.

3.2 Results

Fifteen of the thirty predeclared family-by-context matrix cells produced `L52xx`. All 15 carry the correct family code. The other 15 preserve their ordinary diagnostic. The split follows the predeclared mechanism boundary. It is not a post hoc partition into successful and unsuccessful examples.

Table 2. Study A predeclared matrix

Source context	Cells	Observed outcome
Direct call	5	5/5 `L52xx`
Value alias	5	5/5 `L52xx`
Supported template	5	5/5 `L52xx`
Qualified member	5	5/5 `L5002`
Host import	5	5/5 `L2008`
Method idiom	5	5/5 `L5006`

Table 3. Structured-record conformance

Checked property	Correct/total
Expected diagnostic code	17/17
Expected family	17/17
Expected `requestForm`	17/17
Registry version	17/17
Required fields and types	17/17
Registered-token span	17/17
Exactly one `L52xx` record	17/17
Human and JSON code agreement	17/17

The alias chain and nested template also receive `L5201`. Their primary event remains attached to the policy-matched originating reference rather than to the later alias use or the surrounding template. String assembly, the record-field holder, and the allowed wrapper are accepted. All five misspellings receive `L5002`. All ten textual and local-binding controls are accepted. No control receives `L52xx`.

The local exact-shadow cases are particularly important for interpreting a match. The registry does not reserve a token globally. A refusal occurs only when ordinary lookup has failed and the unresolved token exactly matches checked registry data. The result is a classification of a failed reference at a particular compiler site rather than a lexical ban.

Table 3 reports field-level conformance for all 17 recognized inputs. The registered-token span is the primary span. Each record contains exactly one `L52xx` event. Human and machine renderings agree on the numeric diagnostic, while the JSON representation supplies the separately checked family, source-site form, and registry version.

3.3 Answer to RQ1

R0+B1 distinguishes declared exclusions from ordinary unresolved names for the three tested contexts that share its recognition path. Family conformance is 15/15 in the balanced recognized matrix, two additional nested dynamic-evaluation cases receive `L5201`, every measured structured-record property conforms in 17/17 cases, and the control false-refusal count is 0/15. The result does not extend to the three deliberately out-of-surface contexts.

This answer is tied to the tested mechanism. Parser, namespace, and member handling retain their own diagnostics, and the matrix provides no general rate over possible source programs or possible ways to express an excluded operation.

4 Study B Seeded-Defect Characterization

4.1 Method

For each of 24 tasks, the harness records one clean control and attempts one mutation in seven defect classes, yielding 192 records. Table 4 defines every operator and its expected earliest detector. A mutation is made from the frozen accepted L reference rather than from a model response, which removes uncertainty about whether an earlier defect was already present.

The operators are deliberately single-site, which makes the intended perturbation auditable and reduces competition among possible earliest detectors. The extraction operator changes the received response rather than program semantics, the syntax, binding, type, runtime, and semantic operators each alter one localized site, and the capability operator appends one balanced registered-name request without modifying the preceding implementation. The design does not make every mutant realistic. It makes the expected detector interpretable.

The capability schedule is fixed by task index. Its family distribution is dynamic evaluation 5, host interoperability or FFI 5, macro source generation 5, runtime reflection 5, and dynamic module loading 4. Its source-context distribution is value alias 8, direct call 8, and template 8. The deterministic rotation balances the five families across the three recognized contexts without selecting mutants after observing results.

Balancing matters because a schedule concentrated in one family or context could report success for a narrow implementation accident. The fixed rotation covers every family repeatedly, represents the recognized contexts equally, and passes the same balance to Study D's downstream population. It is not a claim about natural frequency.

Semantic mutations are operator-balanced rather than outcome-balanced. The schedule rotates applicable operator flips before hidden-oracle execution, never searches for an oracle-failing mutation, and never replaces a survivor after observing its output, preserving the distinction between an applicable change and a demonstrated fault.

