

From Lexical Edits to Verified Stemming Policies: Two-Sided Adaptation in Radixor

Leo Galambos

Abstract—A standard Radixor dictionary can serve as a reusable substrate for task-specific lexical policy, rather than a fixed endpoint. Positive assignments merge or redirect forms; negative membership boundaries prevent generic rules from admitting unwanted forms to protected classes. We specify how these extensions are installed, persisted, and checked against posting support. In a shared English MIRACL development case, the reported positive-only realization admits 11,291 unintended surfaces, motivating an observed-vocabulary closure procedure. The adapted deployment improves the standard model on all five recorded retrieval metrics; the companion empirical paper analyzes those selected-task outcomes. We distinguish Radixor’s operational contribution from its inherited instruction language. A direct comparison establishes that XSTEM’s terminal cut-and-append action embeds in the older Egothor patch executor, while one patch can preserve source characters between edits. XSTEM’s published comparison omits this relevant predecessor despite its pre-2022 availability in Lucene. Method-level comparison separates later suffix-deletion and captured-span approaches from the earlier Galambos/Egothor lineage and distinguishes software citation from algorithmic attribution. The contribution is an explicit path from a reusable standard model to a two-sided lexical policy whose compiled realization is checked against retrieval evidence. Closure is restricted to the checked inventory; transfer and equal-budget comparative performance remain untested, and the original audit artifacts are incomplete.

Index Terms—stemming, lexical adaptation, compiler correctness, negative constraints, model reuse, information retrieval

I. INTRODUCTION

A useful default dictionary should be a starting model, not a locked endpoint. Radixor allows a standard compiled model to be reopened, extended, and persisted through the same patch-based stemming engine [1], [2]. The engineering question is whether a task-specific extension can express both the assignments that are wanted and the memberships that must not occur after generalization.

Positive assignment correctness is only half of that question. A word explicitly assigned to a protected output may be correct while an unedited word also reaches that output through a more general rule. In exact-term retrieval, that extra member changes both matching support and term statistics. The present construction closes this gap through explicit negative boundaries and a class-level posting audit.

The companion paper, *Where Retrieval Quality Goes* [3], reports the shared English MIRACL case. A PyStemmer-initialized partition search selects the target relation; standard Radixor hosts its compiled realization. At the common scorer, the final model improves all five recorded means over standard Radixor and PyStemmer. The present paper does not count those outcomes as a second replication. Its focus is how the

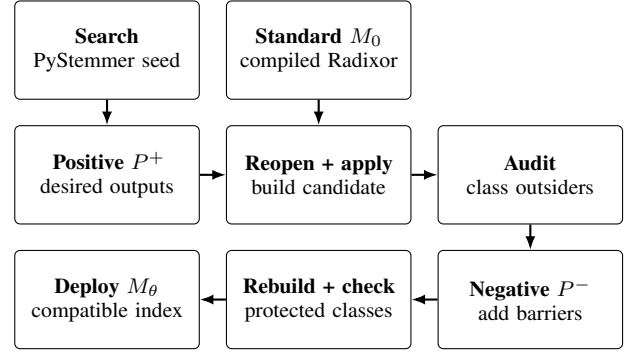


Fig. 1. Two-sided adaptation of a standard compiled dictionary. PyStemmer supplies the search initializer; standard Radixor supplies the deployment substrate. Positive assignments and negative barriers become one updated model. A failed closure check requires another audit/repair round; the reported inventory required one round. Recompile and a compatible index are still necessary.

relation is installed, checked, and compared with historically available adaptation mechanisms.

Three different claims must be separated. *Representational capability* asks whether a target mapping can be expressed. *learning capability* asks how a useful mapping is selected from evidence. *deployment fidelity* asks whether the runtime model implements the selected mapping under the intended analyzer and index semantics. A simple exact table can be fully expressive on a finite inventory without learning anything; a sophisticated learner can generalize without enforcing a hard protected-class constraint. Neither fact determines their comparative efficiency.

The contributions are therefore concrete: a two-sided interface for an existing compiled model; a conditional posting-equivalence argument with an observed-vocabulary closure procedure; and a source-bounded capability comparison that identifies both substantive precedents and the remaining deployment obligation. The article does not use missing documentation as proof that another system cannot implement the same policy. The current customization interface is documented in 2026; these operations must not be backdated wholesale to the 2001 publication.

II. POLICY OBJECTS AND THE SCOPE OF EXACTNESS

Let V be a fixed normalized inventory covering all audited corpus surfaces and the protected query surfaces. A scalar analyzer $M : V \rightarrow S$ emits one output at each original token position; it includes the declared original-token fallback for a missing stemmer result. Its classes are the fibers $M^{-1}(s)$. Let $C_1, \dots, C_s$ be nonempty, pairwise disjoint intended protected

classes and define $P_Q = \bigcup_i C_i$. Choose distinct fixed labels t_i , with $T_Q = \{t_i\}$, and require

$$M(w) = t_i \iff w \in C_i, \quad w \in V. \quad (1)$$

The forward and reverse implications are independent requirements. Positive assignments implement membership; negative boundaries forbid outsiders. An exclusion from C_i does not assert that a form is linguistically unrelated in every context, nor does it specify the best class for that form elsewhere.

For original positions $x_{d,j}$, expected support is $\mathcal{P}_i = \{(d, j) : x_{d,j} \in C_i\}$. Equality of those positional sets yields equal TF and DF. Query-side class matches additionally require the prescribed labels for every protected query surface, including any absent from the corpus. With unchanged document lengths, query-position multiplicity, scorer arithmetic and tie-breaking, it yields identical BM25 rankings for queries accessing only checked classes. Paper 8 proves the label-renaming and retrieval-equivalence result [3]. Matching an aggregate effectiveness score alone is insufficient: distinct runs can have the same mean.

The observed inventory and the unlimited input language are different domains. A scan can establish Eq. (1) on V ; it cannot, without an additional compiler proof, establish it for every future string. Likewise, multiple-output or context-sensitive analysis needs a richer semantic object than one equivalence relation on surface strings. The compiled scalar policy is an intentional restriction, not the most expressive possible lexical system.

III. FROM A STANDARD DICTIONARY TO A TWO-SIDED POLICY

A. The Standard Model Is the Substrate, Not the Limit

The deployment starts with the compiled standard model M_0 . It is reopened as a mutable builder, enriched with a task-specific extension, and materialized as a new immutable model. The original lexical source need not be reconstructed. This is a model update, not a replacement of the stemming algorithm and not a claim that the original binary is edited in place. The standard model supplies general behavior outside the explicit extension; whether rebuilding leaves every unprotected output unchanged is a separate question requiring a broader audit.

The model family can be written schematically as

$$\begin{aligned} M_\theta &= \text{Build}_r(\text{Extend}(\text{Reopen}(M_0), P^+, P^-)), \\ f_\theta(w) &= \text{Lookup}_\ell(M_\theta, w), \end{aligned} \quad (2)$$

where r denotes the reduction configuration and ℓ the lookup policy. The parameterization is predominantly *discrete and structural*: assignment sets, excluded memberships, and execution settings, rather than a single continuous stemming-strength knob. Only compatible, successfully checked extensions belong to the deployable family. The notation does not assert that every imaginable partition is reachable.

