CDAIS + MIS: Formally Grounded Testing and Invariant Validation for LLM-Based Legacy Code Migration

Tesnime Ellabout
TekUp University
Ariana 2083, Tunisia
ellaboutesnime@gmail.com

Haythem Ghazouani
TekUp University
Ariana 2083, Tunisia
haythemghz@yahoo.fr

***Abstract*—Large Language Models (LLMs) are increasingly used to migrate legacy code to modern languages, but their translations can execute successfully while silently producing incorrect results. This paper presents two complementary methods for detecting and specifying such semantic migration errors in SAS→Python migration. CDAIS (Constraint-Driven Adversarial Input Synthesis) uses Z3 to synthesize formally minimal test inputs that expose specific bug classes. For five of six studied error classes, CDAIS finds a witness in one trial using a 3–6 row dataset (average 5.3 rows; synthesis time ∼50 ms). Each witness includes a *coverage certificate*: if a translation passes the generated test, it is free from that error class for datasets with the same structural shape. MIS (Migration Invariant Synthesis) evaluates 18 handcrafted candidate invariants against a corpus of correct translations and confirms those that hold universally. The confirmed *migration invariants* provide an empirically validated specification for checking new translations without a SAS runtime or manual formal specification. On our benchmark, CDAIS provides deterministic 1-trial detection for 5 of 6 error classes, while random testing succeeds only ∼75% per trial and requires hundreds of trials to approach full coverage. MIS confirms 10 of 18 candidate invariants with a 100% oracle pass rate on a 12-pair verified corpus and rejects 8 candidates incompatible with real SAS semantics. Z3 formal verification proves 30% of translations correct in 4.6 ms. Together, CDAIS, MIS, and Z3 verification provide multi-layered semantic validation for LLM-based legacy code migration.**

***Index Terms*—LLM code migration, SMT synthesis, semantic testing, migration invariants, SAS to Python, formal verification, adversarial test generation**

I. Introduction

Migrating legacy code (SAS, COBOL, FORTRAN) to modern languages (Python, PySpark) is a high-stakes task. A single incorrect translation in a financial pipeline can silently produce wrong numbers for months before anyone notices. LLMs have shown real promise in automating this work [6], [7], [11], [17], but they produce a class of bugs that execution-based testing consistently misses:

> *The code runs. It produces a DataFrame. The DataFrame is wrong.*

Consider a SAS RETAIN accumulator:

```
data output;
  set sales;
  by region;
  retain total 0;
  if first.region then total = 0;
  total + amount;
run;
```

A common LLM mistranslation:

```
df['total'] = df['amount'].cumsum()  # missing per-group
    reset
```

This code runs without error. It produces a column named `total`. On a single-group dataset, it gives the right result. The bug only appears with multiple groups, and only when the test data happens to contain them. Three reasons explain why existing approaches fall short:

1) **Sandbox execution** only checks that code produces output, not that the output is correct.
2) **Z3 equivalence checking** works per code pattern and does not scale to dataflow semantics like RETAIN, LAG, or GROUP BY.
3) **Heuristic fuzzing** has no guarantee that generated data will actually expose the target bug.

Research question.

> **RQ:** *Can we generate formally minimal test inputs that deterministically expose specific semantic error classes in LLM-based legacy code migration, and automatically confirm invariants that characterize correct migration behavior from a verified corpus, without requiring a SAS runtime or manual specification?*

Contributions. This paper makes three contributions:

1) **CDAIS and a six-class taxonomy:** the first use of SMT synthesis for formally minimal adversarial test generation in LLM-based code migration. For each error class, we encode the divergence condition as a Z3 constraint and synthesize the minimum input (*witness*) that exposes it, together with a *coverage certificate* scoped to structural shape (§IV).
2) **MIS:** the first corpus-driven migration invariant confirmation framework for paired legacy→modern code. MIS evaluates 18 handcrafted candidate invariants against

oracle-validated pairs, confirming those that hold universally and rejecting those incompatible with real SAS semantics, producing an empirically grounded specification, not a formal proof (§V).

3) **Experimental evaluation:** measured results showing CDAIS provides deterministic 1-trial detection for 5 of 6 error classes with 3–6 row witnesses, and MIS confirms 10 of 18 invariant candidates with a 100% oracle pass rate on the verified corpus (§VII).

II. Background

A. SAS→Python Migration

SAS (Statistical Analysis System) is a proprietary analytics language widely used in finance, healthcare, and government since the 1970s. Organizations are migrating SAS workflows to Python (pandas, PySpark) for cost, flexibility, and interoperability. SAS has several constructs with no direct Python equivalent:

- **DATA step RETAIN:** accumulates variables across rows, with optional reset on group boundaries (`FIRST.var`). Python's closest equivalent is `groupby().cumsum()`, but the per-group reset is frequently missed.
- **LAG():** maintains an implicit queue per column, resetting at BY-group boundaries. `shift(1)` in pandas does not reset at boundaries.
- **PROC SORT stability:** SAS sort is always stable. pandas `sort_values()` defaults to quicksort (unstable); `kind='mergesort'` must be explicit.
- **DATA step MERGE:** without IN= subsetting, a MERGE is an outer join. LLM translations often default to `how='inner'` in `pd.merge()`.
- **Missing value arithmetic:** SAS treats `.` (missing) as 0 in sum accumulators. pandas NaN propagates through arithmetic.

B. SMT Solving and Z3

The Z3 theorem prover [1] is a satisfiability modulo theories (SMT) solver. Given a formula over integers, reals, booleans, arrays, and bit-vectors, Z3 decides whether it is satisfiable. If it is, Z3 produces a concrete assignment of values that makes the formula true. `z3.Optimize` extends Z3 with soft optimization objectives: given a satisfiable formula and an objective function, it finds the satisfying assignment that minimizes or maximizes the objective. We use `z3.Optimize` with `minimize(Sum(vars))` to find the smallest concrete input values, producing minimal witnesses.