Runtime mutation uses one fixed loop-bound widening operator. Twenty-one tasks admit no such mutation because total functions dominate the frozen corpus, so the three applicable tasks locate the runtime detector without supporting any broad claim about runtime-fault sensitivity.

4.2 Results and RQ2

All 24 clean controls survive every detector, and every detected applicable mutation first appears at its expected class, from extraction failures before compilation through semantic mutants exposed only at the hidden oracle.

Table 4. Study B mutation operators and observed earliest outcomes

Defect class	Operator	Applicable	Expected detector	Observed
Extraction	Empty or prose response caught by extractor	24	Extraction	extraction 24
Syntax	Single-token statement corruption	24	Parse band	parse band 24
Binding	Rename one identifier to an unbound spelling	24	Resolution/type	resolution/type 24
Type	Flip a scalar return type	20	Resolution/type	resolution/type 20, n/a 4
Capability	Append a balanced registered-name request	24	Capability	capability 24
Runtime	Widen one loop bound by one	3	Runtime	runtime 3, n/a 21
Semantic	Flip one operator	21	Hidden oracle	oracle 16, survivor 5, n/a 3

All 24 capability mutants receive their family-correct `L52xx` code under R0+B1. Under R0, the same 24 receive `L5002`. Both gates reject all 24, isolating B1's classification change. The registry therefore adds a distinct observable category at the receiving boundary rather than merely renaming a collection of accepted and rejected outcomes. The acceptance decision stays fixed for this mutant population while the reason exposed to downstream consumers changes.

Sixteen semantic mutants are well typed, pass the static gate, execute, and reach a hidden-oracle mismatch. Neither generic nor policy diagnostics speak to these defects because the mutated programs resolve normally, so the first evidence of error is behavioral disagreement with the frozen task contract.

Five applicable operator flips survive the hidden oracle, and three are inapplicable. Survival means equivalence on the frozen oracle inputs, not global equivalence. A surviving mutation may be semantically equivalent for the whole program, may affect an unreachable or irrelevant case, or may differ only outside the finite oracle population. Mutation counts must therefore not be equated with fault counts. Retaining survivors makes this limitation visible instead of inflating detector performance by discarding inconvenient mutants.

Runtime coverage is sparse because only three total-function tasks admit the fixed loop-bound operator. Study B therefore characterizes this mutation system and corpus rather than completeness for natural defects. Its answer to RQ2 is about earliest observations under the declared operators. It is not an estimate of the distribution of failures in unconstrained model-generated programs.

5 Study C Contextual Specification Ablation

5.1 Design

Study C evaluates the surrounding staged boundary under natural first drafts. It uses Endpoint A, the frozen 24 tasks, four language-information conditions, three repetitions, medium effort, fresh stateless sessions, and no repair turn. All 288 cells use R0+B1.

Under the `o200k_base` tokenizer, full supplies about 21.7k supplemental tokens. These comprise about 17.4k grammar and static material, 1.1k standard-library surface material, and 3.2k usage-profile material including seven worked examples. Grammar-static supplies about 17.4k tokens. Examples-only supplies the seven examples at about 0.8k tokens. No-package supplies 0 supplemental language-package tokens. All conditions retain the fixed task scaffold, including the required function name and interface contract.

The comparison varies bundles rather than isolated facts. A contrast identifies the consequence of receiving one bundle instead of another, but it cannot attribute that consequence to a single sentence or token class inside the bundle.

The scheduler interleaves tasks, conditions, and repetitions with a recorded seed. Endpoint A exposes no generation seed. Acceptance requires zero gate diagnostics. Correctness requires label 6. Task-macro contrasts follow Equation 1.

$$\Delta_{a,b} = \frac{1}{24}\sum_{t=1}^{24}\left(\frac{1}{3}\sum_{r=1}^{3} Y_{t,a,r} - \frac{1}{3}\sum_{r=1}^{3} Y_{t,b,r}\right). \tag{1}$$

This is an empirical contrast over the frozen tasks, endpoint, sampling configuration, and run window.