There are two different starting points in the experiment. PyStemmer initializes the explicit partition search; standard Radixor is the compiled host into which the selected policy is inserted. The standard-to-adapted comparison evaluates the benefit of that deployed extension. It is not an ablation of

learning from Radixor alone or a claim that the learned policy contains no information from the PyStemmer prior.

B. Positive Assignments and Negative Boundaries

A positive extension is a partial map $P^+ : w \mapsto s$. Assigning several surfaces the same s makes their occurrences contribute to the same retrieval term. Relative to the standard model, these assignments can also separate previously conflated forms by choosing different outputs. “Positive” therefore means *prescribed membership*, not “more aggressive than the baseline”.

A negative extension identifies surfaces that must stay outside protected classes. In this experiment those requirements come from auditing the realized positive policy, not from independently optimizing a second relevance loss. They are encoded using ordinary patch commands that produce a reserved output, such as $w||\mu$, together with the secondary guard detailed in Section VI. The word is retained in the index, but its occurrences no longer enter the protected class. This is not stopword removal, a negative TF, or a claim that the surface is linguistically unrelated to every protected member.

The distinction is operationally important. Deleting an exact rule can simply expose an inherited generalization; it does not mean “never map this word to that class”. Conversely, a positive assignment may be correct for every explicitly checked form while generic rules admit additional forms. Both directions of the policy need to be represented, and the compiled output—not the extension file alone—must satisfy the required memberships.

C. Applying and Persisting the Extension

The following sequence summarizes the implementation boundary without inventing a replacement for the unavailable original sealer:

- 1) **Reopen the standard artifact.** The documented Python interface uses `Stemmer.to_builder()` or a `TrieBuilder` opened from a compiled model. Reconstruction preserves the compiled rules, including contracted generalizations; it does not recover the original insertion history [1].
- 2) **Install positive assignments.** Insert the selected surface-to-output mappings, resolving competing candidates by the intended scalar policy. The documented `add` operation makes a candidate dominant while retaining alternatives; `set` replaces exact-node candidates. Their difference matters when a node must remain multi-valued. With documented default `store_original=True`, both operations also register an identity assignment for the canonical target. A reproducible extension must pin that setting and include those additional assignments in its audit, rather than assuming that only the supplied source node changes [1].
- 3) **Make specific rules reachable.** Build and load the customized model with `lookup="last"`, as in this experiment. This continues beyond a shallower accepting generalization to a more-specific rule. The default “first” policy may short-circuit before the extension.

The lookup configuration must travel with the deployed artifact, not be assumed from its filename [1].

- 4) **Add and verify negative boundaries.** Audit protected-class support over the declared vocabulary, install barriers for outsiders, rebuild, and repeat the checks. The reported barrier has a dominant append-one-marker command and a distinct append-two-marker guard. A single identity or marker override is not a reproduction of that structural guard. Namespace, candidate order, and positive memberships must be checked after rebuilding and reloading.
- 5) **Persist and deploy a compatible pair.** Save the rebuilt compiled model and apply its scalar output contract consistently to indexing and queries. The source dictionary need not be rebuilt from scratch, but changed term assignments require a compatible rebuilt or updated index. Publish the model, configuration, and index version together.

For a simple non-benchmark example, an identity assignment can protect a brand from a generic suffix rule, while a shared canonical assignment can normalize two desired variants. Such exact customization illustrates the interface; it does not establish that the same linguistic choice improves MIRACL. The present barrier construction additionally controls the reduction predicate and checks full protected-class support. The current API documentation is pinned for this explanation, but the original experimental executable and full barrier inventories are not supplied.

The positive and negative layers are therefore *complementary controls*, not independent effect terms. Compilation, document frequency, and query scoring can couple their consequences. The practical advantage demonstrated here is that both controls can be integrated into one standard-model extension and inspected as one deployable policy.

IV. MATERIALIZING THE ORACLE AS A RADIXOR POLICY

A. Exact Positive Overrides

Radixor represents transformations as compact patch commands stored in a backward trie [2], [4]. The current customization API can reopen a compiled model, add or replace exact word-to-stem rules, and build a model with most-specific lookup. We use this mechanism to assign every surface in an oracle component to a stable synthetic component label. The label is only an implementation identifier; retrieval semantics depend on component membership, not the spelling of the label.

The inherited report records verification of 54,745 explicit surface mappings in the compiled model. This supports realization of the prescribed positive mapping on those checked forms, not yet realization of the complete partition: the reverse implication excluding outsiders remains to be tested. The mapping inventory and model bytes are not available for independent replay here.

B. The Unsealed Surprise

The inherited first full-corpus benchmark of the materialized model does *not* recover the reported explicit-oracle objective.

	Intended	Unsealed	Sealed
Members of C	a, b	a, b, x	a, b
$d_1 = [a, a, x]$: TF	2	3	2
$d_2 = [b]$: TF	1	1	1
$d_3 = [x]$: TF	0	1	0
$df(C)$	2	3	2
$CF(C)$	3	5	3

Positive rules: $a, b \mapsto C$. Barrier: $x \mapsto x||\mu$.

Fig. 2. Illustrative example, not a measured MIRACL class. Correct positive rules do not stop x from entering C through generalization. The barrier restores the intended DF and TF without deleting any token positions. It specifies a retrieval boundary, not a claim that x is linguistically unrelated to a or b .

Its recorded approximate value is

$$J_{\text{unsealed}} = 0.545322, \quad (3)$$

close to PyStemmer and below the explicit oracle trajectory near 0.550. Representative unsealed metrics are nDCG@10 0.366584, Recall@100 0.826964, Recall@1000 0.921461, and AP@1000 0.312445.

Correct positive mappings direct attention to words that were not explicitly overridden, provided index construction and scoring are otherwise aligned. The compiler-semantics study [5] describes exactly such a possibility: an accepting leaf generalizes command selection to additional continuations. If an unseen or unpatched surface traverses a generalized region that now contains an oracle-label command, it can enter an oracle class even though it was not a member of the explicit partition.

This produces a precise hypothesis: the compiled model has *generalization leakage* into protected query classes (Fig. 2). Here leakage means unintended class membership; it is distinct from the explicitly acknowledged reuse of development labels for selection and evaluation. The inherited experiment reports a posting-level test of that hypothesis; the detailed inventories are not present in the supplied package.

V. DIAGNOSING GENERALIZATION LEAKAGE

A. Expected and Actual Query-Class Postings

For each query-relevant oracle class C , let $P_E(C)$ be the expected set of document–position pairs under the explicit oracle projection and $P_A(C)$ the actual positional set in the full index built from the compiled model. A mismatch can have three forms:

- 1) expected-only documents, meaning the runtime model lost oracle matches;
- 2) actual-only documents, meaning the runtime model added documents;
- 3) TF mismatches in documents appearing on both sides.

The audit also stems the complete observed surface vocabulary through the unsealed model and records every unpatched surface that maps into a protected oracle target.

The inherited audit reports that the projection store and the benchmark originally assigned internal document IDs under different shard orderings: lexicographic versus numeric. Its

TABLE I
REPORTED CORRECTED LEAKAGE AUDIT OF THE UNSEALED MODEL.
THESE ARE INHERITED OBSERVATIONS; RAW PER-CLASS AND POSITIONAL
INVENTORIES ARE NOT INCLUDED.

