

Runtime Efficiency of Lexicon-Trained Stemming: Compiled Structure and a Speed–Reference–Agreement Envelope in Radixor

Leo Galambos

Abstract—Lexicon-trained stemming replaces handwritten linguistic rules with a learned transformation model, but this need not imply dictionary-like online cost. We characterize Radixor, a compiled descendant of the Egothor patch-command stemmer, using a frozen 2026-08-25 Java benchmark snapshot covering 20 default language resources. Timing is measured on lower-cased, known dictionary-field occurrences whose model-declared canonical target differs from the surface form; the workload is therefore reference-changing and is not a natural-corpus traffic estimate. Canonical Radixor point estimates range from 42.8 to 150.6 amortized ns/occurrence, with an unweighted resource median of 67.8. Across 18 same-language direct Snowball comparisons, a prespecified closest-available same-harness denominator gives a median Snowball/Radixor ratio of 2.33, while the canonical-denominator sensitivity gives 2.57. A formally defined cross-policy invariant identifies 17 stable Radixor advantages and one Finnish parity case; JMH score-error ranges are used only as within-run measurement-separation diagnostics, not significance tests. A separate resource-wide pairwise audit places Radixor at higher conformity to its declared overlapping morphological reference relation in all 18 matched comparisons, but this audit uses a different input scope and reference data related to Radixor’s training resources. Across the fixed 20-resource census, internal-node count has Spearman $\rho = 0.931$ with runtime versus $\rho = 0.723$ for training-row count; their association gap is 0.208 and remains positive under every one-resource deletion. Descriptive partial-rank sensitivities remain high when controlling separately for training-row count ($\rho = 0.890$) and timing-corpus size ($\rho = 0.875$). A separate English deterministic-prefix experiment shows a non-monotone speed–reference–agreement envelope. The study does not establish natural-traffic latency, causal contraction effects, population-level language inference, or downstream retrieval effectiveness.

Index Terms—stemming, information retrieval, lexicon-trained models, performance, tries, finite-state morphology, Java Microbenchmark Harness, Snowball, Radixor

I. INTRODUCTION

STEMMING is often treated as a small preprocessing step: receive a token, remove or rewrite some morphology, and return a normalized indexing form. That view makes the runtime story of classical rule-based stemmers intuitive. A handwritten suffix algorithm such as Lovins or Porter executes a bounded sequence of string tests and transformations [1], [2]; Snowball makes such rule systems executable from a compact language definition [3]. A lexicon-trained stemmer appears, at first sight, to have a less favorable computational profile. It learns from many word-to-target examples, may encode

thousands of distinct transformations, and must generalize those transformations beyond exact lexical entries.

The Egothor/Radixor lineage was designed around a different separation of concerns. Training converts lexical mappings into reusable edit commands and organizes those commands in a trie; runtime traverses the compiled structure and applies the selected patch command [4], [5]. The lexical resource can therefore be large while the online operation remains a compact decision process. The current Radixor implementation strengthens that separation by compiling immutable patch commands and contracting uniform preferred-command subtrees into accepting leaves [6]. The systems question is whether the *compiled representation* of learned lexical evidence can provide a competitive execution point without requiring online consultation of the training lexicon.

Throughout this paper, a *canonical target* means the model-declared first field of one Radixor lexical row. Historical benchmark artifacts use field names such as `root_correct` and `root_preserved`; those names are retained in source files for provenance, but the scientific text does not interpret the target as a linguistic root morpheme or a unique context-sensitive lemma. A *stem* is the output produced by the runtime system.

A performance comparison cannot be reduced to one timing ratio. Stemmer implementations expose different transformation objectives, APIs, normalization requirements, and output multiplicities. A light or minimal suffix filter can have very low cost because its declared operation is deliberately narrow; a dictionary analyzer can return multiple lemmas; a `TokenFilter` path includes framework overhead that a direct generated stemmer does not. Java timing itself is sensitive to warm-up, JIT compilation, garbage collection, and machine state [7], [8]. A scientifically useful performance study must therefore separate: (i) the exact timed workload, (ii) the linguistic/reference relation used for intrinsic conformity, (iii) the compiled structural properties that vary across models, and (iv) the uncertainty that belongs to the timing campaign.

This study performs that separation on one frozen Radixor benchmark state. It combines four already measured evidence layers:

- 1) a 20-language Java runtime snapshot over normalized, known reference-changing dictionary-field occurrences;
- 2) deterministic exact-target counters and a separately generated resource-wide pairwise conformity audit;
- 3) structural counters for the corresponding full-knowledge compiled models, archived by the edit-cost campaign;

and

- 4) a 10-level English deterministic-prefix experiment that relates lexical-row coverage to exact-target reference agreement and timing within its own benchmark suite.

No new stemmer timings are generated for this article. The contribution is the cross-layer analysis and conservative interpretation of the frozen measurements.

The research questions are:

- RQ1: What amortized runtime range does the compiled mechanism occupy on known, lower-cased, reference-changing dictionary-field occurrences across the 20 default resources, and how does it compare with same-language direct Snowball implementations on those same timing sequences?
- RQ2: Which measured properties of the compiled trie are most strongly associated with runtime across the fixed resource census, and is the structural association robust to obvious scale and benchmark-organization sensitivities?
- RQ3: In a separate resource-wide audit, how does deterministic primary-output stemming conform to the model-declared overlapping morphological reference relation? Does the observed Radixor runtime advantage coincide with lower conformity to that declared relation?
- RQ4: What speed-reference-agreement envelope is observed along the English deterministic nested-prefix construction when the evaluation universe is held fixed?

The principal findings are deliberately narrower than a universal “fastest stemmer” claim. Radixor’s 20 canonical resource-specific point estimates span 42.8–150.6 amortized ns/occurrence. In the 18 languages with a direct Snowball comparator, a prespecified closest-available same-harness denominator gives an unweighted median ratio of 2.33, while the canonical-denominator sensitivity gives 2.57. Defining a stable advantage as lower Radixor point estimate and separated JMH score-error ranges under *both* denominator policies yields 17 stable advantages and Finnish parity. The structural comparison is stronger than a pointwise correlation contrast: internal nodes exceed training rows by 0.208 in Spearman association with runtime, and the paired leave-one-resource-out gap remains positive from 0.105 to 0.235. One-control partial-rank sensitivities remain 0.890 after controlling training rows and 0.875 after controlling timing-corpus size. The English experiment shows a separate within-language pattern: exact-target agreement changes systematically along the prefix path, whereas runtime point estimates are not monotone.