5.2 Results and RQ3

Table 5 reports the complete label counts and contrasts against full. Bracketed ranges are 95 percent task-resampling intervals. Full reaches 100.0 percent acceptance and correctness. Grammar-static has two other-static rejections. Examples-only preserves acceptance but has two runtime failures and three oracle mismatches. No-package has 17 parser and 13 other-static rejections.

The oracle-correctness contrasts against full are −2.8 [−8.3, 0.0] for grammar-static, −6.9 [−18.1, 0.0] for examples-only, and −41.7 [−62.5, −20.8] for no-package. All 17 no-package parser rejections contain host-language idioms. Every one uses statement-terminating semicolons, 15 use C-style parenthesized loops, and 15 use parenthesized conditions.

The no-package failures show a concrete receiving-boundary effect. Without supplemental L material, Endpoint A frequently emits host-language-like syntax, and supplying the full package removes those observed failures for this corpus and run window.

Table 5. Study C task-macro outcomes and contrasts against full

Condition	Acceptance	Oracle correct	Parser reject	Acceptance contrast	Stored labels
Full	100.0 [100,100]	100.0 [100,100]	0.0%	reference	6:72
Grammar-static	97.2 [91.7,100]	97.2 [91.7,100]	0.0%	−2.8 [−8.3, 0.0]	3O:2, 6:70
Examples-only	100.0 [100,100]	93.1 [81.9,100]	0.0%	0.0 [0.0, 0.0]	4:2, 5:3, 6:67
No-package	58.3 [37.5,79.2]	58.3 [37.5,79.2]	23.6%	−41.7 [−62.5, −20.8]	2:17, 3O:13, 6:42

Examples-only demonstrates the distinct role of semantic evidence. Every candidate passes the static gate, yet two fail at runtime and three disagree with the hidden oracle. Worked examples are sufficient for high acceptance in this setting, but acceptance does not entail task correctness. The result supports the staged outcome model because later detectors contribute observations that the compiler cannot supply.

No Study C candidate emits `L52xx`, so Study C does not estimate the natural prevalence of excluded requests. Its role is contextual validation. Gate acceptance and semantic correctness separate under natural model output, and a language-information bundle materially affects which detector first receives a candidate even when the receiving gate is unchanged.

6 Study D Diagnostic-Value Ablation

6.1 Design

Study D answers RQ4 using Study B's 24 balanced capability mutants. Four feedback conditions, three repetitions, and one downstream turn produce $24 \times 4 \times 3 = 288$ completed cells with zero ignored cells. Each call uses hosted Endpoint A at medium effort in a fresh stateless session. Within each repetition, the 24-task by four-condition block is deterministically shuffled with schedule seed 52026. Per-cell records include prompt hashes.

The fixed prompt scaffold contains the mutated program and the sentence "This program was rejected by the receiving gate of language L." Conditions C1 through C3 then add "The gate reported:" followed by the payload verbatim, and the prompt asks for a corrected complete program that passes the gate and preserves intended behavior. C0 receives the rejection sentence with no diagnostic payload. One turn isolates the diagnostic's marginal value at the first downstream decision; multi-turn recovery would mix the initial intervention with later observations.

The four conditions are as follows.

C0 rejection-only The rejection sentence without a diagnostic payload.

C1 generic-diagnostic The pinned R0 `L5002` diagnostic for that mutant.

C2 policy-text The R0+B1 human `L52xx` diagnostic with family and registry prose.

C3 policy-JSON The R0+B1 JSON diagnostic containing the policy object.

C1 uses feedback captured from the pinned R0 build, and C2 and C3 use the corresponding R0+B1 human and structured records, so the pairing holds the candidate and rejection event fixed while varying only the delivered payload. The payload is the complete intervention: wording, length, representation, and metadata cannot be detached from policy content, so payload size is reported beside the behavioral outcomes rather than treated as an independently controlled covariate.