Quantity	Value
Query-relevant oracle classes	1,684
Posting-identical classes	625
Mismatched classes	1,059
Observed corpus surfaces	6,046,942
Verified positive corpus surfaces	54,617
Positive mapping mismatches	0
Leakage surfaces	11,291
Leakage collection frequency	3,040,762
Leakage DF sum	2,602,771
Expected-only documents (reported)	0
Actual-only documents (reported)	2,520,134
TF L_1 difference	3,040,762
TF-mismatch documents (reported)	52,252

correction derives a shard-offset permutation from the MIRACL files before comparing postings; the permutation itself is not supplied for replay. This changes no DF/CF aggregate; it only makes document-level intersections meaningful.

B. Leakage Accounting Identity

Table I summarizes the reported corrected audit. Its account is internally consistent at the supplied aggregate level; the classwise claims below remain inherited observations. Of 1,684 query-relevant oracle classes, 625 have exactly equal postings and 1,059 differ. The audit inventory contains 6,046,942 surfaces. Its 54,617 verified positive corpus surfaces and the 54,745 explicit policy mappings are differently scoped reported counts; the supplied summaries do not identify the 128-form difference. Only 11,291 unpatched surfaces leak into protected classes, but they account for 3,040,762 term occurrences.

The following identities describe the reported audit. The supplied report does not establish cross-class deduplication of its document totals. They must not be interpreted as counts of globally distinct affected documents. First,

$$P_E(C) \subseteq P_A(C) \quad (4)$$

for every protected class in the reported additive posting comparison. Zero expected-only documents establishes document-support inclusion; TF inclusion additionally uses the verified positive assignments and the position-level leakage account. Document-support inclusion alone would not establish TF equality or a nonnegative TF difference.

Second, for every one of the 1,684 query classes,

$$CF_A(C) - CF_E(C) = CF_{\text{leak}}(C). \quad (5)$$

The inherited account reports a zero residual after subtracting inventoried leakage in every class; the aggregate files alone cannot establish that classwise assertion. Globally,

$$\sum_{C,d} |tf_A(C,d) - tf_E(C,d)| = 3,040,762, \quad (6)$$

which is exactly the total leakage collection frequency. Furthermore,

$$CF_{\text{leak}}(C) = 0 \quad (7)$$

is equivalent in this audit to posting equality: all 625 zero-leakage classes are equal, and all 1,059 positive-leakage classes differ.

If its positional premises are verified, this is a conservation-style explanation of the structural discrepancy, stronger than correlation alone. The reported positive agreement and additive excess support that explanation, but aggregate agreement alone cannot locate the excess positions or independently certify the cause of a ranking change.

C. Why a Small Leakage Set Can Matter

The 11,291 leakage surfaces are only about 0.187% of the 6.05 million observed surfaces. Their effect can nevertheless be large because leakage is concentrated. The inherited audit summary reports that the highest-mass 20 protected classes account for roughly 48.5% of leakage CF, and the highest 100 for about 87%; the per-class inventory required to reconstruct that concentration curve is not included. Because BM25 IDF is a nonlinear decreasing function of DF, localized added mass can change the relative strength of query-term evidence. Thus global vocabulary percentages are poor proxies for retrieval impact.

This observation suggests a specific explanation of the unsealed result: even correct positive assignments can coexist with generalized outsider matches that change query-class statistics. The reported audit attributes the discrepancy to this mechanism. A controlled replay is still needed to isolate its contribution to effectiveness from checkpoint, index, and scoring differences; correctness of the assignments is not a judgment of their linguistic quality.

D. A Conditional Exactness Result for Protected Classes

The leakage account can be stated as an explicit theorem with testable assumptions. Let $C_1, \dots, C_s$ be pairwise disjoint intended classes on a fixed observed vocabulary V . Let $t_1, \dots, t_s$ be distinct runtime labels. Assume every intended member $w \in C_i$ satisfies $M(w) = t_i$, every corpus position emits one label, and no surface outside C_i is meant to enter that class. Define

$$L_i = \{w \in V \setminus C_i : M(w) = t_i\}. \quad (8)$$

Proposition 1 (additive leakage and sealing). Under those assumptions, for every class and document,

$$tf_M(t_i, d) = tf_E(C_i, d) + \sum_{w \in L_i} tf(w, d). \quad (9)$$

Hence classwise excess CF and the summed TF L_1 difference equal the occurrence mass of the corresponding leakage sets. If a recompiled model preserves all intended positive assignments and excludes every L_i and any newly introduced outsider from all protected labels, it restores the protected classes' positional postings on V .

Proof. The surfaces producing t_i are the disjoint union $C_i \cup L_i$. Partition the original positions of each document according to those two sets. Counting positions yields Eq. (9). Every difference is nonnegative, so taking absolute values leaves the differences unchanged, and summing over documents gives the CF identity. With no outsiders remaining and the positives preserved, the two positional sets are equal. $\square$

The theorem makes clear why zero expected-only *documents* is not sufficient: an expected occurrence could be lost within a document that still retains another expected match. Positive assignment preservation and positional accounting rule out that case. It also makes clear why matching total excess CF alone is insufficient to identify a cause: a wrong per-document allocation can have the same total. The reported corrected audit supplies a stronger classwise account; its raw positional evidence remains a required archival artifact.

The retrieval-equivalence result in the companion paper and Proposition 1 together imply ranking equality for the audited queries only when all their accessed classes, multiplicities, document statistics, and tie semantics meet the contracts. They do not establish that the supplied final model exactly matches an unavailable terminal oracle checkpoint. Nor do they extend a scan of V into a proof over all possible future strings.

E. An Operational Audit Must Preserve Document Identity

The document-ID permutation is part of the audit, not an incidental file-format detail. Before comparing postings, a validator should map both stores to a common external passage identifier or verify a one-to-one permutation from each internal ID space. It should then check corpus cardinality, identifier uniqueness, original lengths, and a logical digest independent of physical shard order. Equal aggregate CF can survive an incorrect permutation while document-level intersections become meaningless.

The three mismatch categories should be reported per class before any global summarization. Their document totals may overlap across classes; a sum over class–document pairs must not be called a count of unique affected documents. Similarly, surface DF sums count a document once per surface and are not the DF of the union. The leakage DF sum 2,602,771 and reported actual-only document total 2,520,134 answer different questions and need not agree.

A compact archival audit row would contain the protected label, intended member count, unintended member count, expected and actual DF/CF, support-only mismatches, TF L_1 residual, and the identities of relevant source inventories. A deterministic aggregate hash can verify repeatability, but the row-level inventories must remain available to explain a failed check. No synthetic per-class distribution is inferred from the supplied totals.

VI. CONFLATION BARRIERS AND FIXED-POINT SEALING

A. Negative Policy Constraints

The leakage audit supplies a natural negative policy: each inventoried leakage surface must be kept out of the protected query classes. Mapping every such word to an independent synthetic label is possible, but need not share the same compact

edit shape. The construction below deliberately makes that shape identical across barriers. Radixor’s patch representation permits command sharing in this construction; no serialized-size comparison with alternative boundary encodings is available.

Let μ be a Unicode private-use marker represented here as U+E000. The benchmark tokenizer accepts only Letter, Mark, and Number categories; therefore μ cannot occur in an ordinary normalized token. For a boundary surface w , define the primary barrier transform

$$B(w) = w\|\mu. \quad (10)$$

Different words remain distinct because $w_1\|\mu \neq w_2\|\mu$, but all barriers share the same edit shape: preserve the input and append one marker. In a backward patch-command representation this can be shared as a common insertion program rather than a per-word reconstruction.