The contributions are:

- 1) an occurrence-normalized 20-resource runtime characterization of a compiled lexicon-trained stemmer with explicit JMH uncertainty and source-state provenance;
- 2) a direct-Snowball analysis with two prespecified Radixor denominator policies and a machine-verifiable cross-policy stability invariant;
- 3) a structural association analysis with paired LOO contrast and partial-rank scale sensitivities over a fixed resource

census; and

- 4) a fixed-language deterministic-prefix English speed-reference-agreement envelope, kept explicitly separate from the multilingual comparator and pairwise conformity analyses.

II. BACKGROUND AND NOVELTY BOUNDARY

A. Rule-Based, Lexicon-Trained, and Finite-State Morphology

Classical stemming algorithms encode their conflation behavior directly in procedural rules or suffix tables [1], [2]. Snowball preserves this model while providing a language for defining and generating implementations of stemming algorithms [3]. CISTEM is a more recent rule-based German system designed after a comparative study of public German stemmers [9]. These systems are attractive performance baselines because their online representations are relatively compact, but their objectives and transformation scopes are not identical.

Lexicon-trained stemming moves some of the linguistic specification from source code into data. Egothor represents a word-to-target mapping as an edit or patch command and generalizes commands through a trie [4]. Stempel in Apache Lucene retains the same broad algorithmic lineage for Polish [10]. The important architectural property is that the training lexicon is compiled: the runtime does not need to scan training examples or run dynamic-programming edit distance for every token.

This idea is adjacent to, but distinct from, finite-state morphology. Finite-state methods have long shown that large lexical/morphological descriptions can be compiled into efficient automata [11]. Compact digital search structures similarly exploit shared prefixes or transitions; double-array tries and minimal acyclic finite-state automata are established examples of reducing lexical lookup cost through representation design [12], [13]. Radixor is not claimed here to introduce trie lookup or automaton minimization. Its relevant distinction is the object stored in the search structure: learned executable patch commands, with current compilation additionally able to terminate early in uniform preferred-command regions.

B. Why Performance Must Be Coupled to Linguistic Scope

A stemmer benchmark can be misleading when one system performs a narrow normalization and another attempts a broader learned mapping. Intrinsic stemming evaluation has therefore long distinguished under- and over-stemming behavior from retrieval-level effects [14], [15]; multilingual evidence also shows that the most intrinsically accurate stemmer need not produce the largest information-retrieval gain [16]. The frozen Radixor documentation accordingly reports runtime, exact-target agreement, and pairwise conflation conformity as separate dimensions [6]. Exact-target agreement asks whether an output equals the model-declared canonical target stored in the lexical resource. The pairwise audit asks whether distinct forms that share at least one annotated morphological group are conflated while reference-negative pairs remain separated. Neither is a downstream retrieval measure.

This distinction is particularly important for direct Snowball comparison. Snowball algorithms are hand-specified suffix

stemmers, not predictors of the particular canonical targets used to train Radixor. The separately generated pairwise overlay below is therefore *not* an independent gold standard or a fairness correction: Radixor was trained from the same lexical-resource family that defines the reference relation, whereas Snowball was not. Its limited purpose is to test whether lower measured runtime coincides with lower conformity to the particular resource relation Radixor was compiled to implement. The analysis consequently keeps execution time and resource-relative conformity as two separate evidence layers rather than a joint same-sample score.

C. Java Microbenchmarking

Managed-runtime performance is difficult to measure because JIT compilation, allocation, GC, scheduling, and VM state can produce substantial variation [7]. JMH exists specifically to structure JVM microbenchmarks around warm-up, forks, state isolation, and result aggregation [8]. The present study does not claim that JMH removes all environmental variance. Instead, it retains the frozen project’s three-fork protocol and reports the published 99.9% JMH score error for each method. Cross-language associations use only deterministic model descriptors plus those point estimates; they are not reinterpreted as a random sample of languages.

D. Novelty Boundary

The novelty is not a new stemmer, trie, automaton, or benchmarking framework. Nor is it a new proof that compiled lexical systems can be efficient. The contribution is the empirical connection between a modern lexicon-trained stemming architecture and its measured runtime state across 20 separately compiled language resources: the paper relates actual JVM timing to actual compiled-trie topology, then uses same-language direct baselines and a fixed-universe coverage experiment to distinguish structural scale from raw lexical scale and to test whether runtime advantages coincide with lower conformity to Radixor’s declared resource relation.

III. RADIXOR RUNTIME ARCHITECTURE

A. From Lexical Mapping to Executable Patch

A Radixor model begins with lexical rows whose first field is a canonical target and whose remaining fields are surface forms. During model construction, each surface-to-target mapping is converted into a patch command. Compound commands encode a short sequence of edits; common special cases have dedicated compiled command classes. The learned commands are associated with paths in a backward trie so that suffix evidence is encountered early for natural-language morphology [4], [6].

The online path is shown in Fig. 1. The benchmark corpus is normalized before the timed method. A token is traversed through the immutable compiled trie, the preferred command is resolved, and that command is applied to construct the output. Dictionary loading, model construction, corpus creation, and benchmark setup do not occur inside the measured operation.

A useful conceptual decomposition for a token w is

$$T(w) = T_{\text{lookup}}(w; M) + T_{\text{patch}}(c(w)), \quad (1)$$

where M is the compiled model and $c(w)$ is the selected command. Equation (1) is a decomposition of the implementation path, not a fitted timing model. The published benchmark does not instrument the two terms separately. Instead, the structural analysis asks whether aggregate runtime across models tracks observable proxies for T_{lookup} : nodes, edges, path length, dense lookup state, and command diversity.

B. Uniform-Subtree Contraction

The current compiled baseline can stop traversal before the lexical key is exhausted. If all reachable entries in a subtree resolve to the same preferred command, the subtree can be represented by an accepting leaf. This is a semantics-preserving transformation for the preferred result used by the benchmark path: subsequent characters cannot change the selected command. Contraction therefore makes model richness and hot lookup depth different quantities.