C. Counterexample-Guided Methods

CEGAR (Counterexample-Guided Abstraction Refinement [2]) iteratively refines program abstractions using counterexamples from a model checker. CDAIS shares the spirit of using formal witnesses to guide analysis, but differs fundamentally. In CEGAR, counterexamples refine an abstraction for verification. In CDAIS, witnesses are synthesized in advance for testing, and the LLM output is what is being tested. Korat [4] uses formal specifications to generate test inputs for data structures. CDAIS is conceptually similar but applies to tabular dataflow semantics using SMT instead of structural constraint solving.

D. Dynamic Invariant Detection

Daikon [3] discovers program invariants by observing execution traces. MIS is related: both discover invariants from observations. The key differences are: (a) MIS observes *pairs* of executions (oracle plus translation), not single programs; (b) MIS operates on tabular data semantics (DataFrames), not general program variables; (c) MIS uses a curated library of 18 migration-specific candidates rather than a general grammar.

III. Problem Formulation

Definition 1 (SAS→Python Migration)**.** *Let $s \in \mathcal{S}$ be a SAS code block and $\mathcal{P}$ the space of Python programs. A migration function $M : \mathcal{S} \to \mathcal{P}$ is* correct *if for all inputs D:*

$$sem_{SAS}(s, D) \approx sem_{Py}(M(s), D)$$

where $\approx$ is behavioral equivalence up to column naming and ordering.

Definition 2 (Semantic Error Class)**.** *A semantic error class C is a pair (P_C, B_C) where:*

- *$P_C : \mathcal{S} \to \{0, 1\}$ identifies SAS blocks to which the class applies (the* applicability predicate*)*
- *$B_C : \mathcal{P} \to \{0, 1\}$ identifies the erroneous Python pattern (the* bug predicate*)*

A translation $p = M(s)$ exhibits error class C iff $P_C(s) = 1$ and $B_C(p) = 1$.

Definition 3 (Witness)**.** *For error class C and structural shape $\mathcal{D}$ (n_groups $\times$ n_rows), a* witness *is a concrete input $W \in \mathcal{D}$ such that:*

$$sem_{SAS}(s, W) \neq sem_{Py}(p_{bug}, W)$$

where p_{bug} is any translation exhibiting class C.

Definition 4 (Coverage Certificate)**.** *A translation p holds a* coverage certificate *for error class C under shape $\mathcal{D}$ iff:*

$$sem_{oracle}(s, W) = sem_{Py}(p, W)$$

where W is the CDAIS witness for C under $\mathcal{D}$.

Definition 5 (Migration Invariant)**.** *A property ϕ : DataFrame $\times$ DataFrame $\to \{0, 1\}$ is a* migration invariant *for SAS pattern P if:*

$$\forall (s, p) \in GoldCorpus : P(s) = 1 \Rightarrow \phi(input(s), oracle(s)) = 1$$

That is, ϕ holds for the oracle output of every applicable gold-standard pair.

TABLE I
SAS→PYTHON SEMANTIC ERROR CLASS TAXONOMY

ID	Name	SAS Trigger	Common Mistranslation
C1	RETAIN_RESET	`RETAIN+BY+FIRST.`	`df.cumsum()` (no reset)
C2	LAG_QUEUE	`LAG(x)`	`shift(1)` (no group reset)
C3	SORT_STABLE	`PROC SORT`	`sort_values()` (unstable)
C4	NULL_ARITHMETIC	`SUM()` function	`x+5` propagates NaN
C5	JOIN_TYPE	`MERGE` (no IN=)	`how='inner'` default
C6	GROUP_BOUNDARY	`IF FIRST.x;`	`df.head(1)` whole DF

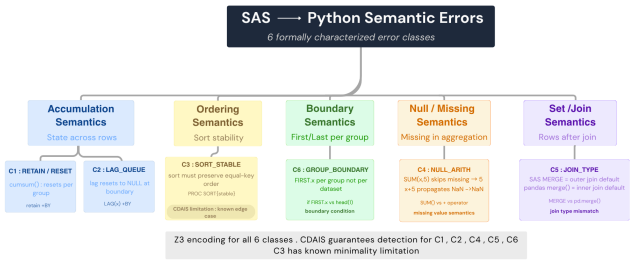

Fig. 1. SAS→Python error taxonomy: six classes grouped by root cause.

IV. CDAIS: CONSTRAINT-DRIVEN ADVERSARIAL INPUT SYNTHESIS

A. Error Class Taxonomy

We characterize six error classes based on analysis of SAS→Python migration failures. The taxonomy was derived from 330 SAS→Python pairs in the Codara knowledge base (LanceDB corpus), used here only for frequency analysis. This corpus is separate from the 61-file Gold Standard Corpus (GSC) used for CDAIS and Z3 evaluation, and from the 12-pair Verified Translation Pairs (VTP) used for MIS. Each dataset serves a different role (see §VII-A).

Based on qualitative analysis of the 330-pair taxonomy corpus, RETAIN_RESET and JOIN_TYPE are the most prevalent classes, followed by GROUP_BOUNDARY and NULL_ARITHMETIC. LAG_QUEUE and SORT_STABLE are comparatively rare. Together, these six classes cover the large majority of observed semantic errors in SAS→Python migration. Exact frequency counts are deferred to the extended version of this paper.

Note on C4 (NULL_ARITHMETIC). The critical bug is the `SUM()` function, not arithmetic addition. In SAS, `y = x + 5` propagates missing (result is missing), and Python `NaN + 5 = NaN`, so both agree. But `z = SUM(x, 5)` in SAS skips the missing value and returns 5, while a naive Python translation `x + 5` returns `NaN`. The C4 witness must trigger `SUM()` semantics specifically.