An implementation detail matters. Radixor’s uniform-subtree contraction can replace a region of single-valued leaves by an accepting generalized leaf. If every barrier leaf stored only the same append-one-marker command, the compiler could legally generalize that command upward and apply it to protected positive surfaces. The reported sealing construction therefore stores two globally shared values at each barrier node:

$$B_1(w) = w\|\mu, \quad (11)$$

$$B_2(w) = w\|\mu\|\mu, \quad (12)$$

with the first made dominant. Provided both distinct values survive the chosen reduction up to the uniformity check, the second value prevents that node from satisfying the single-value predicate. Scalar lookup must separately be verified to return $B_1(w)$. Thus only two generic patch-command shapes are used by this construction; neither successful pinning under every reduction mode nor constant total storage follows from their count.

A surface-specific barrier can also remove useful matches for an unprotected query, because separately blocked forms receive distinct outputs rather than a shared lexical target. Exclusion from a protected class is therefore neither proof of a linguistically false relation nor a guarantee that all other queries improve.

The marker must be treated as an internal output label. It is added *after* tokenization and must not be removed by downstream normalization or retokenization. Its exclusion from input tokens is not, by itself, proof that arbitrary generic outputs can never collide with it; the label namespace must be reserved and the protected-class exclusion checked. The measured claim below is for the scanned inventory. The two shared commands also do not make total barrier storage constant: the compiler must still represent the boundary surfaces.

This guard is an implementation technique, not a general requirement of the policy abstraction. A future compiler could expose an explicit “pinned exact rule” primitive and represent the same negative constraint more directly.

B. Sealing Procedure

The sealing algorithm takes the positive compiled oracle, the 11,291 leakage surfaces from the corrected audit, and the 1,684 query-relevant protected classes. It performs four checks:

- 1) verify the source protected partition before modification;
- 2) apply the two-command barrier pair to every leakage surface;
- 3) verify every intended corpus and query member against its prescribed class label under effective scalar lookup after rebuilding and reloading;
- 4) scan the declared 6,046,942-surface corpus inventory for outsiders reaching a protected label; check query-only positive forms separately rather than assuming that a corpus scan covers them.

If new leakage appears, another barrier round is generated. The process stops at a fixed point or a configured round limit.

For the reported English model, the inherited account states that the initial 11,291 barriers suffice. The recorded first post-sealing scan covers all 6,046,942 inventoried corpus surfaces and reports

$$|L_1| = 0, \quad CF(L_1) = 0. \quad (13)$$

Preservation of all 1,684 protected query classes is also reported, without a second closure layer. The supplied final benchmark manifest identifies the sealed model by SHA-256

```
8f942746c8cf6c92fa71f3e0f3759698
4296f09884022db6fdbba74beda88a13e.
```

The resulting runtime architecture is therefore

$$M_{\text{deploy}} = M_{\text{generic}} \oplus P^+ \oplus P^-. \quad (14)$$

The generic model supplies the fallback outside the task-specific policy. The protected-class audit does not certify that all other outputs are unchanged by compilation.

C. Two Fixed Points with Different Meanings

A sealed vocabulary and a jointly tuned retrieval system solve different problems. The outsider closure condition is $L_V(M) = \emptyset$, but the full certificate also requires labelled positive agreement:

$$A_V(M) \wedge L_V(M) = \emptyset, \quad (15)$$

where $A_V(M)$ means $M(w) = t_i$ for every intended corpus or protected query member $w \in C_i$. Without A_V , a model that loses or interchanges protected classes can pass an outsider-only scan. Under A_V , outsider closure states that no unintended surface in V belongs to a protected class. It says nothing about whether another intended partition would improve retrieval, or whether a different BM25 point would be preferable. Conversely, an empirically stationary objective says nothing about whether the compiler silently changed the selected partition.

For fixed vocabulary V , a barrier loop has a simple conditional termination argument. Suppose each nonterminal round permanently excludes at least one newly found outsider, preserves all previous exclusions and intended positive assignments, and never reintroduces them through recompilation. Then at most $|V \setminus P_Q|$ such additions are possible. This is a finite monotonicity argument, not a termination guarantee for an arbitrary compiler. If recompilation can undo earlier exclusions or alter protected labels, those premises fail; cycle detection, a round limit, and full revalidation remain necessary.

The reported one-round success is stronger than merely reaching a round limit but narrower than proving general convergence. It states that the checked English inventory was closed after the seeded barriers were applied. The difference of 16 between that inventory and the final no-stemming census prevents silently treating the two vocabularies as identical.

D. Why the Secondary Barrier Command Matters

The guard relies on a specific compiler predicate: a node still storing two distinct values at the uniformity check is not uniformly single-valued. Any earlier pruning or dominant-only projection that removes the secondary value invalidates this structural argument. The primary value must also remain first under the actual ranking and dominance rules, and the lookup must resolve the intended exact node. The compiler-semantics study distinguishes ranked, unordered, and dominant value observables and distinguishes command selection from patch execution [5]. A deployer must test the barrier under the exact reduction mode and API used here; exposing all alternative outputs would no longer be the same scalar contract.

A small regression suite should therefore include a protected positive form, its boundary neighbor, an unseen form sharing their traversal suffix, and a form colliding with a reserved output label. It should inspect results before and after reduction, serialization, and reload. Equal selected commands do not necessarily yield equal output strings because a patch executes on its input. Two shared append-command shapes do not imply two total trie nodes or constant model size.

Identity-output barriers are an alternative only when the unchanged surface is itself outside every protected output namespace. Otherwise $w \mapsto w$ may still collide with a protected target. An exact-exception side map, a pinned-node compiler primitive, and reserved marker outputs are therefore different engineering choices implementing the same negative membership constraint. Their memory, lookup, and maintainability costs require a matched comparison; the present result does not certify one as optimal.

E. Deployment, Versioning, and Vocabulary Growth

A retrieval policy should be frozen together with its normalizer, tokenizer, generic model, positive assignments, barriers, compiler mode, lookup contract, output namespace, and index build. The checksum of a compiled file identifies only one part of that bundle. A query analyzer using another policy version can break the class correspondence even when both versions are individually correct.

The corpus vocabulary may be used for an unlabeled closure audit without using held-out relevance judgments, but that is a transductive deployment assumption: it validates a known collection, not arbitrary future vocabulary. For a growing index, new surfaces require an admission audit against protected labels, followed by recompilation and reindexing wherever outputs changed. Removing or stripping the private-use marker downstream is a semantic change, not harmless text cleanup. A model update and an index update should therefore be deployed together, with rollback to a compatible pair.

There are also two senses of “leakage” in this study. *Compiler leakage* is unintended membership in a protected lexical class. *Evaluation leakage* is use of information reserved for assessing generalization during model selection. Vocabulary sealing repairs the former; it does not repair the latter. This distinction is essential when designing the held-out extension.

VII. HISTORICAL ADAPTABILITY IS NOT ONE CAPABILITY

A. Comparison Criteria and Source Boundaries

We compare four separate properties: the native transformation language, how rules or groups are selected, where user corrections enter, and what is verified after deployment. Historical priority additionally requires an earlier disclosure of the same relevant mechanism: a later system with an overlapping capability is not evidence against an earlier publication. A richer primitive language is not automatically a better learner; a finite lookup table is not automatically less expressive on its recorded inventory. Table II therefore describes operations rather than assigning a single sophistication score. Its background rows are not used to infer full-system parity. The closer XSTEM, HPS and CST comparisons below are grounded in their method descriptions, with implementation corroboration where available. “Not specified” means absent from the cited mechanism or interface description, not proved impossible.