The C2 versus C1 contrast and the conceptual policy-handling construct were prespecified. A post-run construct audit revealed that the original lexical operationalization misclassified four legitimate or inert occurrences, so we report the resolver-aware `L52xx`-absence correction as the main analysis and retain the original lexical operationalization as a sensitivity analysis. The remaining contrasts are secondary.

All outcomes are mechanical. *Gate accept* requires zero R0+B1 diagnostics. The main-analysis *post-repair* `L52xx` *absence* outcome records whether recompilation emits any policy diagnostic. It does not imply syntactic validity or gate acceptance, and a parse-rejected repair never reaches the resolver; those outcomes are reported separately. The *registered-spelling absent* outcome is the original conservative lexical conjunction, retained as a sensitivity analysis, requiring no `L52xx`, no unresolved registered name, and no surviving bare registered-name use. *Oracle correct* requires exact base-task output. *Other-registered-family substitution* records a different registered family that triggers `L52xx` after the repair.

These outcomes separate policy handling from overall quality; hidden-oracle correctness is the strongest because it requires extraction, static acceptance, execution, and agreement with the frozen outputs. The population consists of probes appended to otherwise-correct programs rather than integrated capability uses. Even C0 reaches an 88.9 percent task-macro absence rate, often through wholesale rewriting, so the experiment measures differential handling of a recognized policy request, not redesign of an integrated capability use.

We report task-macro means over 24 tasks with 72 cells per condition. Input payload tokens use the `o200k_base` tokenizer already used in Study C. UTF-8 bytes and endpoint-reported output tokens are also reported.

Table 6. Study D one-turn outcomes; payload sizes in o200k_base tokens and UTF-8 bytes, output tokens as reported by the provider

Condition	Payload tok.	Payload bytes	Gate accept	L52xx absent	Spelling absent	Oracle correct	Other-family subst.	Mean output tok.
C0 rejection-only	0	0	63.9%	88.9%	87.5%	29.2%	11.1%	608
C1 generic-diagnostic	68	213	79.2%	79.2%	75.0%	54.2%	20.8%	271
C2 policy-text	95	344	87.5%	93.1%	93.1%	59.7%	6.9%	93
C3 policy-JSON	110	421	86.1%	93.1%	93.1%	44.4%	6.9%	95

6.2 Results

Table 7 gives paired task-macro contrasts. Intervals are 95 percent task-resampling intervals from a paired task-cluster bootstrap over the frozen corpus. They describe empirical task-macro contrasts and are not superpopulation inference.

Under the randomized interleaved schedule, assignment to the delivered C2 payload rather than the delivered C1 payload changed immediate repair behavior for this endpoint, corpus, and run window. Post-repair L52xx absence rises from 79.2 to 93.1 percent, while other-registered-family substitution falls from 20.8 to 6.9 percent. Of the 67 C2 outputs without L52xx, 63 pass the gate and four remain rejected for non-policy diagnostics, while all 57 such C1 outputs pass the gate, so the ten-cell absence difference comprises six additional accepted repairs and four shifts to other rejection classes.

The policy-over-generic oracle-correctness contrast is positive at 5.6 points, but its interval includes zero. This separates a resolved effect on handling of the appended request from an unresolved effect on complete task behavior. The evidence does not establish that the same payload generally improves the surrounding algorithm.

The L52xx-absence contrast is positive in all five families, between +6.7 and +26.7 points. It is descriptively concentrated in the eight tasks assigned template-context probes, at +41.7 points in that subgroup and 0.0 in the direct-call and value-alias subgroups. Because source context was assigned between tasks rather than crossed within each task, these subgroup differences do not identify a causal context effect. The artifact contains the full breakdown; cells are small and no inferential statistics are reported.