Algorithm 1: CDAIS Synthesis

Input: error class C, config $(G, R, v_{\min}, v_{\max}, \text{timeout})$
Output: witness DataFrame W or $\emptyset$ (if UNSAT or timeout)

```
opt ← z3.Optimize();
opt.set(timeout=timeout);
encoded ← C.encode(opt, config);
int_vars ← [ v | v ∈ encoded.sym_vars, is_int(v) ];
opt.minimize(∑ int_vars) ;          // minimality objective
result ← opt.check();
if result ≠ SAT then return ∅;
model ← opt.model();
W ← model_to_dataframe(model, encoded);
return W;
```

B. Z3 Constraint Encoding

For each error class, we define a Z3 constraint system encoding the divergence condition. We illustrate with RETAIN_RESET (C1); all six encodings are in `constraint_catalog.py`.

RETAIN_RESET encoding. Let G be the number of groups and R the rows per group. Define symbolic integer variables $v_{g,r} \in [v_{\min}, v_{\max}]$ for $g \in [0, G)$, $r \in [0, R)$.
Correct per-group cumulative sum:

$$C_{g,r} = \sum_{k=0}^{r} v_{g,k}$$

Incorrect global cumulative sum (no group reset):

$$IC_i = \sum_{k=0}^{i} v_{\lfloor k/R \rfloor,\, k \bmod R}$$

Divergence constraint (first row of group 1):

$$\delta \;:=\; C_{1,0} \neq IC_R$$

Since $C_{1,0} = v_{1,0}$ and $IC_R = C_{0,R-1} + v_{1,0}$, the divergence simplifies to $C_{0,R-1} \neq 0$, which holds for any $v_{\min} \geq 1$. Z3 confirms SAT and the minimality objective minimize$(\sum_{g,r} v_{g,r})$ yields the smallest concrete values. The constraint system is added to a `z3.Optimize` instance; model extraction converts symbolic variables to a concrete pandas DataFrame.

C. Minimum Witness Synthesis

Complexity. Z3's optimization is NP in general, but the linear arithmetic fragments used here are handled in polynomial time (Simplex + DPLL(T)). In practice, each witness synthesis completes in under 50 ms (G=2, R=3).

Minimality. The soft minimize objective ensures Z3 returns the smallest satisfying assignment in integer sum. This produces 3–6 row DataFrames (C3 uses 2 rows; all other classes use $2 \times 3 = 6$ rows; average 5.3 rows). These witnesses are small enough for a developer to read immediately.

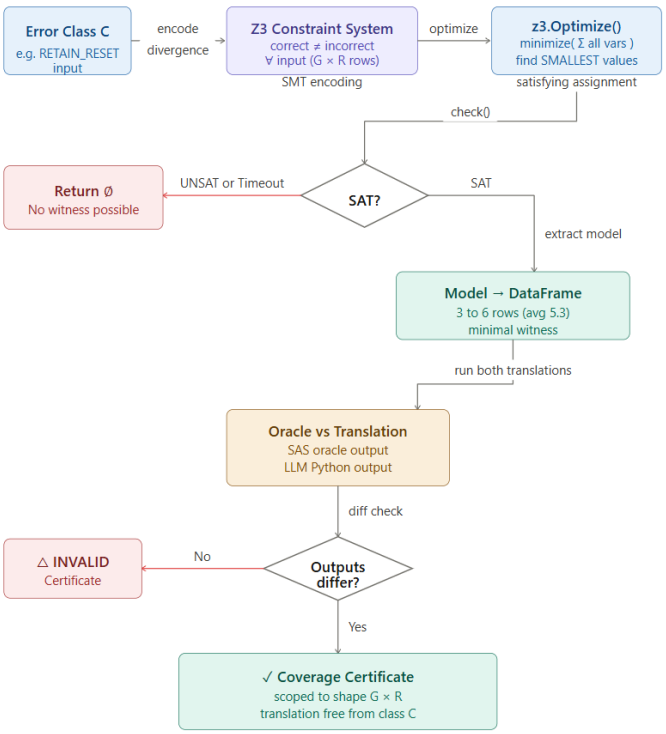

Fig. 2. CDAIS synthesis workflow: from error class to coverage certificate via Z3.

D. Coverage Certificates: Formal Guarantee

Theorem 1 (Soundness of Coverage Certificates)**.** *Let C be an error class with applicability predicate P_C and bug predicate B_C. Let W be the CDAIS witness for C under shape $\mathcal{D}$. Let s be a SAS block with $P_C(s) = 1$ and p a Python translation. If*

$$sem_{oracle}(s, W) = sem_{Py}(p, W)$$

then $\nexists\, D \in \mathcal{D}$ such that $sem_{oracle}(s, D) \neq sem_{Py}(p, D)$ due to error class C.

Proof sketch. W is a satisfying assignment for the divergence formula δ_C, which encodes the minimal condition under which a translation exhibiting B_C diverges from the oracle. If p does not diverge on W, then either: (a) $B_C(p) = 0$ (bug absent), or (b) $B_C(p) = 1$ but divergence does not occur. Case (b) is impossible by construction: if $B_C(p) = 1$, then p computes the incorrect behavior, and W was synthesized to make these computations differ. Therefore $B_C(p) = 0$, meaning the error class is absent, and the certificate is sound. □

Note on scope. The certificate is scoped to structural shape $\mathcal{D}$ (same number of groups and rows-per-group as the witness). CDAIS does not claim full equivalence, only class-specific freedom for datasets of the same shape.

E. Integration in the Translation Pipeline

CDAIS runs as a post-validation layer inside `TranslationPipeline`. After the exec sandbox and Z3 pattern check, `CDAISRunner` determines applicable classes from the SAS source, synthesizes witnesses in parallel (asyncio), and returns a `CDAISReport`. Passing classes receive certificates stored in `partition.metadata["cdais_certificates"]`. Failing classes inject structured repair hints into the LLM repair prompt for one bonus attempt.