B. XSTEM and the Older Egothor Instruction Language

XSTEM’s paper specifies a suffix rule by a matching suffix, a number of terminal characters to remove, and a literal suffix to append. The longest matching suffix is found in a reverse trie; optional successive modules allow different stemming strengths [14]. This is not merely an exact dictionary, but it is also not equivalent to an arbitrary edit program as a native rule representation.

Egothor’s historical executor provides deletion, insertion, substitution and a skip instruction, allowing edits separated by retained source characters. This is verifiable in Lucene 4.10.4 at commit 69175ad4fe8d (full identifier in the source citation), dated 3 March 2015 [8], [16]. The corresponding Stempel facade selects a trie command and executes it; its documentation explicitly distinguishes the Polish packaged tables from the language-independent algorithm [9]. The older scientific lineage is documented in the 2001 paper and Lucene’s references [2], [17].

For a valid nonempty input w , define a terminal action

$$T_{n,r}(w) = w[0 : |w| - n]r, \quad 0 \leq n \leq |w|, \quad (16)$$

where n is fixed and r is a literal replacement. Both sides of the following embedding use the same atomic symbol and length convention. The historical Java executor counts UTF-16 code units; this is not automatically the code-point convention of another runtime. The witnesses below use ASCII, where those units agree. Suffix matching and module selection are separate from this action.

a) *Proposition 2 (terminal-action embedding)*.: Every action in Eq. (16), for valid lengths and supported nonempty outputs, can be expressed by an Egothor edit program: delete the final n characters, then insert the characters of r in the order required by the executor. With the historical right-to-left cursor, appending the reversed character sequence as insertion commands produces r in its normal order. Zero deletion or an empty replacement simply omits that part. Counts larger than one instruction’s representation can be divided into valid runs. This embeds the action, not an entire multi-pass XSTEM model or its learning procedure.

The inclusion is strict at the level of one native rule action that must serve a family of inputs. Consider the synthetic family $pcuz$, where p is any prefix, $c \neq x$, and u is one character chosen from at least two distinct symbols. The patch $-bRx$ skips the final two characters and substitutes x for the preceding character, producing $pxuz$. A single $T_{n,r}$ cannot implement that family. For $n < 3$, the original c survives. For $n \geq 3$, a fixed literal replacement cannot retain the differing value of u . Distinct suffix rules can enumerate the cases, and a multi-pass system supplies further composition; neither possibility erases the native instruction distinction. This is a symbolic construction, not a linguistic example or throughput benchmark.

b) *Single-pass preservation and a compactness witness*.: If a single suffix pass keeps the same keys, eligibility checks and longest-match choice, replacing each terminal action by its embedded patch preserves that pass’s outputs on the valid input domain. This assumes no reduction that changes the selected commands; it is not an equivalence claim for an arbitrary multi-pass pipeline.

The native-action distinction also has a precise compactness consequence. On inputs cuz with $u \in A^k$, $|A| \geq 2$, one patch can skip the final $k + 1$ symbols and replace c by x . A single-pass cut-and-append implementation that changes the initial c must delete the whole input and reproduce xuz as a literal. It therefore needs at least $|A|^k$ distinct literal replacement actions for this finite family. This is a symbolic rule-count separation under those restricted primitives, not a measured model-size result or a lower bound for multi-pass XSTEM. It identifies the benefit of retaining source material rather than enumerating it.

Thus *terminal cut-and-append is a special case of patch execution*, whereas patch execution can reuse unchanged internal source material within a single command. This supports describing XSTEM’s elementary transformation model as narrower than Egothor’s. It does not prove that the set of functions realizable by arbitrary XSTEM pipelines is a strict subset of the functions realizable by every Radixor configuration, nor that the larger instruction set always produces a smaller or faster model.

C. A Specific and Material Prior-Art Omission

XSTEM’s related-work and comparison sections discuss Lovins, Dawson, Porter, Krovetz and Paice. Its 13-item bibliography does not cite Galambos, Egothor or Stempel; Lucene is nevertheless named as an available stemming distribution [14]. The accessible Lucene release above predates the first XSTEM

TABLE II

CAPABILITY COMPARISON. ROWS DESCRIBE THE CITED HISTORICAL ARTIFACT OR PUBLICATION, NOT EVERY POSSIBLE EXTENSION OF ITS CODE. NO CROSS-SYSTEM PERFORMANCE EXPERIMENT IS REPORTED.

System or lineage	Native rule / representation	Selection and correction	Relation to Radixor’s deployment contract
Krovetz / KStem [6]	Lexical resources plus morphological processing	Curated exceptions and direct conflations	Important correction precedent; not the same patch-model reconstruction or protected-posting audit.
Xu–Croft [7]	Corpus-conditioned conflation classes	Distributional evidence changes class grouping	Adapts the relation, rather than only a global strength setting; no equivalent compiled-model update contract established by the evidence reviewed here.
Galambos / Egothor; distributed via Stempel [2], [8], [9]	Trie-selected sequences of deletion, insertion, substitution and copying/skipping	Edit programs inferred from word-stem examples; trie reduction and generalization	Direct predecessor, not an unrelated competitor. Patches and data-driven stemming predate Radixor and XSTEM. Modern builder and audit details are not backdated.
Lucene override filter [10]	Exact token-to-output map before a downstream stemmer	Editable assignments; a matched token is protected from further stemming	Can encode any finite scalar target map with full coverage. Storage, fallback and rebuild engineering differ; no absolute finite-domain expressivity advantage follows.
HPS (2014/2015) [11]	Learned terminal-deletion length	Corpus-derived supervision; controllable clustering threshold	Restricted output family, not a general patch executor; see Proposition 3.
CST affix rules (2009) [12], [13]	Ordered wildcard captures and replacements	Supervised corrections in an exception tree	Genuine internal-rewrite comparator; implicit negation is not the same claim as a checked protected-posting invariant.
XSTEM [14]	Longest-suffix trie; terminal cut-and-append; separate optional passes	Editable exemplar rules, proper-name protection and exact exceptions	Native action is a restricted patch pattern; generalization and local exceptions are genuine overlap. No protected-posting closure procedure is specified in the paper.
Selective stemming [15]	Query-dependent choice of stemming treatment	Learned selector using query and frequency features	Adapts when stemming is applied, not the scalar word-to-stem relation itself. Query conditioning is outside the present context-free policy.
Radixor, this construction	Reusable compiled patch model with scalar/alternative lookup	Positive assignments and negative membership boundaries installed in the standard model	Combines realization, observed-vocabulary exclusion checks, TF/DF support audit and persistence; open-vocabulary closure and matched-system efficiency remain untested.

submission on 9 May 2022, and the Galambos publication dates from 2001 [2], [16].

This omission matters technically. A comparison of configurable, generalizing trie-selected transformations should address the earlier example-trained patch-command design, not just fixed suffix strippers and whole-word lookup. The published comparison is therefore incomplete with respect to a directly relevant predecessor. XSTEM’s stated English focus narrows its evaluation scope, but does not remove a language-independent transformation architecture from the relevant methodological prior art. This conclusion concerns the coverage of that paper, not the author’s motives, derivation of code, or overall competence.