6.3 Resolver-aware metric correction

Table 8 reclassifies all 288 stored outputs using the R0+B1 resolver. Exactly four repairs retain a registered spelling that the resolver legitimately resolves or that is inert in non-code text. Three occur in C1 and one in C0, with none in C2 or C3. Under the paper's exact-shadow semantics these cases are policy-compliant. The earlier lexical outcome was therefore differentially conservative against C1.

The resolver-aware recomputation shrinks the C2 minus C1 contrast from +18.1 to +13.9 points, while its interval still excludes zero. No repair in any condition leaves the original family's L52xx in place; every resolver-visible failure is substitution into a different registered family. The registered-spelling-absent row in Table 6 retains the original conservative values.

Raw JSON is not better model-facing feedback in this sample. C2 and C3 have identical observed task-macro rates for L52xx absence and other-registered-family substitution, but these aggregates do not represent identical per-cell successes: for both outcomes the 72 paired task-repetition cells split symmetrically into two C2-only and two C3-only disagreements.

Oracle correctness shows a different paired asymmetry. C2 and C3 are both correct in 30 cells, C2 alone is correct in 13, C3 alone is correct in two, and neither is correct in 27. The C3 minus C2 oracle contrast is −15.3 points with an interval excluding zero. The structured record supplies coordinates, field structure, and serialized metadata useful to deterministic consumers, but those properties do not make the larger raw JSON payload a better instruction for this endpoint.

6.4 Answer to RQ4

For appended registry-recognizable policy requests, policy-text feedback reduces repeated or substituted policy refusals in one-turn output relative to generic L5002. It increases post-repair L52xx absence by 13.9 points with an interval excluding zero, and because no repair in any condition left the original family's diagnostic in place, every remaining policy event in this sample is a substitution into another registered family, so the 13.9-point substitution reduction mirrors the same events. This does not imply an equal improvement in valid repair: the gate-acceptance contrast is +8.3 points [0.0, 18.1] and the oracle-correctness contrast is +5.6 points [−4.2, 18.1]. The aggregate contrast is driven by the template-assigned tasks. Policy-JSON has identical observed task-macro handling rates but lower oracle correctness than policy text, with symmetric per-cell disagreement on the equal-rate outcomes and asymmetric discordance on oracle correctness.

7 Interpretation and Threats

7.1 Claims supported by the four studies

Studies A and B evaluate the policy classifier: exact classification on the tested unresolved-identifier path with full structured-property conformance, nothing about arbitrary syntax or arbitrary expressions of an excluded operation, and a paired R0 versus R0+B1 comparison showing that B1

Table 7. Study D paired contrasts in percentage points with 95 percent task-resampling intervals

Contrast	Gate accept	L52xx absent	Oracle correct	Other-family subst.
C2 minus C1	+8.3 [0.0, 18.1]	+13.9 [5.6, 23.6]	+5.6 [−4.2, 18.1]	−13.9 [−23.6, −5.6]
C1 minus C0	+15.3 [5.6, 26.4]	−9.7 [−19.4, −0.0]	+25.0 [12.5, 38.9]	+9.7 [1.4, 19.4]
C3 minus C2	−1.4 [−6.9, 4.2]	0.0 [−4.2, 4.2]	−15.3 [−27.8, −4.2]	0.0 [−4.2, 4.2]

Table 8. Resolver-aware reclassification of stored outputs

Cond.	Spelling absent	Resolved or inert	Orig. remains	Other-family
C0	63	1	0	8
C1	54	3	0	15
C2	67	0	0	5
C3	67	0	0	5

adds a policy category without changing rejection while sixteen well-typed semantic mutations first become observable at the hidden oracle. Study C shows that producer context strongly affects what reaches the boundary and that accepted natural output still requires semantic evidence. Study D measures immediate handling behaviorally: generic `L5002` feedback encourages symptom-level rebinding or other-family substitution more often than policy text, the uncertain 5.6-point oracle contrast prevents a stronger claim about general task correctness, and raw policy JSON conforms for tools yet is worse than policy text on oracle correctness, so the structured-record claim stays scoped to deterministic tool consumption and audit.