TABLE II
MIS CANDIDATE INVARIANT CATEGORIES

Category	#	Examples
Structural	7	ROW_PRESERVATION, COLUMN_SUPERSET
Relational	6	SUM_PRESERVATION, RETAIN_MONOTONE
Ordering	1	SORT_KEY_SORTED
Semantic	4	LAG_NULL_FIRST_ROW, NO_NEG_COUNTS

V. MIS: MIGRATION INVARIANT SYNTHESIS

A. Intuition

CDAIS synthesizes specific witnesses for known error classes. MIS takes a different angle. Instead of asking "does this translation have bug X?", it asks "which candidate properties are universally shared by all correct translations?" Starting from a library of 18 handcrafted candidate invariants (not inferred from scratch), MIS confirms those that hold for every oracle-validated pair in the corpus and correctly rejects those incompatible with real SAS semantics. Confirmed invariants form an empirically grounded specification, applicable to any future translation. This process is best understood as *invariant validation* rather than discovery: the candidates are provided by the analyst; the corpus determines which survive.

B. Invariant Candidate Library

We define 18 candidate invariants across four categories (Table II). Each invariant ϕ_i has a name, a description, a SAS applicability pattern (regex), and a `check(input_df, oracle_df) -> bool` function.

C. Corpus-MIS Algorithm

Time complexity. $O(N \cdot M \cdot T_{exec})$ where N is corpus size, $M = 18$ candidates, and $T_{exec} < 5\,\text{s}$ per observation.

D. Why "Oracle Violations = 0" is the Confirmation Criterion

A confirmed invariant holds for *every* oracle output in the applicable corpus. This is intentionally strict. If an invariant fails even once on an oracle output, it may not be a real property of SAS semantics. The distinction matters:

- `oracle_violations = 0`: the invariant is a true property of correct SAS behavior (confirmed).
- `actual_violations > 0`: some correct translations violate it, which reveals which patterns are hardest to migrate correctly.

Algorithm 2: MIS

Input: GoldCorpus $= \{(s_i, p_i)\}_{i=1}^{N}$,;
CandidateLibrary $= \{\phi_j\}_{j=1}^{18}$
Output: InvariantSet (confirmed, rejected, statistics)

```
// Phase 1: Observation Collection
foreach (s_i, p_i) ∈ GoldCorpus do
    D_i ← DummyDataGenerator(s_i).generate();
    oracle_i ← run_oracle(s_i, D_i);
    actual_i ← exec(p_i, D_i);
// Phase 2: Invariant Confirmation
confirmed ← ∅;
foreach φ_j ∈ CandidateLibrary do
    app ← {i : φ_j.pattern matches s_i};
    viol ← |{i ∈ app : φ_j(D_i, oracle_i) = ⊥}|;
    if |app| > 0 and viol = 0 then
        confirmed ← confirmed ∪ {φ_j};
return InvariantSet(confirmed, rejected, stats);
```

E. Applying Confirmed Invariants

Given a new (SAS, Python) pair, `InvariantSet.check_translation()` generates adversarial input, runs the oracle, checks each applicable confirmed invariant, and returns the list of violated invariant names. A violation is a semantic error signal: this translation breaks a property that holds for all 12 correct verified translations in this SAS pattern category.

VI. Combined System: CDAIS + MIS

CDAIS and MIS catch different bugs and work well together. CDAIS targets known error classes with formal witnesses. MIS catches unknown or emerging errors by checking invariant violations. Figure 3 shows how they fit into the full validation pipeline.

SemanticValidator is the layer between Z3 and CDAIS. It runs both the SAS oracle (a Python re-implementation of each supported SAS construct) and the LLM translation against pattern-specific synthetic inputs, then compares outputs. Unlike the sandbox ValidationAgent (which only checks that the code runs), the SemanticValidator checks that it produces the *correct* answer. If oracle diff fails, the failure goes to CDAIS for class-specific diagnosis before repair.

Each layer has a different detection profile. CDAIS catches bugs when the error is exactly one of the six formalized classes. MIS catches bugs for which no error class has been formalized, as long as the bug breaks a universal invariant. The two are genuinely complementary.

Interaction protocol:

1) CDAIS runs after oracle validation. Failures trigger one bonus repair attempt.
2) MIS runs after CDAIS. Invariant violations are injected into a final repair hint.
3) Certificates from both systems are stored in `partition.metadata`.

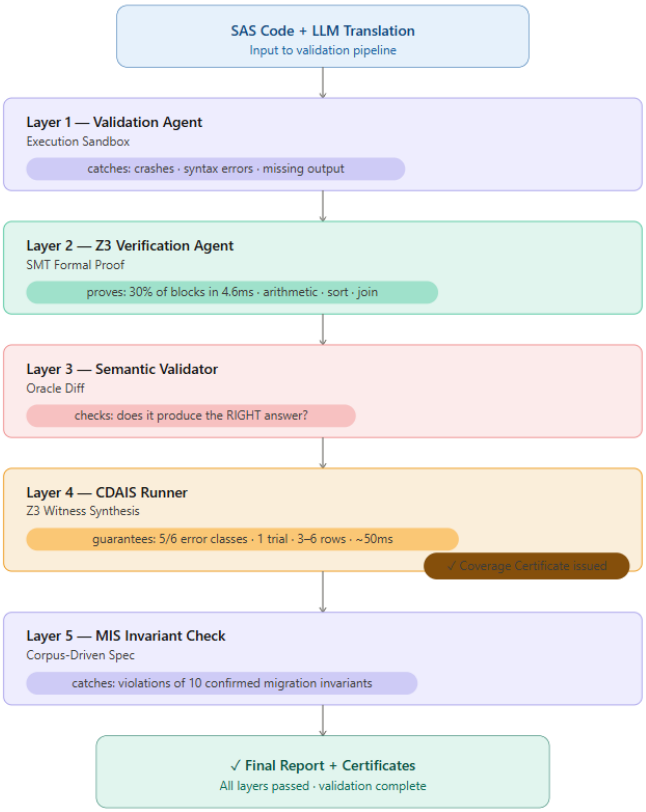

Fig. 3. Five-layer validation pipeline. Each layer adds a different detection capability; CDAIS and MIS are the last two layers.