This property motivates, but does not prove, the central structural hypothesis. A larger training resource may create more lexical distinctions, yet repeated distinctions can still share paths or collapse into uniform decision regions. The correct empirical question is consequently not “does dictionary size predict speed?” but “which properties of the *compiled decision graph* covary with speed?”

IV. EXPERIMENTAL METHODOLOGY

A. Frozen Evidence and Source Identity

The article-facing data are preserved in Radixor repository snapshot 0c3b13f [6]. The physical 2026-08-25 performance campaign records a more specific identity: Radixor/Java 4.2.0-6-g84e57fb, core source revision 84e57fb27ae40913569e826858a8ebb07cf2ea01, and checksums for the measured tracked/untracked source state. Documentation integration occurred after physical measurement, so the later article-facing snapshot is not substituted for the dirty measured state.

Structural descriptors come from the full-knowledge D1I1R1M0 baseline in the archived edit-cost matrix. Those rows carry the same measured Radixor identity and core revision. The edit-cost builder uses the production reduction mode and default dominant-winner settings and explicitly compares the full-knowledge experimental compiled trie with the production loadCompiled model over every dictionary form before continuing [6]. This establishes preferred-output equivalence under the relevant production compilation configuration, not byte-for-byte object identity. The full-knowledge structural counters are also invariant across the five frozen split seeds. Pairwise conformity counts are exact TP/FP/FN/TN projections from the clean public quality snapshot, generated separately from timing but from the same lexical-resource family. The article package records upstream checksums for every source projection.

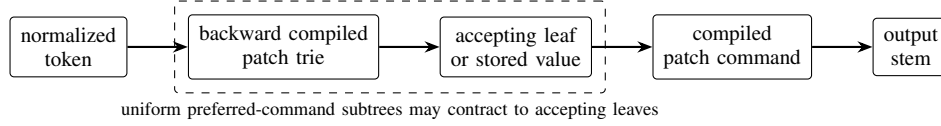


Fig. 1. Measured Radixor runtime path. Training compiles lexical mappings into a backward patch-command trie. The 2026-08-25 baseline additionally contracts a subtree to an accepting leaf when all reachable entries resolve to the same preferred command. The present study measures the resulting architecture; it does not provide an uncontracted same-run causal ablation.

B. Timing Workload

For each language, benchmark setup parses every single-token field occurrence in the bundled Radixor lexical rows, applies `trim().toLowerCase(Locale.ROOT)`, and retains occurrence multiplicity. The timed sequence is therefore *not* a deduplicated vocabulary: if the same normalized surface string occurs in several lexical rows, those occurrences remain separate. It is also not frequency-weighted by natural text. The changed timing corpus then retains only occurrences for which the normalized surface form differs from the normalized canonical target. If fewer than 5,000 such occurrences exist, the complete sequence is repeated in stable order until the timing corpus reaches at least 5,000 entries; larger sequences are neither sampled nor truncated [6]. One JMH operation processes the complete language-specific in-memory occurrence sequence.

Let S_ℓ be the JMH average-time score in ns/op for resource ℓ and N_ℓ its number of timed dictionary-field occurrences. The normalized statistic is the *amortized batch cost*

$$t_\ell = \frac{S_\ell}{N_\ell}. \quad (2)$$

It is neither natural-traffic throughput nor single-request latency. The published score error E_ℓ is divided by the same denominator, producing $[(S_\ell - E_\ell)/N_\ell, (S_\ell + E_\ell)/N_\ell]$. Non-overlap of two displayed ranges is used only as a conservative within-campaign *measurement-separation diagnostic*. It is not a paired hypothesis test, supplies no cross-method p -value, and is not adjusted for multiple comparisons.

All published speed methods use JMH 1.37, three separate JVM forks, five one-second warm-up iterations and seven one-second measurement iterations per fork, one benchmark thread, and a fixed 6 GiB heap. The workstation runs OpenJDK 25.0.4 on Fedora Linux 44 and an AMD Ryzen 5 8600G. The CPU governor was set to performance; no explicit CPU affinity was imposed. Forks improve within-method measurement characterization but are not independent replications of machine state, and the physical campaign was not a randomized/interleaved pair experiment.

Because corpus normalization is setup work, Radixor uses `getNormalizedString` inside the timed method and does not repeatedly lowercase the already normalized input. Direct Snowball methods receive the same normalized strings. Lucene TokenFilter comparators include normalization required by their measured public pipelines. We therefore do not pool direct and TokenFilter paths as implementation-identical observations.

Cross-resource medians, ratio medians, and rank coefficients are *unweighted across resource identities*: each lan-

TABLE I
FROZEN JAVA PERFORMANCE PROTOCOL.

Item	Value
Date	2026-08-25
JMH	1.37, average time
Forks	3 JVM processes
Warm-up	5×1 s per fork
Measurement	7×1 s per fork
Threads	1
Heap	<code>-Xms6g -Xmx6g</code>
JVM	OpenJDK 25.0.4
CPU	AMD Ryzen 5 8600G
OS	Fedora Linux 44
Timed data	normalized reference-changing occurrences

guage/resource contributes one value. They do not estimate the runtime distribution of multilingual production traffic.

C. Direct Snowball Comparison and Pairwise Reference Relation

The cleanest external algorithmic comparison uses official generated Snowball Java implementations because they expose direct, deterministic, single-output APIs. The frozen run contains 18 shared language workloads: 17 non-English Snowball 3.1.0 algorithms plus English Porter2. The same campaign contains multiple Radixor JMH methods for several languages. The *primary denominator policy* therefore uses the closest available same-harness path: for 14 legacy Snowball-language cases it uses the Radixor method from the same parameterized Snowball benchmark class; English uses the two methods in its dedicated English class. Czech, Persian, and Polish are direct-only Snowball 3.1.0 additions whose direct state has no Radixor method, so their primary denominator is the canonical same-run MultiLanguage Radixor row. A second, prespecified *canonical policy* uses the language-page Radixor row for all 18 resources.

For each resource and denominator policy d we compute

$$R_{\ell,d} = \frac{t_{\ell,\text{Snowball}}}{t_{\ell,\text{Radixor},d}}. \quad (3)$$

$R_{\ell,d} > 1$ means Snowball has the larger point-estimate amortized cost on that known reference-changing occurrence workload. The two multilingual medians are reported together as unweighted descriptive summaries under the two denominator policies; neither is treated as a unique population estimand.