XSTEM remains relevant here as a later, simpler suffix-rule comparator, not as evidence that it anticipated Egothor or supplies an equivalent full deployment mechanism. Conversely, its smaller native action vocabulary does not invalidate its practical usefulness or show that the entire IR literature failed for the same bibliographic reason.

D. HPS: Learned Selection Within a Restricted Output Family

HPS here denotes Brychcín and Konopík’s *High Precision Stemmer*, not the unrelated hierarchical Persian stemmer. Its experiments cover six European languages; the retrieval task covers four. Section 4.2 defines the predicted class as the suffix

length to delete, $y \in \{0, \dots, M\}$. Cluster longest-common-prefix outputs supply supervision. The reported configuration uses $M = 3$ and two passes; its clustering threshold controls aggressiveness, but was not tuned to the evaluation task [11, Secs. 4.2, 5.1–5.2, 5.5]. Thus multilingual learning and unseen-word prediction are real capabilities, while the native action is terminal deletion rather than arbitrary string rewriting.

a) Proposition 3 (prefix-output restriction).: For any finite sequence of valid suffix deletions, the final output is a prefix of the initial input. Write

$$w_{t+1} = w_t[0 : |w_t| - m_t], \quad 0 \leq m_t \leq |w_t|. \quad (17)$$

Each w_{t+1} is a prefix of w_t ; transitivity proves the assertion by induction. This holds even if a perfect oracle selects every m_t . Additional learned parameters or deletion passes cannot remove this representational restriction. The inference is ours, derived from the published action, not a performance claim made by HPS.

The restriction concerns relations as well as preferred output labels. Consider the synthetic inventory $V = \{ab, cd, ef, gh\}$ and desired distinct classes $\{ab, cd\}$ and $\{ef, gh\}$. Members of either pair have no shared nonempty prefix. If empty output is disallowed, the first required merge is impossible. If it is allowed, both pairs must map to the same empty string, contradicting their required separation. An explicit unrestricted target map can instead assign q to the first pair and r to the

second. Patch execution can realize those assignments; whether a particular trained and reduced Radixor model does so must still be verified. This witness proves a limitation of prefix-only output relations, not that arbitrary class learning is solved by Radixor.

This is a stronger technical distinction than a disputed language count. HPS is relevant to learning *which* permitted cut to choose, not evidence that its action language matches Egothor’s. Its 2014 online publication and 2015 issue date also cannot anticipate the documented 2001 Galambos lineage. Its results neither establish nor refute equal-budget performance against an adapted Radixor model.

E. CST: A Genuine Multi-Affix and Exception-Rule Comparator

Jongejan and Dalianis’s 2009 CST method learns from full-form–lemma pairs. Its wildcard rules preserve matched variable spans between prefix, internal and suffix replacements; corrective children and ordered siblings determine which rule fires. Its intrinsic evaluation covers ten languages; search evaluation is future work [12, Secs. 3–5, 9]. These are genuine multi-affix capabilities, independently of the attribution deficiency examined below.

The published operation is corroborated by the pinned CST executor. Its `samestart`, `sameend` and `substr` routines locate prefix, suffix and internal boundaries; `rewrite` reassembles replacement literals with captured input spans [13]. In abstract notation, its rule form permits

$$a_0 u_1 a_1 \cdots u_m a_m \mapsto b_0 u_1 b_1 \cdots u_m b_m, \quad (18)$$

where u_i are the captured spans. Unlike a literal cut-and-append action, the replacement need not enumerate their contents. Current project documentation separately identifies the older suffix trainer and the more general `affixtrain` trainer [18]. These current sources corroborate operations; their later interface details are not backdated to 2009.

This is substantive overlap with example-driven, generalizing internal rewrites and error correction. It is not a proof of complete equivalence. Variable-length captures and fixed-count patch skips impose different matching/execution contracts; no inclusion theorem for the complete CST and Egothor systems is established here. Likewise, an ordered-path failure condition controls *rule eligibility*, whereas Eq. (1) constrains *output-class membership over a declared inventory*. The former does not by itself prove the latter, although an additional checking layer could impose it. The cited descriptions do not establish Radixor’s complete reopen–extend–persist–posting–audit procedure in CST. An undocumented feature is not a proved impossibility.

The date matters independently of this genuine technical relevance: CST 2009 is later than Galambos 2001. It does not displace that precedence. Nor does genuine overlap in internal rewriting establish either complete-system parity or independent priority. The scientific merits of CST’s own rule learner must be assessed separately from its treatment of the earlier algorithm.

F. Algorithmic Attribution Is Not Software Attribution

Stempel is the Polish-resource distribution and integration of the Egothor engine, not a separate inventor of that engine. The distribution’s overview distinguishes Galambos’s implementation from Bialecki’s compilation and cites Galambos’s scientific work; the facade explicitly identifies a language-independent Egothor core [9], [17]. Algorithmic attribution in this comparison therefore belongs to Galambos/Egothor [2], [4], while Stempel is credited for the distributed resource, integration and its evaluation. The Polish training tables do not limit the engine’s transformation language to Polish.

CST’s related work cites Stempel as Bialecki (2004), whereas its conclusion uses “Stempel algorithm (Bialecki 2008)”; its bibliography lists 2004 and no Galambos or Egothor entry [12, pp. 146, 152–153]. The year mismatch is a concrete bibliographic inconsistency. More importantly, citing the distribution does not identify the original general patch-command methodology. The related-work account is incomplete in that algorithmic attribution; the presence of a Stempel reference is not sufficient credit for the underlying technique.

This is not an objection to citing software or to Bialecki’s resource contribution. Software provenance and methodological precedence answer different questions and should both be represented. It also does not prove that CST’s captured-span learner is identical to Egothor’s, or that its authors derived it from Egothor. The defensible criticism is narrower and specific: the earlier mechanism is not properly identified or compared as such. XSTEM omits the lineage entirely; CST names the distribution but does not supply the primary algorithmic attribution. Neither observation licenses a field-wide causal explanation of retrieval outcomes.

G. Finite Tables, Learning, and the Correct Priority Claim

Xu–Croft’s earlier corpus-conditioned grouping is background evidence that learned conflation as a broad idea predates the present system [7]; it is not used here as an implementation-level equivalence claim. Selective stemming makes a different, query-conditioned decision [15]. Neither label decides the comparison between a reusable patch model and a complete deployment contract.

a) *Proposition 4 (finite exact-map realizability).*: For finite V and any scalar target $f : V \rightarrow S$, an exact override table storing $(w, f(w))$ for every $w \in V$, with output protection against downstream rewriting, realizes f on V . Proof is direct lookup for each member. The number of stored pairs is at most $|V|$. Lucene’s override filter documents the required override-and-protect behavior [10]. This includes both desired merges and separations on V ; behavior outside V , resource cost, and maintenance effort are separate questions.

The distinctive contribution is therefore not another suffix stripper or merely an editable word list. Radixor makes an existing generalizing patch model a reusable substrate for two-sided lexical policy and connects its compiled behavior to an explicit retrieval-evidence contract. Standard-model reconstruction, positive assignments, protected class boundaries, persistence and observed-inventory verification belong to one workflow. The examined XSTEM, HPS and CST descriptions

do not establish that combined contract. This is a specific systems advance over treating the shipped dictionary as the end of the method’s adaptability, not a claim that competing programs could never acquire additional capabilities.