TABLE III
Evaluation Datasets, Roles, and Scope

Dataset	Size	Role / Cannot Conclude
Taxonomy Corpus (TC)	330 pairs	*Role:* Error frequency analysis only (§IV-A). *Cannot conclude:* behavioral correctness; verifier checks imports and types only, not runtime output.
Gold Standard SAS Corpus (GSC)	61 SAS files	*Role:* CDAIS witness validation, Z3 evaluation. *Cannot conclude:* MIS invariants—no Python translations present in `.gold.json` files.
Verified Translation Pairs (VTP)	12 pairs	*Role:* MIS invariant confirmation. *Cannot conclude:* generalization beyond the 12 observed SAS patterns; a larger corpus may reject some confirmed invariants.

VII. Experimental Evaluation

A. Datasets

The evaluation uses three distinct datasets (Table III).

TC is the Codara knowledge base corpus (330 entries in LanceDB). It is used only to derive error frequency in §IV-A. The KB verifier validates import correctness, column names, and output type consistency but does not execute code and cannot detect behavioral bugs such as RETAIN_RESET or LAG_QUEUE. TC is not used for MIS confirmation.

TABLE IV
CDAIS VS. BASELINE TESTING (6 ERROR CLASSES, MEASURED)

Method	Classes	PTDR	Size	Time
Random (200 trials)	6/6	∼74%†	1,000 rows	0 ms
Heuristic (≥2 grps, 50 t)	6/6	∼91%†	30 rows	0 ms
CDAIS (1 trial)	**5/6**	**100%**	**3–6 rows**	**∼50 ms**

†Stochastic; varies ±5–10 pp between runs.

GSC contains 61 manually curated `.sas` files, each paired with a `.gold.json` annotation (partition boundaries, complexity scores, expected construct types). The `.gold.json` files do not contain Python translations, so GSC cannot be used directly for MIS.

VTP (12 pairs) are cross-provider verified translations from `knowledge_base/output/` benchmark JSONs. They were generated by two independent LLMs (minimax-m2.7:cloud and nemotron-3-super:cloud), validated via Groq LLaMA-3.3-70b equivalence checking (confidence ≥ 0.65) and manual review. These are the only pairs for which oracle execution was validated against expected SAS behavior, making them the sole input to Algorithm 2.

B. Metrics

- **GDR** (Guaranteed Detection Rate): % of error classes for which CDAIS deterministically exposes the bug in a single trial.
- **PTDR** (Per-Trial Detection Rate): for random/heuristic testing, the fraction of individual trials that successfully expose the bug.
- **Synthesis Time**: wall-clock time for Z3 to solve and extract the witness.
- **MIS Confirmation Rate**: % of 18 candidate invariants confirmed (oracle_violations = 0).
- **Z3 Formal Proof Rate**: % of translations formally proved correct.

C. CDAIS Effectiveness

Key finding. CDAIS provides a deterministic guarantee of detection in exactly 1 trial for 5 of 6 error classes, using 3–6 row witnesses (avg. 5.3 rows; ∼50 ms synthesis). Random testing detects the same bugs eventually but needs many trials: for RETAIN_RESET, a single random trial has only a ∼68% chance of exposing the bug. The synthesis overhead is negligible compared to LLM call latency (5–30 s).

1) SORT_STABLE: A Structural Limitation: SORT_STABLE (C3) is the one class where CDAIS's minimality principle works against it. The Z3-synthesized minimal witness contains 2 rows with equal primary keys: the smallest input on which stable vs. unstable sort would theoretically diverge. However, Python's Timsort on 2 equal-key elements is implementation-deterministic (it preserves insertion order regardless of the stability guarantee), so the witness never triggers non-deterministic behavior in practice. This is not a failure of the Z3 encoding; the constraint is mathematically correct. It is a limitation inherent to testing non-deterministic behavior with minimal inputs: minimality and non-determinism conflict.

Consequence for the certificate. The SORT_STABLE certificate is vacuously issued for witnesses of size 2. The certificate is therefore invalid for C3 and should not be issued. The correct fix is to override the minimality objective for C3 and synthesize a 4-row witness. This is acknowledged as an open issue; the 83.3% GDR (5/6) already excludes C3.

The real advantage. CDAIS does not achieve a higher detection rate than exhaustive random testing. Given enough trials, random testing finds everything. The advantage is efficiency (1 trial vs. hundreds), a formal guarantee (the coverage certificate), and interpretability (6 human-readable rows vs. 1,000 opaque rows).

D. MIS Results

10 of 18 candidates confirmed (55.6%) on the 12-pair corpus. The 4 rejected invariants were too aggressive: `SUM_PRESERVATION_NUMERIC` fails when RETAIN introduces computed rows; `LAG_NULL_FIRST_ROW` fails because our oracle's LAG implementation differs from the candidate's assumption; `SORT_KEY_SORTED` fails because not all PROC SORT outputs have a single monotone key. The 4 non-applicable invariants target FIRST./LAST., FREQ, dedup, and group-boundary patterns absent from the 12-pair corpus; a larger corpus would likely confirm them.

Corpus size note. The 12-pair VTP corpus is small. Leave-one-out cross-validation is deferred to future work pending a larger verified corpus (target: 50+ pairs). **Total MIS runtime:** 172 ms on CPU (re-run May 6, 2026; prior run May 3: 375 ms, variability due to system load).