Let $L_{\ell,d} = 1$ when the Radixor point estimate is lower than Snowball's under policy d , and $D_{\ell,d} = 1$ when their

reported JMH score-error ranges do not overlap. A *stable Radixor advantage* is the deterministic invariant

$$A_\ell = \prod_{d \in \{\text{primary}, \text{canonical}\}} L_{\ell,d} D_{\ell,d}. \quad (4)$$

This summarizes cross-policy robustness in the frozen campaign; it is not a statistical significance indicator.

The pairwise conformity layer uses a different input scope and is defined separately. Let $\mathcal{U}_\ell$ be the distinct original surface forms retained by the ALL_WORDS evaluator, and let $G(u)$ be the set of lexical/morphological groups containing form u . The overlapping reference relation is

$$g(u, v) = \mathbf{1}[G(u) \cap G(v) \neq \emptyset], \quad u \neq v, \quad (5)$$

while deterministic stemmer h induces

$$p_h(u, v) = \mathbf{1}[h(u) = h(v)]. \quad (6)$$

TP, FP, FN, and TN are counts over distinct unordered form pairs according to (g, p_h) . Because forms may have multiple memberships, g is an overlapping resource relation and need not be transitive; it is not synonymy or semantic equivalence. PRIMARY_OUTPUT equality supplies the deterministic predicted relation. Balanced accuracy is

$$\text{BA} = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right). \quad (7)$$

The audit retains original forms and distinct surface identity for pair counting, whereas timing lowercases dictionary-field occurrences, preserves occurrence multiplicity, and keeps only reference-changing entries. Consequently BA is a separate resource-wide conformity audit, not a fairness-adjusted “quality-at-speed” coordinate on the timed sample. Because the same resource family supplies Radixor’s training evidence and the reference groups, BA is not independent linguistic validation against Snowball.

D. Structural Association Analysis

The full-knowledge structural projection provides dictionary rows; distinct compiled patch commands; internal nodes; leaves; edges; accepting leaves; value references; logical leaf paths; longest path; average path length; dense lookup nodes; and dense-table slots. We join these deterministic descriptors to the canonical published Radixor runtime row for each resource.

Because the 20 resources are a fixed project census rather than an i.i.d. sample from a population of languages, conventional population-significance tests would create a misleading inferential frame. We report Spearman rank correlation $\rho_s(X, t)$ as a descriptive monotone-association statistic. The primary contrast compares internal-node and training-row associations directly:

$$\Delta_\rho = \rho_s(N_{\text{internal}}, t) - \rho_s(N_{\text{rows}}, t). \quad (8)$$

We recompute Δ_ρ after deleting each resource in turn; the paired leave-one-resource-out range tests whether the ordering of these two association strengths is concentrated in one language.

As an additional scale sensitivity, we report one-control partial correlations of ranked variables. For ranked variables X, Y, Z ,

$$\rho_{XY \cdot Z} = \frac{\rho_{XY} - \rho_{XZ}\rho_{YZ}}{\sqrt{(1 - \rho_{XZ}^2)(1 - \rho_{YZ}^2)}}. \quad (9)$$

We apply Eq. (9) to internal nodes versus runtime, controlling separately for training-row count and timing-occurrence count. These are descriptive single-control robustness diagnostics over the fixed census, not causal coefficients or population estimates.

Three further restrictions test measurement organization: (i) a 15-resource subset in which every canonical runtime row comes from the same MultiLanguage benchmark class; (ii) an alternate-measurement analysis that substitutes the same-run Snowball-harness Radixor row wherever one exists; and (iii) an 18-resource subset excluding Persian and Yiddish, whose sequences are repeated to reach 5,000 timing occurrences. Log-log Pearson correlations remain machine-readable but are not used for the central claim.

The predictors are strongly collinear because nodes, edges, value references, command diversity, and path statistics arise from the same compiled object. The analysis is observational: token length, morphology, command distributions, workload scale, and cache residency can co-vary with structure. None of the reported coefficients is an independently identified marginal mechanism or a per-node causal cost.

E. English Deterministic-Prefix Operating Curve

The English coverage benchmark builds Radixor from deterministic nested row prefixes at 10% increments from 10% to 100%, evaluates exact-target agreement on the complete 1,002,414 dictionary-field occurrences, and times the same 208,540 normalized reference-changing occurrence sequence. The evaluation universe includes both selected and unselected rows. The resulting series is a *fixed-language deterministic-prefix reference-agreement operating curve*, not a randomized coverage experiment or held-out causal learning curve.

Coverage-specific models can differ in compiled structure, and several runtime points have substantial JMH uncertainty. The curve is used only to characterize the observed speed-reference-agreement envelope along this exact prefix construction. Its 100% timing point belongs to a dedicated JMH suite and must not be substituted for the canonical English comparison row from another method.

V. RESULTS

A. Multilingual Radixor Runtime

Across the 20 default language resources, canonical Radixor timing ranges from 42.8 ns/occurrence (Yiddish) to 150.6 ns/occurrence (German), with an unweighted resource median of 67.8 ns/occurrence. The range should not be interpreted as a ranking of linguistic difficulty or expected multilingual traffic: each row corresponds to a different model and reference-changing occurrence sequence.

Table II shows representative points. Raw training size is visibly not a sufficient runtime ordering: English has the largest row count in the set but a mid-range canonical runtime, whereas German and Finnish are slower despite much smaller training resources; their compiled tries are among the largest.

TABLE II
SELECTED FULL-KNOWLEDGE CANONICAL RADIXOR RUNTIME AND STRUCTURE. ERROR IS THE PUBLISHED JMH SCORE ERROR AFTER CONVERSION TO NS/OCCURRENCE.

Lang.	ns/occ.	Error	Internal nodes	Commands
YI	42.8	0.3	1,055	202
NN	44.0	6.1	3,247	299
DA	48.0	7.2	3,484	138
PL	75.4	1.6	13,398	846
EN	80.8	8.3	30,868	337
RU	106.9	3.1	46,437	1,840
FR	116.4	2.8	47,066	2,730
FI	148.8	2.7	80,790	2,683
DE	150.6	4.8	81,363	6,986

TABLE III
SELECTED DIRECT SNOWBALL COMPARISONS. R IS THE PRIMARY SNOWBALL/RADIXOR AMORTIZED-TIME RATIO. Δ BA COMES FROM THE SEPARATE RESOURCE-WIDE ALL_WORDS CONFORMITY AUDIT, NOT THE TIMED OCCURRENCE SAMPLE.