7.2 Scope and construct validity

The measured property is source-visible policy-request classification, not runtime confinement, proof-carrying code [15], software fault isolation [20], or an information-flow theorem [18]. A policy-matched reference is not authority: the classifier observes a failed lexical lookup whose exact spelling appears in checked registry data, an admitted wrapper could use capabilities internally without exposing a registered spelling, and the empty `authorityTags`, `qualifiedSelectors`, and `importRoots` fields provide no evidence about object-capability tracking.

The numeric codes have meaning under the declared versioning contract. Study A demonstrates conformance at registry version 1 only, which is why the paper describes the records as versioned policy evidence rather than as unqualified stable evidence.

Study A's thirty cells are a researcher-selected Cartesian product, not a population denominator, and its three positive contexts share one unresolved bare-identifier mechanism, so fifteen of thirty reports matrix results rather than a coverage rate. The controls bound only the tested false-refusal surface and cannot enumerate every shadowing or scope interaction.

Study B uses seeded faults rather than natural model errors. Applicability differs by task, five semantic mutants survive the finite oracle, the runtime operator applies to only three tasks, and single-site mutations simplify the interactions found in natural programs. The hidden oracle is finite. Exact output supports correctness only for the frozen evaluation, while an oracle mismatch is direct evidence of a violated requirement, and the claims use that asymmetry.

7.3 Study C validity and replay

Study C uses one hosted endpoint. The blinded paper masks its identity, while the artifact discloses the exact model and version identifier, client versions, effort setting, prompt hashes, and run window. No generation seed or route-stability guarantee is available, task-resampling intervals quantify variation across the frozen corpus only, and public language materials make contamination impossible to exclude.

The four conditions are bundle comparisons, so the observed contrasts validate the practical packages as delivered rather than independent effects of grammar clauses, library declarations, usage guidance, examples, or token count. The fixed task scaffold remains in every condition, so no-package means the task-specific contract remains while supplemental language material is absent. Stored candidates, diagnostics, execution results, and oracle outputs deterministically regenerate every table without contacting the endpoint; re-executing prompts is a stochastic replication of the protocol.

7.4 Study D validity

Study D uses one endpoint, one downstream turn, and mutants that are registry-recognizable by construction, so it establishes nothing about requests outside the classifier surface, and it omits iterative workflows that could recover from weak first edits.

The `L52xx`-absence outcome aligns policy-event handling with the compiler's exact-shadow semantics, but it is a recompilation observation rather than proof of removal: a parse-rejected repair never reaches the resolver and still emits no `L52xx`, which is why gate acceptance is a separate outcome. The registered-spelling-absent metric is retained only as a secondary conservative lexical measure.

Feedback conditions bundle wording, length, representation, and metadata, so payload sizes are part of the observed interventions rather than separable nuisance variables. C3 is larger than C2 under both the shared tokenizer and UTF-8 bytes, and the C3 versus C2 contrast cannot identify whether the oracle difference arises from syntax, field density, ordering, length, or another representational feature.

The population consists of policy probes appended to otherwise-correct programs. High removal under rejection-only feedback shows that wholesale rewriting can delete an obvious probe without redesigning an integrated capability use. The experiment therefore measures differential handling of a recognized appended policy request in this constructed population. Integrated-mutant redesign is future work.

8 Related Work

8.1 Generation, validation, and feedback

Grammar Prompting, PICARD, Synchromesh, and type-constrained generation place grammatical or type information inside decoding [14, 17, 19, 21]. R0+B1 instead classifies persisted output from any producer after it reaches the receiving compiler. The locations are complementary. Producer-side constraints reduce invalid candidates, while receiving-side diagnostics remain necessary for stored source, uncontrolled producers, and later handling.