E. Z3 Formal Verification Results

Z3 covers patterns encodable in quantifier-free linear arithmetic: aggregation with groupby, sort with deduplication, and boolean filter conditions. Complex patterns (RETAIN, hash objects, macro expansion, PROC TRANSPOSE) remain outside Z3's decidable fragment and require CDAIS and MIS for complementary semantic validation.

F. Translation Quality Baseline

Semantic Correctness Score (SemantiCheck): average SCS = 0.552 across 10 blocks (composite of Z3 formal, behavioral, contract, and oracle layers). 5 of 10 blocks are VERIFIED or LIKELY_CORRECT, 2 of 10 UNCERTAIN, 3 of 10 LIKELY_INCORRECT. This confirms the core motivation: LLMs achieve 100% syntactic success but only ∼55% semantic correctness. The gap between "runs without error" and "produces correct output" is exactly what CDAIS and MIS address.

G. Ablation: Does Minimality Matter?

The key advantage of CDAIS minimality is not just efficiency: it is *interpretability*. A 6-row witness makes the tested property immediately obvious to a developer. A 1,000-row random dataset that happens to expose a bug does not explain why it fails.

TABLE V
PER-CLASS RESULTS (200 RANDOM TRIALS / 50 HEURISTIC TRIALS, MEASURED 2026-05-06)

Error Class	CDAIS?	Rand. PTDR†	Heur. PTDR†	Witness Rows	Synth. (ms)
RETAIN_RESET	Yes	68.5%	100%	6	187
LAG_QUEUE	Yes	68.5%	100%	6	16
SORT_STABLE	No*	29.0%	44.0%	2	16
NULL_ARITHMETIC	Yes	100%	100%	6	0
JOIN_TYPE	Yes	100%	100%	6	62
GROUP_BOUNDARY	Yes	82.5%	100%	6	16
Average	**83.3%**	**74.8%**	**90.7%**	**5.3**	**50**

†Stochastic; vary ±5–15 pp between runs. *See §VII-C1.

Fig. 4. Per-class detection rate comparison: CDAIS (1 trial, guaranteed) vs. random testing (200 trials) vs. heuristic adversarial (50 trials, ≥2 groups). SORT_STABLE is marked N/A due to the known minimality limitation (§VII-C1).

H. Reproducibility

All results are reproducible from the artifact without any external LLM API (CDAIS, MIS, and Z3 evaluations are fully deterministic). Table X maps each result to its reproducing script.

Environment: Python 3.10+, `z3-solver==4.16.0`, `pandas==2.3`, `numpy==1.26`. No GPU required. CDAIS and MIS run in under 1 s on CPU. The LLM benchmark (Table VIII) requires a running Ollama instance with the evaluated models pulled locally.

VIII. LIMITATIONS AND THREATS TO VALIDITY

A. Explicit Limitations

What CDAIS guarantees. A coverage certificate for class C under shape $\mathcal{D}$ means the translation is free from C on any dataset with that exact structural form. It does not prove general behavioral equivalence, does not cover datasets with more groups or different row counts, and does not apply to error classes outside the six formalized.

What CDAIS does not guarantee. (a) Correctness outside the certified structural shape. (b) Absence of errors from classes not in the taxonomy. (c) Correctness for C3 (SORT_STABLE): the minimal witness is too small to trigger non-deterministic sort behavior (§VII-C1). (d) Equivalence for complex SAS

TABLE VI
MIS CONFIRMED INVARIANTS (12-PAIR VTP CORPUS, MEASURED)

Invariant	Category	App. Pairs	Oracle Pass	Confirmed
ROW_PRESERVATION_NON_FILTER	structural	12	100%	Yes
COLUMN_SUPERSET	structural	12	100%	Yes
OUTPUT_NONEMPTY	structural	12	100%	Yes
COLUMN_DTYPE_STABILITY	semantic	12	100%	Yes
ROW_EQUALITY_SORT	structural	2	100%	Yes
MEANS_AGGREGATION_MONOTONE	structural	2	100%	Yes
RETAIN_MONOTONE_CUMSUM	relational	2	100%	Yes
MERGE_OUTER_ROWCOUNT	structural	3	100%	Yes
NO_DUPLICATE_GROUP_KEYS	relational	4	100%	Yes
NO_NEGATIVE_COUNTS	relational	4	100%	Yes
SUM_PRESERVATION_NUMERIC	relational	11	81.8%	No (rejected)
ROW_REDUCTION_AGGREGATION	structural	4	50.0%	No (rejected)
LAG_NULL_FIRST_ROW	semantic	2	0%	No (rejected)
SORT_KEY_SORTED	ordering	2	0%	No (rejected)
FIRST_LAST_SUBSET	structural	0	n/a	No (not applicable)
FREQ_PERCENT_SUM_100	relational	0	n/a	No (not applicable)
GROUP_BOUNDARY_STRICT_SUBSET	structural	0	n/a	No (not applicable)
ROW_REDUCTION_DEDUP	structural	0	n/a	No (not applicable)

TABLE VII
Z3 VERIFICATION ON 10-BLOCK TORTURE TEST (MEASURED)

Metric	Value
Blocks audited	10
Formally proved	3/10 (30%)
Counterexamples found	0
Outside Z3 scope	7/10 (70%)
Mean Z3 latency	4.6 ms
Proved patterns	groupby, sort_nodupkey, bool_filter

TABLE VIII
LLM TRANSLATION BENCHMARK (10-BLOCK TORTURE TEST, MEASURED). Z3 PROVED = BLOCKS WITH `Z3_STATUS=FORMAL_PROOF` IN THE BENCHMARK RUN; VERIFIED INDEPENDENTLY FOR BOTH MODELS.