The historical conclusion is equally specific and affirmative: these later comparators do not displace the Galambos/Egothor lineage. XSTEM uses a narrower elementary transformation; HPS learns within a prefix-output restriction; CST provides genuine captured-span rewriting but does not supply the missing original-method attribution. Those findings justify distinguishing capabilities rather than diluting the earlier contribution through a generic “adaptive stemmer” category. First-ever priority over all earlier morphology research, comparative adaptation cost, and superiority of every complete model require different evidence. Proposition 4 also precludes presenting arbitrary finite scalar mapping as exclusive to Radixor.

VIII. WHAT THE SHARED EXPERIMENT ESTABLISHES

The empirical case uses 799 English MIRACL development queries and 32,893,221 passages [3], [19]. Token positions, corpus size, document lengths and BM25 $(k_1, b) = (1.125, 0.3)$ are held fixed. The selected target is supplied by a PyStemmer-initialized partition search, not by learning exclusively from the standard Radixor relation.

The standard model obtains $J = 0.529014$, the positive-only realization approximately 0.545322, and the final sealed model 0.550119; PyStemmer obtains 0.545684. The explicit positive-only and audit values are inherited reported observations, while the final means are available as machine-readable summaries. The companion paper analyzes all five metrics, their query-level concentration, and the full-corpus index census [3]. The sequence motivates a deployment-fidelity explanation, but the missing terminal oracle checkpoint and raw audit inventories prevent independent replay of every intermediate equality.

The technically strong result is that standard-model adaptation is associated with a realized improvement at the declared scorer while preserving the core engine. It is not a measurement of a negative-only contribution, a randomized test of barrier causality, or a comparison with an equally configured Lucene/XSTEM/HPS deployment. Reusing the same development judgments for policy selection and evaluation further prevents interpreting the reported gains as unbiased transfer estimates [20].

The inventory itself exposes useful limits: 54,745 explicit positive mappings versus 54,617 reported positive corpus surfaces differ by 128; the audit vocabulary versus final no-stemming census differs by 16. These differences must be reconciled by member identities rather than assumed explanations. A hash without the sealed model bytes is an identity reference, not a replayable artifact. The present package preserves the available data and states those missing inputs explicitly.

IX. MATCHED COMPARISONS AND OPEN CLAIMS

A comparative test should freeze the target relation before comparing its realizations. Otherwise a better learner can conceal a worse deployment mechanism, or vice versa. Use

the same surface inventory, positive assignments, forbidden memberships, tokenization, ranker and query folds; distinguish scalar output equality, class-membership equality, and exact posting support.

Three levels answer different questions. First, test native operations: terminal rewrites, internal edits, identity preservation, ambiguous outputs and exception precedence. Second, test complete policy realizations using Radixor, exact overrides with a fixed fallback, and an XSTEM-style modular suffix implementation. Permit an explicit construction for each rather than forcing all systems into Radixor’s API. Third, compare independently selected policies with equal training information and optimization budgets. HPS and corpus-conditioned class induction belong at this learning level, not merely at the exact-override level.

Report effective artifacts and operational costs: construction time, changed records, compiled bytes, resident memory, lookup latency distributions, index rebuild cost, and disagreement counts on held-out forms. Keep vocabulary-seen/query-held-out and vocabulary-held-out tests separate. For negative control, report outsiders into protected classes as well as protected forms lost. Alternative-output APIs need set or ranked-list tests in addition to primary output checks. No number of passing scalar tests establishes the behavior of an untested alternative API.

For novelty, maintain a dated claim-to-source matrix that distinguishes predecessor mechanisms from current integration. The existence of a richer earlier patch language is established by primary source code; exclusive global priority is not established by failing to find a competitor. For historical causality, reproduce particular experiments under controlled interventions before extrapolating to the field. Independent reviewers can challenge these arguments, but their endorsement would not replace a proof, missing artifacts, or an experiment.

X. CONCLUSION

Radixor makes a standard compiled dictionary an adaptable deployment substrate: positive assignments and negative boundaries can be integrated through the existing patch engine, rebuilt, and checked against protected-class support. The posting identity separates intended assignments from unintended generalized memberships and provides a concrete verification obligation. The reported case demonstrates why that obligation matters even after the explicitly edited words are correct.

The historical comparison yields a positive technical conclusion, not only a novelty caveat: XSTEM’s elementary cut-and-append action is a restricted instance of the earlier Egothor patch executor. Its omission of that relevant predecessor leaves its published comparison materially incomplete. Generalizing lookup and example-oriented configuration were not innovations introduced by XSTEM. CST’s software citation likewise does not supply the missing primary attribution to Galambos/Egothor; its captured-span learner remains a technically distinct, later comparator. This conclusion concerns the native action and documented lineage, rather than every possible multi-pass function or a cross-system speed ranking.

Systems published before the current Radixor release supply overlapping parts of adaptive functionality; their later dates do

not displace the earlier Egothor lineage. The stronger contribution is not “the first adaptable stemmer” or exclusive finite-domain expressivity. It is an explicit, integrated relation between policy construction, compiled realization, negative boundary protection, and retrieval evidence. This is a substantive systems advance in the documented workflow without requiring a claim that competitors are incapable of equivalent behavior. Predictive, efficiency, and open-vocabulary advantages must each be tested under matched controls.

The companion empirical paper and this systems paper jointly shift the evaluative question from a name or default dictionary to an identified policy and its actual realization. They report one shared development case. Independent replication and matched competing implementations remain the decisive next tests.

a) Historical conclusion.: The named later comparators do not erase the earlier Galambos/Egothor contribution. Their verified overlap differs, and none of the examined descriptions establishes the complete two-sided deployment contract studied here. This supports an explicit, technically differentiated contribution rather than either universal exclusivity or an equally unsupported assertion that earlier alternatives already supplied everything Radixor provides.

REPRODUCIBILITY

The source package rebuilds the manuscript and retains the original final summaries and labelled intermediate observations. The general customization documentation is pinned to a repository revision; the original optimizer and sealer executable revisions, complete audit inventories, ranked runs and sealed-model bytes are not supplied. No new retrieval experiment, competing-system benchmark, external reviewer endorsement, or historical-priority certificate is claimed. Appendix B specifies the missing replay record.

APPENDIX A LOGICAL SEALING LOOP

The sealing procedure begins only after this abstract partition has been materialized. It deliberately separates *policy search* from *compiler closure*. Given the prescribed protected labels T_Q and the declared corpus/query inventory V , define

$$L(M) = \{w \in V \setminus P_Q : M(w) \in T_Q\}, \quad (19)$$

where P_Q is the set of surfaces that intentionally belong to protected query classes. The sealing loop is then

$$M_{i+1} = \text{Compile}(M_i \oplus B(L(M_i))). \quad (20)$$

After each compilation and reload, verify $A_V(M_{i+1})$ against the intended labels before checking outsiders. Reject the candidate if a positive member has moved, even when L is empty; only Eq. (15) permits success. A deliberate label renaming must instead supply a verified injective correspondence for positives, queries, and postings. The inherited English account reports success after the first barrier application.