Lang.	Radixor ns/occ.	R	Δ BA
FI	151.3	0.99	0.2450
PL	75.4	1.20	0.1675
CS	70.7	1.23	0.2103
DE	181.2	2.16	0.1344
EN	80.8	2.78	0.0111
IT	71.8	4.84	0.1304
FA	53.0	5.78	0.4405
YI	42.8	7.04	0.0980

B. Direct Snowball Comparison

Figure 2 reports the primary direct-Snowball ratios. Under the closest-available same-harness policy, Radixor has the lower point estimate in 17 of 18 comparisons. The unweighted primary median is 2.33, compared with 2.57 under the canonical-denominator policy. Finnish is the only point-ordering disagreement: the primary Radixor point estimate is 151.3 ns/occurrence versus 149.9 for Snowball and their score-error ranges overlap; the canonical Radixor value is 148.8 ns/occurrence and still overlaps.

Applying Eq. (4), 17 of 18 resources satisfy the stable-advantage invariant. Finnish alone does not. Thus the cross-policy conclusion is *17 stable advantages plus one parity case*; the invariant remains a deterministic robustness summary of one frozen campaign, not a significance test.

The separate ALL_WORDS pairwise audit gives higher Radixor balanced accuracy in all 18 matched resource-defined comparisons, with median difference 0.1450. This establishes only that the stable runtime advantages do not coincide with lower conformity to the particular overlapping reference relation Radixor was compiled from. It is not independent adjudication between linguistic objectives and does not use the same normalized occurrence sample as timing [14]–[16].

The frozen matrix also contains other implementation paths. Several deliberately narrow light/minimal filters have lower point estimates than Radixor—for example, German Minimal is 93.6 ns/occurrence versus canonical Radixor’s 150.6—so the following dictionary/table cases are contextual examples, not a complete speed ranking. German CISTEM takes 1,193.1 ns/occurrence versus Radixor’s 150.6; Polish Stempel direct 295.6 versus 75.4; Lucene Morfologik on Polish 1,247.0 versus

TABLE IV
SPEARMAN ASSOCIATION WITH CANONICAL RADIXOR AMORTIZED NS/OCCURRENCE. “COMMON” RESTRICTS TO 15 MULTILANGUAGE-HARNESS RESOURCES; “NO REPEAT” EXCLUDES THE TWO SEQUENCES REPEATED TO 5,000 OCCURRENCES.

Predictor	All 20	Common	No repeat
Internal nodes	0.931	0.918	0.917
Distinct commands	0.896	0.911	0.924
Average path length	0.811	0.836	0.783
Training rows	0.723	0.700	0.649
Timing occurrences	0.745	0.614	0.668

75.4; and Ukrainian Morfologik direct 650.8 versus 49.0. Their objectives, API paths, and output semantics differ, so they are not pooled into a universal factor [10], [17].

C. Runtime Covaries with Compiled Structure

Table IV reports the full fixed-census association and measurement-organization restrictions. Internal-node count has $\rho = 0.931$ over all 20 resources and remains $\rho = 0.918$ in the 15-resource common-harness subset, $\rho = 0.919$ after alternate same-run Radixor substitution, and $\rho = 0.917$ after excluding the two repeated-to-5,000 resources. Distinct patch-command count is similarly stable.

The direct contrast with training-row scale is stronger than separate coefficients alone. The full association gap in Eq. (8) is $\Delta_\rho = 0.208$. Under every one-resource deletion the paired gap remains positive, from 0.105 to 0.235. Thus no single resource is required for internal-node rank association to exceed training-row rank association in this census.

One-control partial-rank diagnostics are also high: internal nodes versus runtime remain $\rho = 0.890$ after controlling training-row rank and $\rho = 0.875$ after controlling timing-occurrence-count rank. These diagnostics do not identify a causal node effect; they show that the monotone relationship is not removed by either obvious scale variable considered separately.

Figure 3 shows the internal-node relationship on log scales. English is a useful deviation from raw lexical scale: 396,939 training rows compile to 30,868 internal nodes and 80.8 ns/occurrence, while German’s 54,092 rows compile to 81,363 internal nodes and 150.6 ns/occurrence. Compilation substantially changes the row-count ordering before runtime.

D. Runtime Is Not Explained by a Simple Quality Ranking

In the separate ALL_WORDS resource-wide audit, Spearman association between Radixor balanced accuracy and canonical Radixor runtime is $\rho = -0.220$; its leave-one-resource-out values range approximately from -0.374 to -0.089 . This fixed snapshot therefore does *not support a simple monotone explanation* in which higher pairwise reference conformity systematically requires slower execution.

That statement is deliberately limited. Timing and pairwise conformity use different input normalization and weighting, and a weak rank coefficient cannot exclude non-monotone, subgroup-specific, or confounded relationships. Morphology, ambiguity, token length, source conventions, and compiled