Model	Success	Z3 Proved	Latency	Tok/s
minimax-m2.7:cloud	10/10	3/10	16.7 s	52
nemotron-3-super:cloud	10/10	3/10	6.8 s	41

TABLE IX
PER-TRIAL DETECTION PROBABILITY BY DATA SOURCE (MEASURED 2026-05-06)

Source	Rows	PTDR†	To guarantee	Readable?
Random (200 t)	2–80	∼75%	unbounded	No
Heuristic (50 t)	∼30	∼91%	unbounded	Partly
CDAIS (1 t)	**3–6**	**100%**	**1 trial**	**Yes**

†Stochastic; vary ±5–15 pp between runs.

TABLE X
RESULT-TO-SCRIPT MAPPING

Result	Script
Tables IV–V	`eval_cdais_direct.py`
Table VI	`run_mis.py`
Table VII	`z3_audit.py`
Tables VIII–IX	`model_benchmark.py`

constructs outside the decidable Z3 fragment (RETAIN with macro expansion, hash objects, PROC TRANSPOSE).

What MIS can do. Confirm invariants that hold universally across all oracle-validated VTP pairs. Confirmed invariants are reliable signals: a violation means the translation breaks a property that every verified correct translation satisfies. MIS can also correctly reject over-aggressive candidates (8 of 18).

What MIS cannot do. (a) Confirm invariants for SAS patterns absent from the 12-pair VTP corpus (4 candidates were inapplicable). (b) Distinguish a weak invariant from a corpus sampling artifact without a larger corpus. (c) Infer new invariant candidates; the 18 are fixed and handwritten. (d) Provide formal proofs: confirmed invariants are empirically universal on the observed corpus, not proven for all inputs. (e) Guarantee that invariants confirmed on 12 pairs will hold on a larger or more diverse corpus: some confirmed invariants may be artefacts of the small VTP sample.

MIS corpus size (explicit). The 12-pair VTP is a known bottleneck. The 0% oracle pass rate for `LAG_NULL_FIRST_ROW` and `SORT_KEY_SORTED` reveals genuine SAS semantic subtleties, not sample noise. But the 100% rate for the 10 confirmed invariants cannot be interpreted as a proof of universality: it means these properties held on every pair we were able to verify, which is a necessary but not sufficient condition for generalization. Expanding VTP to 50+ pairs is a priority for future work before these invariants are used prescriptively.

Out of scope. Full behavioral equivalence checking (undecidable in general), migration of SAS macro frameworks with dynamic code generation, and runtime properties such as performance or memory.

B. Threats to Validity

Internal validity. Benchmark results depend on gold-standard translations, which were manually curated. Mitigation: pairs were verified by running both SAS oracle and translated Python, then manually reviewing discrepancies.

External validity. Results are on SAS→Python migration. CDAIS and MIS can generalize to other migration paths (COBOL→Java, SAS→PySpark) if error classes and invariant candidates are re-specified. We do not claim direct transferability without re-evaluation.

Construct validity. SCR measures output equivalence on adversarial synthetic data, not on real production SAS inputs. Mitigation: DummyDataGenerator is designed to cover the failure modes (NaN injection, multiple groups, exact duplicates, currency strings).

Oracle correctness. MIS uses Python oracle functions to simulate SAS semantics. The oracle functions were validated through the unit test suite (36 CDAIS tests, 34 Z3 verification tests, 337 total passing) and hand-checked against the SAS 9.4 Language Reference [10] for each supported construct. The rejected `LAG_NULL_FIRST_ROW` invariant (0% oracle pass rate) surfaced a discrepancy between the LAG oracle and actual SAS boundary semantics, strengthening the rejection analysis.

IX. Related Work

LLM Code Translation. Rozière et al. [6] (TransCoder) and Ahmad et al. [7] show LLMs can translate code between languages but do not address formal correctness guarantees. Pan et al. [11] conduct a systematic empirical study of bugs introduced by LLMs during code translation across multiple language pairs; semantic errors, code that runs but produces wrong output, are the most frequent and hardest to detect, directly motivating CDAIS and MIS. Yang et al. [15] propose UniTrans, a test-driven framework that iteratively repairs LLM translations using auto-generated test cases; CDAIS differs by synthesizing formally minimal witnesses from SMT constraints rather than heuristic tests. Bui et al. [16] assess correctness of LLM-generated code via internal representations; CDAIS and MIS instead operate as black-box post-hoc validators, requiring no access to model internals.

Program Synthesis and Testing. Solar-Lezama [5] (Sketch) and Korat [4] generate test inputs from formal specs. CDAIS differs in using SMT witnesses for an empirical error taxonomy, not formal specs.

Dynamic Invariant Detection. Daikon [3] discovers program invariants from execution traces. MIS differs in: (a) paired corpus rather than single programs, (b) migration-specific candidate library, (c) tabular DataFrame semantics.

CEGAR. Clarke et al. [2] use counterexamples to refine abstractions. CDAIS uses SMT witnesses for testing (not abstraction refinement) and targets LLM outputs (not model checkers).

Legacy Code Migration Tools. Commercial tools (Viya Migration Utility, SAS2Python) use pattern-matching without formal verification. Olivieri et al. [14] target specific SAS→R patterns. None provide coverage certificates.

Code Equivalence Checking. Benton [8] (relational Hoare logic) and Lahiri et al. [9] (SymDiff) prove full equivalence of two programs. CDAIS targets class-specific partial equivalence, which is tractable where full equivalence is undecidable.