Text2DSL studies structured grammar, API, and identifier context with factorial ablation [9, 10]. Study C similarly varies language information, but to validate a fixed staged receiving boundary with parser, static, runtime, and oracle outcomes reported separately. It does not claim the first context ablation.

Other work separates DSL executability from correctness, compiles generated code against live lexical scope, or evaluates a production scientific DSL through staged structural, execution, and scientific validation [1, 6, 22]. Our boundary instead identifies a declared excluded family when an exact registered reference fails lookup at a supported site, and Study C supplies the analogous reminder that structural acceptance and task correctness remain distinct.

FeedbackEval evaluates feedback-driven repair broadly [3]. Study D narrows the intervention to one receiving-boundary ambiguity and holds the mutant population, task contract, and downstream turn fixed, permitting a direct comparison between generic unresolved-name information and a policy-specific explanation.

Krishnamurthi and Flatt ablate type-error diagnostic detail for coding agents in programs resolved far enough to support typing [11]. Study D varies policy content for excluded requests that would otherwise remain generic unresolved names, and additionally compares the human rendering with the raw machine rendering. Identical observed task-macro request-handling rates do not imply identical paired outcomes or identical hidden-oracle correctness.

8.2 Restricted APIs and capability scope

Clippy's disallowed-method and disallowed-type facilities and Error Prone's `RestrictedApi` attach policy to resolved program elements, so a successful lookup identifies the API being restricted [2, 5]. R0+B1 addresses the complementary case in which the facility is deliberately absent and lookup fails, where the compiler otherwise has only the generic unresolved-name category used for spelling errors.

Object-capability work distinguishes authority carried through references from textual names [13], and capability-tracking systems can enforce richer static properties for agent-produced programs by reasoning about reference and effect flow [16]. R0+B1 makes no such claim. Its evaluated path matches a declared excluded name after failed lookup, the reserved metadata fields are empty, and allowed modules remain outside the measured policy evidence.

8.3 Testing and staged evidence

Differential testing motivates the R0 versus R0+B1 accept-parity comparison [12], and mutation testing motivates controlled seeding with explicit retention of equivalent mutants [4, 7, 8]. Study B uses mutations to localize evidence across a receiving boundary rather than to score a test suite, and Study D reuses the balanced mutants as a controlled population without turning them into a prevalence sample.

9 Conclusion

R0+B1 makes declared excluded facilities observable on a specific unresolved-identifier path without reclassifying arbitrary text, unsupported syntax, or locally resolved names, and its structured records conform on every measured field. Studies A and B validate the classifier, Study C confirms that gate acceptance and semantic correctness separate under natural output, and Study D shows that policy text increases post-repair `L52xx` absence by 13.9 points over generic feedback, an effect driven by the template-assigned tasks, with every remaining policy event being substitution into another registered family. Policy-JSON has identical observed handling rates but is worse on hidden-oracle correctness. The resulting claim is precise: versioned policy classification can reduce repeated or substituted policy refusals in one-turn output for appended, registry-recognizable requests in this constructed population, and it does not infer intent, establish runtime confinement, redesign an integrated capability use, or make accepted programs correct.

Artifact Availability

The anonymous artifact at https://anonymous.4open.science/r/dsl-boundary-lmpl26/ contains pinned R0 and R0+B1 builds, the pseudonymized B1 patch, the frozen corpus and oracles, all prompts and payloads, complete records for Studies A through D, and deterministic rerun or replay scripts including the reclassification analysis. It discloses the exact endpoint, client, and run configuration to reviewers, while language and organizational identifiers are pseudonymized for review.