This decomposition orders diagnostic checks; it does not uniquely identify a cause from one symptom. A search that finds no improvement may have exhausted its budget

rather than its representable policies. Incorrect positive outputs require checking assignment installation, lookup, reduction, and fallback. If positives agree but postings differ, align document identifiers, source positions, vocabulary, and index construction before attributing residual excess to outsider membership. Even with closure established, compare the exact oracle checkpoint, query aggregation, score arithmetic, and ties before claiming retrieval equality. The reported document-ID correction illustrates why these controls cannot be skipped.

APPENDIX B MINIMUM REPLAY AND AUDIT RECORD

A complete experimental release needs more than the article’s build inputs. It should pin corpus and qrels revisions, document-ID and length digests, normalized query text, tokenizer implementation, language algorithm and baseline model versions, the sealed model bytes matching its checksum, the compiler’s lookup and dominance configuration, and the retrieval/evaluation source revisions. The oracle additionally requires candidate edges, exact scoring budgets, seeds, tolerances, accepted-state checkpoints, and a terminal explicit partition. Rejected proposals are valuable for investigating objective plateaus and path dependence.

For compilation, retain every intended surface-to-class mapping, the two guard outputs and their selection behavior, the exact audited vocabulary, per-class posting comparisons, leakage inventories, and every closure iteration. Reconcile the 54,745 positive entries versus 54,617 reported corpus-positive surfaces, and the 6,046,942-surface audit versus 6,046,958-term final no-stemming census by listing the differing members. A plausible explanation is not a replacement for those identities.

For retrieval, retain query-level metric vectors and top-1,000 passage runs, together with score/tie conventions and the resampling implementation. These enable new confidence calculations, rank-transition matrices, and annotation-exposure diagnostics. The present package regenerates aggregate-derived tables, figures, and consistency checks; it explicitly lists which of these replay artifacts are not yet available.

REFERENCES

- [1] Radixor Project, “Customizing a dictionary in python,” 2026, pinned interface documentation; not an experiment executable pin. Accessed 20 September 2026. [Online]. Available: <https://github.com/leogalambos/Radixor/blob/daa8471570baa6e4eb04c93b6cdc94d4159c2d90/docs/python/customization.md>
- [2] L. Galambos, “Lemmatizer for document information retrieval systems in JAVA,” in *SOFSEM 2001: Theory and Practice of Informatics*, ser. Lecture Notes in Computer Science, vol. 2234. Springer, 2001, pp. 243–252. [Online]. Available: https://doi.org/10.1007/3-540-45627-9_21
- [3] —, “Where retrieval quality goes: Rethinking stemming in modern search,” ResearchGate, 2026, Radixor Research Series, Paper 8. Author-released research manuscript. [Online]. Available: <https://doi.org/10.13140/RG.2.2.22208.90887>
- [4] —, “Semi-automatic stemmer evaluation,” in *Intelligent Information Processing and Web Mining*, ser. Advances in Soft Computing. Springer, 2004, vol. 25, pp. 209–218. [Online]. Available: https://doi.org/10.1007/978-3-540-39985-8_22
- [5] —, “Compiling Learned Lexical Transformations: Semantic Canonicalization and Accepting-Leaf Generalization in Patch-Command Tries,” ResearchGate, 2026, Radixor Research Series, Paper 7. Author-released research manuscript. [Online]. Available: <https://doi.org/10.13140/RG.2.2.12232.69128>

- [6] R. Krovetz, “Viewing morphology as an inference process,” in *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993, pp. 191–202.
- [7] J. Xu and W. B. Croft, “Corpus-based stemming using cooccurrence of word variants,” *ACM Transactions on Information Systems*, vol. 16, no. 1, pp. 61–81, 1998.
- [8] Apache Lucene Project and Egothor Developers, “Egothor Diff.java: patch generation and execution, lucene 4.10.4,” 2015, pinned release source; accessed 20 September 2026. [Online]. Available: <https://github.com/apache/lucene-solr/blob/69175ad4fe8d224ad6a348d05f686105999a92e8/lucene/analysis/stempel/src/java/org/egothor/stemmer/Diff.java>
- [9] Apache Lucene Project, “StempelStemmer.java, lucene 4.10.4,” 2015, source documents Egothor provenance and language-independent core; accessed 20 September 2026. [Online]. Available: <https://github.com/apache/lucene-solr/blob/69175ad4fe8d224ad6a348d05f686105999a92e8/lucene/analysis/stempel/src/java/org/apache/lucene/analysis/stempel/StempelStemmer.java>
- [10] —, “StemmerOverrideFilter,” Lucene 3.6.2 API documentation; historical override mechanism, accessed 2026-09-20. [Online]. Available: https://lucene.apache.org/core/3_6_2/api/contrib-analyzers/org/apache/lucene/analysis/miscellaneous/StemmerOverrideFilter.html
- [11] T. Brychcín and M. Konopík, “HPS: High precision stemmer,” *Information Processing & Management*, vol. 51, no. 1, pp. 68–91, 2015.
- [12] B. Jongejan and H. Dalianis, “Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike,” in *Proceedings of ACL-IJCNLP*. Association for Computational Linguistics, 2009, pp. 145–153. [Online]. Available: <https://aclanthology.org/P09-1017/>
- [13] Centre for Language Technology, University of Copenhagen, “CSTlemma: captured-span rewrite executor, applyrules.cpp,” pinned source inspected 20 September 2026; corroborates current execution, not a historical version pin for the 2009 experiment. [Online]. Available: <https://github.com/kuhumcst/cstlemma/blob/f4f4865311fced50be868613b8c2e320333d43e8/src/applyrules.cpp>
- [14] K. Baker, “XSTEM: An exemplar-based stemming algorithm,” 2024, version 2, 2 June 2024; first submitted 2022. Preprint. [Online]. Available: <https://arxiv.org/abs/2205.04355v2>
- [15] G. Göksel, A. Arslan, and B. T. Dinçer, “A selective approach to stemming for minimizing the risk of failure in information retrieval systems,” *PeerJ Computer Science*, vol. 9, p. e1175, 2023.
- [16] Apache Lucene Project, “Lucene 4.10.4 release commit,” 2015, dated 3 March 2015; availability witness, not first Egothor release. [Online]. Available: <https://github.com/apache/lucene-solr/commit/69175ad4fe8d224ad6a348d05f686105999a92e8>
- [17] A. Bialecki and Apache Lucene Project, “Stempel: Algorithmic stemmer for polish language,” 2015, overview shipped with Lucene 4.10.4; describes the earlier Egothor design and cites Galambos’s publications. [Online]. Available: <https://github.com/apache/lucene-solr/blob/69175ad4fe8d224ad6a348d05f686105999a92e8/lucene/analysis/stempel/src/java/overview.html>
- [18] Centre for Language Technology, University of Copenhagen, “CSTlemma: trainer and runtime documentation,” pinned documentation, accessed 20 September 2026. [Online]. Available: <https://github.com/kuhumcst/cstlemma/blob/f4f4865311fced50be868613b8c2e320333d43e8/README.md>
- [19] X. Zhang, N. Thakur, O. Ogundepo, E. Kamaloo, D. Alfonso-Hermelo, X. Li, Q. Liu, M. Rezagholizadeh, and J. Lin, “Miracl: A multilingual retrieval dataset covering 18 diverse languages,” *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1114–1131, 2023.
- [20] G. C. Cawley and N. L. C. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010. [Online]. Available: <https://www.jmlr.org/papers/v11/cawley10a.html>