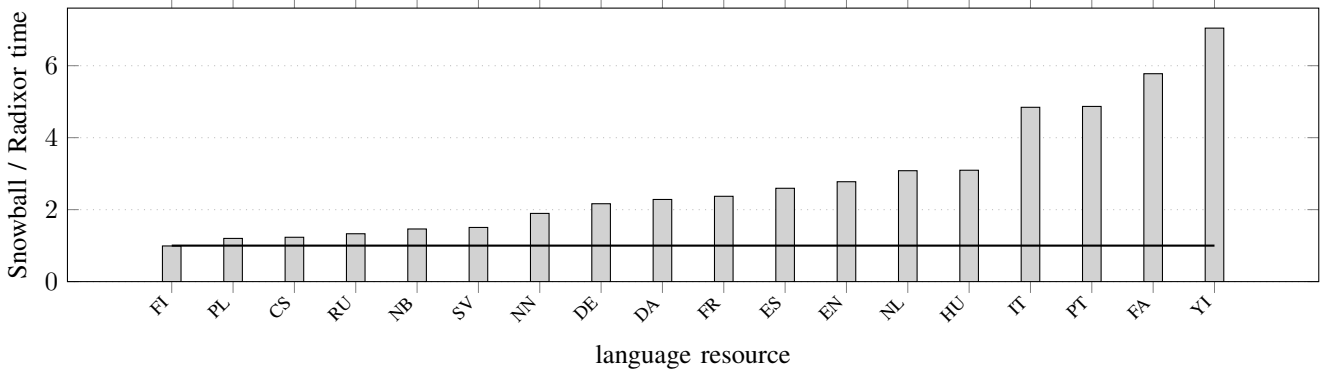


Fig. 2. Primary point-estimate runtime ratios for 18 direct Snowball comparisons on the same normalized reference-changing occurrence sequences. Values above 1 mean larger Snowball amortized time. Finnish is unresolved parity; the canonical Radixor denominator is analyzed separately as a sensitivity policy.

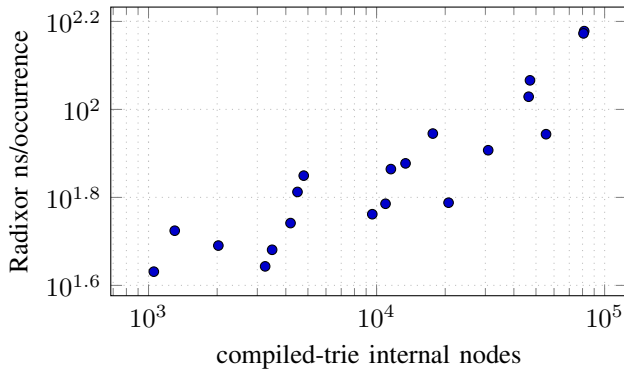


Fig. 3. Full-knowledge compiled-trie internal nodes versus canonical Radixor amortized runtime for the 20 fixed resources. Both axes are logarithmic. $\rho = 0.931$ is descriptive, not a per-node causal model.

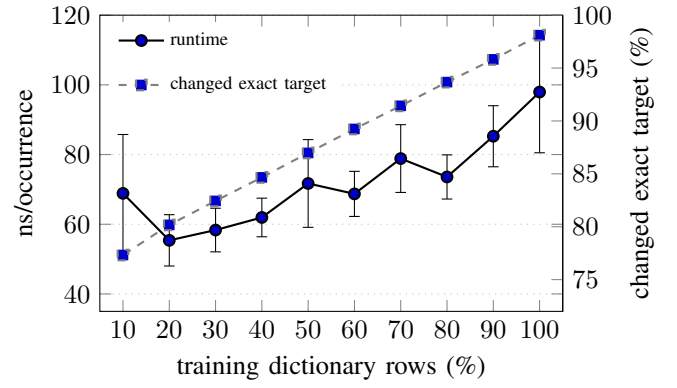


Fig. 4. English deterministic-prefix training-row coverage versus changed-occurrence exact-target agreement and JMH runtime. Timing error bars are published JMH score errors; the exact-target counters are deterministic for the fixed resource.

structure all vary across resources. The result is contextual evidence against one simple cross-resource ordering, not proof that runtime and linguistic difficulty are independent.

E. English Coverage: A Speed-Reference-Agreement Envelope

Table V and Fig. 4 show the deterministic-prefix English experiment. These points belong to the dedicated coverage suite; its 100% runtime (97.9 ns/occurrence) is not the canonical English comparison value (80.8 ns/occurrence) and is never substituted for it. Changed-occurrence exact-target agreement moves from 98.110% at the full prefix to 77.327% at 10%. All-occurrence exact-target agreement moves from 97.668% to 93.057%. Canonical-target preservation over occurrences that are themselves targets remains comparatively stable, from 97.552% to 97.190%.

Runtime is not monotone. The full model measures 97.9 ns/occurrence. The lowest point estimate, 55.4 ns/occurrence, occurs at the 20% prefix, while 10% rebounds to 68.9 ns/occurrence. Several adjacent score-error ranges overlap and the 10% and 100% points have relatively wide uncertainty. The deterministic exact-agreement counters themselves have no sampling uncertainty for this fixed dictionary; uncertainty bars apply only to JMH timing.

The defensible result is an observed envelope along one nested-prefix path. Reducing the selected prefix moves both

compiled representation and reference agreement, but the timing direction and magnitude are not a universal function of coverage. The data do not identify 20% as an optimum and do not estimate a randomized coverage effect.

VI. DISCUSSION

A. Lexical Training Size and Online Work Are Different Objects

The central systems result is a separation between training evidence and online executable state. Radixor uses extensive lexical data during model construction, yet the timed operation traverses a compiled decision structure and executes a patch command. On the measured workstation, canonical values for the known reference-changing occurrence workload remain in the tens to low hundreds of ns/occurrence. Across direct Snowball comparisons, 17 resources satisfy the cross-policy stable-advantage invariant and Finnish remains parity. The primary and canonical unweighted median ratios are 2.33 and 2.57, respectively.

This is not a theorem that learned models are faster than rule systems. It is an implementation-specific result over a reference-defined workload. The architectural lesson is narrower: the number of lexical examples used to construct a model is not the number of examples that must be consulted online.

TABLE V
ENGLISH DETERMINISTIC-PREFIX OPERATING CURVE ON THE FIXED COMPLETE DICTIONARY. TIMING UNCERTAINTY IS THE PUBLISHED JMH SCORE ERROR AFTER DIVISION BY 208,540 REFERENCE-CHANGING OCCURRENCES.

Training prefix	ns/occ.	Error	All exact target	Changed exact target	Target preserved
100%	97.9	17.4	97.668%	98.110%	97.552%
90%	85.3	8.8	97.239%	95.821%	97.612%
80%	73.6	6.3	96.827%	93.673%	97.656%
70%	78.9	9.7	96.392%	91.430%	97.695%
60%	68.7	6.5	95.935%	89.244%	97.693%
50%	71.7	12.6	95.453%	86.979%	97.678%
40%	62.0	5.5	94.939%	84.667%	97.638%
30%	58.3	6.3	94.398%	82.443%	97.538%
20%	55.4	7.4	93.821%	80.174%	97.406%
10%	68.9	16.9	93.057%	77.327%	97.190%

B. Compiled Structure Is the Stronger Monotone Associate

The structural result can be stated without relying on two point coefficients alone. Internal-node association exceeds training-row association by $\Delta_\rho = 0.208$ in the full census and by at least 0.105 under every one-resource deletion. The association remains strong in the common-harness and non-repeated restrictions, and its one-control partial-rank values remain 0.890 given training-row rank and 0.875 given timing-occurrence rank.