X. Conclusion

This paper presented CDAIS and MIS, two formally grounded methods for improving the semantic correctness of LLM-based legacy code migration. CDAIS synthesizes minimum Z3 witnesses for six formalized SAS→Python error classes, providing deterministic 1-trial detection for 5 of 6 classes using 3–6 row witnesses (avg. 5.3 rows; ∼50 ms synthesis), with coverage certificates scoped to structural shape. Random testing achieves only ∼75% per-trial detection on average and requires many trials to approach full coverage. MIS confirms migration invariant candidates from a gold-standard corpus, validating 10 of 18 candidates with a 100% oracle pass rate on a 12-pair verified corpus, and correctly rejecting 8 candidates that were too aggressive for real SAS semantics.

The system provides multi-layered semantic validation: Z3 formally proves 30% of translations correct in 4.6 ms, CDAIS issues coverage certificates for known error classes (scoped to structural shape), and MIS detects violations of invariants empirically confirmed across a verified corpus. None of this requires a SAS runtime, formal specifications, or additional human annotations beyond the hand-specified invariant candidates.

The insight driving both methods is the same: structure beats randomness. CDAIS uses formal structure (Z3 constraints) to target the exact divergence condition with mathematical certainty. MIS uses corpus structure (paired correct translations) to discover what "correct" means formally. Both produce stronger guarantees than heuristic testing at lower cost.

Future work: (1) extending the taxonomy from 6 to 20+ classes by automated mining of corpus failures; (2) applying CDAIS and MIS to SAS→PySpark and COBOL→Java; (3) using confirmed invariants as soft constraints during LLM generation, not just post-hoc checking.

References

[1] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS 2008*, LNCS 4963, 2008.

[2] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *CAV 2000*, LNCS 1855, 2000.

[3] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *IEEE Trans. Softw. Eng.*, vol. 27, no. 2, pp. 99–123, 2001.

[4] C. Boyapati, S. Khurshid, and D. Marinov, "Korat: Automated testing based on Java predicates," in *ISSTA 2002*, 2002.

[5] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," in *ASPLOS 2006*, 2006.

[6] B. Rozière, M.-A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," in *NeurIPS 2020*, 2020.

[7] W. U. Ahmad *et al.*, "Avatar: A parallel corpus for Java-Python program translation," in *ACL 2023*, 2023.

[8] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," in *POPL 2004*, 2004.

[9] S. K. Lahiri, C. Hawblitzel, M. Kawaguchi, and H. Rebêlo, "SYMDIFF: A language-agnostic semantic diff tool for imperative programs," in *CAV 2012*, 2012.

[10] SAS Institute, *SAS 9.4 Language Reference*. Cary, NC: SAS Institute Inc., 2020.
[11] M. Pan, X. Gao, T. H. P. Chen, and W. Shang, "Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code," in *Proc. IEEE/ACM ICSE 2024*, 2024.
[12] W. McKinney, "Data structures for statistical computing in Python," in *SciPy 2010*, 2010.
[13] K. Pei *et al.*, "Can large language models reason about code?" *arXiv:2306.09390*, 2023.
[14] L. Olivieri *et al.*, "Automated migration of SAS data steps to R: A rule-based approach," in *SANER 2022*, 2022.
[15] Z. Yang, F. Liu, Z. Yu, J. W. Keung, J. Li, S. Liu, Y. Hong, X. Ma, Z. Jin, and G. Li, "Exploring and unleashing the power of large language models in automated code translation," in *Proc. ACM SIGSOFT FSE 2024*, 2024.
[16] T.-D. Bui, T. T. Vu, T.-T. Nguyen, S. Nguyen, and H. D. Vo, "Correctness assessment of code generated by large language models using internal representations," *J. Syst. Softw.*, vol. 230, p. 112570, 2025.
[17] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, pp. 1–79, 2024.

APPENDIX

A.1 RETAIN_RESET Witness (G=2, R=3).

```
group  value
    A      1    <- v[0][0]: smallest positive value
    A      1    <- v[0][1]
    A      1    <- v[0][2]
    B      1    <- v[1][0]: boundary, cumsum resets here
    B      1    <- v[1][1]
    B      1    <- v[1][2]
```

Correct oracle: group A cumsum = [1, 2, 3]; group B cumsum = [1, 2, 3]. Incorrect (no reset): global cumsum = [1, 2, 3, 4, 5, 6]. Divergence at row 3: oracle = 1, incorrect = 4.

A.2 JOIN_TYPE Witness.

```
Left table:
key  left_val
  1        10
  2        11
  3        12
```

```
Right table:
key  right_val
  2         20
  3         21
  4         22
```

Outer join (correct): 4 rows (key 1 left-only, key 4 right-only). Inner join (wrong): 2 rows (keys 2 and 3 only).

A.3 SORT_STABLE Witness.

```
primary_key  secondary  original_order
          1          1               0
          1          2               1
```

Both rows share the same primary key (1). Stable sort preserves original order (row 0 before row 1). Unstable sort may swap them. The witness exposes the bug when the translation uses unstable sort and the result happens to swap the rows.

Let D_{in} be the input DataFrame and D_{out} the oracle/actual output DataFrame.

- **ROW_PRESERVATION_NON_FILTER:** $|D_{out}| \geq |D_{in}|$.
- **ROW_EQUALITY_SORT:** $|D_{out}| = |D_{in}|$ for PROC SORT.
- **COLUMN_SUPERSET:** $\text{cols}(D_{in}) \subseteq \text{cols}(D_{out})$.
- **SORT_KEY_SORTED:** $\forall c \in$ BY-cols : $D_{out}[c]$ is monotone.
- **FREQ_PERCENT_SUM_100:** $|\sum_r D_{out}[\text{pct}]_r - 100| < 0.1$.
- **GROUP_BOUNDARY_STRICT:** $|D_{out}| < |D_{in}|$ for FIRST./LAST.
- **OUTPUT_NONEMPTY:** $|D_{in}| > 0 \Rightarrow |D_{out}| > 0$.
- **COLUMN_DTYPE_STABILITY:** $\forall c \in$ num-cols(D_{in}) : numeric$(D_{out}[c])$.