References

[1] Negin Ayoughi, David Dewar, Shiva Nejati, and Mehrdad Sabetzadeh. 2026. DSL or Code? Evaluating the Quality of LLM-Generated Algebraic Specifications: A Case Study in Optimization at Kinaxis. arXiv:2601.00469 doi:10.48550/arXiv.2601.00469

[2] Clippy Developers. 2026. Disallowed Methods and Disallowed Types. https://doc.rust-lang.org/clippy/lint_configuration.html. Tool documentation, accessed July 2026.

[3] Dekun Dai, MingWei Liu, Anji Li, Jialun Cao, Yanlin Wang, Chong Wang, Xin Peng, and Zibin Zheng. 2025. FeedbackEval: A Benchmark for Evaluating Large Language Models in Feedback-Driven Code Repair Tasks. *arXiv preprint arXiv:2504.06939* (2025). Version 2 revised 26 February 2026. doi:10.48550/arXiv.2504.06939

[4] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer* 11, 4 (1978), 34–41.

[5] Error Prone Developers. 2026. RestrictedApi. https://errorprone.info/bugpattern/RestrictedApi. Tool documentation, accessed July 2026.

[6] Ethan Holbrook, Juan C. Verduzco, and Alejandro Strachan. 2026. Evaluating LLM-Generated Code for Domain-Specific Languages: Molecular Dynamics with LAMMPS. arXiv:2603.20630

[7] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678.

[8] Rene Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are Mutants a Valid Substitute for Real Faults in Software Testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[9] Alexander V. Kozachok, Alexander M. Nazimov, and Shamil G. Magomedov. 2026. Context-Aware Distillation and Ablation for Text2DSL. arXiv:2606.22578

[10] Alexander V. Kozachok, Alexander M. Nazimov, and Shamil G. Magomedov. 2026. Text2DSL: LLM-Based Code Generation for Domain-Specific Languages. arXiv:2606.22586

[11] Shriram Krishnamurthi and Matthew Flatt. 2026. Type-Error Ablation and AI Coding Agents. arXiv:2606.01522

[12] William M. McKeeman. 1998. Differential Testing for Software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[13] Mark S. Miller, Bill Tulloh, and Jonathan S. Shapiro. 2005. The Structure of Authority: Why Security Is Not a Separable Concern. In *Multiparadigm Programming in Mozart/Oz (Lecture Notes in Artificial Intelligence, Vol. 3389)*. Springer, 2–20. doi:10.1007/978-3-540-31845-3_2

[14] Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin T. Vechev. 2025. Type-Constrained Code Generation with Language Models. *Proceedings of the ACM on Programming Languages* 9, PLDI, Article 171 (2025), 26 pages. Also available as arXiv:2504.09246. doi:10.1145/3729274

[15] George C. Necula. 1997. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 106–119. doi:10.1145/263699.263712

[16] Martin Odersky, Yaoyu Zhao, Yichen Xu, Oliver Bračevac, and Cao Nguyen Pham. 2026. Tracking Capabilities for Safer Agents. arXiv:2603.00991

[17] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable Code Generation from Pre-trained Language Models. In *International Conference on Learning Representations*.

[18] Fred B. Schneider, J. Gregory Morrisett, and Robert Harper. 2001. A Language-Based Approach to Security. In *Informatics: 10 Years Back. 10 Years Ahead*, Reinhard Wilhelm (Ed.). Lecture Notes in Computer Science, Vol. 2000. Springer, 86–101. doi:10.1007/3-540-44577-3_6

[19] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 9895–9901. doi:10.18653/v1.2021.emnlp-main.779

[20] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. 1993. Efficient Software-Based Fault Isolation. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*. 203–216. doi:10.1145/168619.168635

[21] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar Prompting for Domain-Specific Language Generation with Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 36. 65030–65055. doi:10.5555/3666122.3668959

[22] Yaoyu Zhao, Yichen Xu, Oliver Bračevac, Cao Nguyen Pham, Frank Zhengqing Wu, and Martin Odersky. 2026. Lacuna: Safe Agents as Recursive Program Holes. arXiv:2605.28617