These diagnostics do not identify independent causal mechanisms. Internal nodes, edges, commands, path depth, token length, and cache behavior are interdependent. The result supports a robust ordering statement—compiled structural scale covaries more closely with runtime than raw row count under the tested sensitivities—not a quantitative cost model for nodes or commands.

C. Runtime and Resource Conformity Are Separate Axes

The matched pairwise audit is favorable to Radixor but intentionally asymmetric. All 18 direct-Snowball matches have higher Radixor balanced accuracy against Radixor’s declared resource relation, yet Radixor was trained from the same resource family. In addition, timing is lower-cased, occurrence-weighted, and restricted to known reference-changing forms, whereas ALL_WORDS pairwise evaluation uses original distinct forms over the complete resource relation. The audit therefore cannot be used as a fairness correction or a same-sample quality-at-speed coordinate.

Its useful role is narrower: the 17 stable runtime advantages do not coincide with lower conformity to the relation Radixor was compiled to implement. Consistent with prior stemming evaluation literature, this intrinsic statement does not imply downstream retrieval effectiveness [14]–[16].

D. Implications for Search and Text-Normalization Systems

Stemming can appear at indexing and query time, but amortized full-sequence JMH cost does not directly translate to end-to-end request latency. Tokenization, normalization chains, memory allocation, concurrency, cache state, and I/O may dominate. The present results only remove one architectural assumption: a large lexical training source need not force dictionary-scale online lookup work in a compiled stemmer.

The English prefix experiment similarly supports model-management experimentation rather than a universal tuning

law. Moving along the deterministic prefix changes the observed reference-agreement/runtime point, but the timing response is non-monotone and path-specific. Application-specific model selection therefore requires a workload and linguistic objective defined for that deployment.

VII. THREATS TO VALIDITY

A. One Physical Performance Campaign

All timing values come from one workstation/JDK campaign. Three JMH forks are separate JVM processes, not independent replications over machines or days. Methods were not pair-randomized or interleaved, CPU affinity was not fixed, and temporal machine-state effects can remain. Published JMH score errors characterize within-method measurement in that campaign; they are not repeated-campaign confidence intervals. Replication on additional x86/ARM systems, JDKs, days, and randomized benchmark order would test portability.

B. Reference-Defined, Occurrence-Weighted Timing Workload

Timing input is derived from Radixor lexical resources, lowercased in setup, restricted to known occurrences whose canonical target differs from the surface form, and not deduplicated across rows. Occurrence multiplicity therefore reflects dictionary representation, not natural language frequency. Inclusion is asymmetric because Radixor’s reference mapping decides which occurrences enter the speed workload. Unknown words, arbitrary compounds, spelling variants, preserve-heavy traffic, and natural query/index distributions may exercise different failure depths, command paths, and caches. Reported ns/occurrence is amortized full-sequence batch cost, not natural-traffic latency.

C. Timing and Pairwise Audit Use Different Input Scopes

The ALL_WORDS pairwise audit is not evaluated on the timing sample. It retains original included surface forms, represents each distinct surface string once for pairwise identity, and allows multiple group memberships. Timing instead lowercases dictionary-field occurrences, preserves their multiplicity, and removes target-equal occurrences. Therefore no inference in this paper depends on treating the two layers as identical samples.

D. Comparator Scope and API Path

Direct Snowball is the principal direct algorithmic comparator. Other rows include Lucene TokenFilters and dictionary/table systems with different normalization, lifecycle, and output semantics; several narrow light/minimal filters are faster than Radixor in the frozen matrix. Contextual examples are not a complete speed ranking and are not pooled into one factor.

E. Deterministic Repetition, Cache Residency, and Token Length

Persian and Yiddish contain fewer than 5,000 reference-changing occurrences, so their complete sequences repeat in stable order. Repetition can increase branch/cache predictability. Excluding both leaves internal-node association at $\rho = 0.917$ and command-count association at $\rho = 0.924$. The benchmark still does not normalize for characters per occurrence or directly instrument cache misses, so token length and residency remain potential co-variables.

F. Reference-Relative NLP Evaluation

The canonical targets and pairwise groups come from lexical resources associated with Radixor’s default models; the standard default inputs are UniMorph-derived in the frozen catalog [6], [18]. A canonical target is not asserted to be a linguistic root or a unique context-sensitive lemma. The pairwise groups represent the available morphological/resource annotation, not semantic equivalence. Radixor has an in-resource informational relationship that external rule systems do not, so higher exact-target or pairwise conformity is not universal linguistic superiority.

G. Cross-Resource Association Is Not Causal

The 20 resources are heterogeneous and the structural predictors are highly collinear. Spearman coefficients, paired LOO gaps, partial-rank sensitivities, and harness restrictions expose descriptive robustness but cannot identify the marginal cost of a structural feature or generalize to an i.i.d. language population. Partial-rank values control one ranked scale variable at a time and are not causal adjustment.

H. No Uncontracted Same-Run Ablation

Every published Radixor timing row uses contracted compiled patch tries. No matched uncontracted 20-language JMH campaign exists under the same environment. Subtree contraction is therefore a documented architectural mechanism consistent with the representation, not an isolated causal effect size in this paper.

I. English Prefix Curve Is In-Resource and Path-Specific

The English evaluation includes training rows and follows one deterministic nested prefix; it is not randomized and is not a held-out causal learning curve. The dedicated coverage-suite 100% timing point is method-specific and cannot replace the canonical English comparator point. Exact-target counters are deterministic for the fixed dictionary, while timing uncertainty belongs only to JMH measurements.

J. Retrieval Effectiveness Is Outside Scope

Neither exact-target agreement nor pairwise morphological-reference conformity estimates indexing/query effectiveness. Changes in conflation can affect recall and precision differently by collection and query distribution. Downstream IR claims require relevance-judged retrieval experiments and are not inferred here [15], [16].

VIII. REPRODUCIBILITY

The publication bundle separates physical measurement provenance from article-level analysis. The article-facing snapshots reside in Radixor repository revision `0c3b13f` [6]. The physical 2026-08-25 performance campaign identifies Radixor/Java `4.2.0-6-g84e57fb`, core base revision `84e57fb27ae40913569e826858a8ebb07cf2ea01`, and a preserved dirty working-tree patch/untracked-source checksum state. This physical measurement identity is not rewritten as the later article-facing snapshot.

The article bundle contains:

- canonical 20-resource runtime projections with benchmark-family/method identities and normalized occurrence counts;
- 18 direct-Snowball rows containing primary and canonical Radixor denominator policies plus the machine-readable `stable_radixor_advantage` invariant;
- exact resource-wide pairwise TP/FP/FN/TN projections for deterministic PRIMARY_OUTPUT systems;
- 20 full-knowledge structural profiles from the archived edit-cost campaign and production preferred-output equivalence checks;
- the ten English deterministic-prefix timing and exact-target counter rows;
- `structural_contrasts.csv`, containing the internal-node minus training-row association gap, paired LOO gap range, and one-control partial-rank sensitivities;
- `scripts/analyze.py`, which regenerates all primary and sensitivity analyses; and
- `scripts/verify_analysis.py`, which independently recomputes cardinalities, the cross-policy 17+1 Snowball invariant, paired LOO structural contrast, partial-rank controls, harness/non-repeat restrictions, and English endpoints.

The analysis does not parse rounded Markdown tables. Provenance names upstream SHA-256 values for the dated speed, corpus, English-coverage, pairwise-quality, and edit-cost artifacts; compact source projections and derived outputs have separate manifests. Some upstream field names retain historical `root_*` terminology, while this article reports those counters as canonical-target quantities. LaTeX figures read derived CSVs directly. The repository root remains the stable article-facing locator; versioned snapshot and measured-source identities supply the immutable scientific provenance inside the text.

IX. CONCLUSION

A large lexical training source does not force a stemmer to perform large-dictionary lookup online. In the frozen Radixor

implementation, lexical mappings are compiled into shared trie paths and executable patch commands. On one workstation, the 20 canonical models process the benchmark’s known, lower-cased, reference-changing dictionary-field occurrences at 42.8–150.6 amortized ns/occurrence, with an unweighted resource median of 67.8. These values do not estimate overall natural-traffic latency.

Across 18 direct Snowball workloads, the two prespecified Radixor denominator policies give unweighted median Snowball/Radixor ratios of 2.33 (primary) and 2.57 (canonical). The cross-policy invariant identifies 17 stable Radixor advantages and Finnish parity. A separate resource-wide pairwise audit gives higher Radixor conformity to its declared overlapping morphological relation in all 18 matches, but that audit is training-related and uses a different input scope from timing.

The strongest general systems result is structural and associational. Internal nodes have $\rho = 0.931$ with canonical runtime versus $\rho = 0.723$ for training rows; the gap is 0.208 and remains positive (0.105–0.235) under every one-resource deletion. Internal-node partial-rank association remains 0.890 after controlling training-row rank and 0.875 after controlling timing-occurrence rank. Together with the common-harness, alternate-measurement, and non-repeated-corpus sensitivities, this supports the statement that measured runtime covaries more closely with compiled representation than with raw lexical-row scale in this fixed census.

The English deterministic-prefix series adds a separate within-language observation: exact-target reference agreement moves substantially along the prefix path while runtime point estimates are non-monotone. The paper does not isolate the causal benefit of subtree contraction, establish a hardware-independent ranking, evaluate OOV or natural production traffic, treat canonical targets as linguistic roots, or infer retrieval effectiveness from intrinsic conformity. Its contribution is a reproducible systems result: lexically learned conflation behavior can be compiled into a compact executable representation whose measured cost on known reference-changing forms is frequently competitive with direct rule-based baselines and is strongly associated with the structure produced by compilation.

REFERENCES

- [1] J. B. Lovins, “Development of a stemming algorithm,” *Mechanical Translation and Computational Linguistics*, vol. 11, pp. 22–31, 1968.

- [2] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130–137, 1980.
- [3] —, “Snowball: A language for stemming algorithms,” Snowball Project, 2001, accessed Sep. 2, 2026. [Online]. Available: <https://snowballstem.org/texts/introduction.html>
- [4] 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.
- [5] —, “Semi-automatic stemmer evaluation,” in *Intelligent Information Processing and Web Mining*. Springer, 2004, pp. 209–218.
- [6] —, “Radixor multilingual benchmark snapshot,” GitHub repository, article-facing snapshot 0c3b13f485a9ad0b460c0931e4497ea95bed66a1, 2026, frozen Java speed, exact-root, corpus, quality, generalization, and edit-cost data; accessed Sep. 2, 2026. [Online]. Available: <https://github.com/leogalambos/Radixor>
- [7] A. Georges, D. Buytaert, and L. Eeckhout, “Statistically rigorous Java performance evaluation,” in *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications*. ACM, 2007, pp. 57–76.
- [8] OpenJDK, “JMH: Java microbenchmark harness,” OpenJDK Code Tools project, version 1.37, 2023, version 1.37 released Aug. 3, 2023; accessed Sep. 2, 2026. [Online]. Available: <https://github.com/openjdk/jmh>
- [9] L. Weissweiler and A. Fraser, “Developing a stemmer for German based on a comparative analysis of publicly available stemmers,” in *Language Technologies for the Challenges of the Digital Age*, ser. Lecture Notes in Computer Science. Springer, 2018, vol. 10713, pp. 81–94.
- [10] Apache Software Foundation, “Apache Lucene,” Version 10.5.0 used by the Radixor benchmark adapters, 2026, accessed Sep. 2, 2026. [Online]. Available: <https://lucene.apache.org/>
- [11] K. R. Beesley and L. Karttunen, *Finite-State Morphology*. Stanford, CA: CSLI Publications, 2003.
- [12] J. ichi Aoe, “An efficient digital search algorithm by using a double-array structure,” *IEEE Transactions on Software Engineering*, vol. 15, no. 9, pp. 1066–1077, 1989.
- [13] J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson, “Incremental construction of minimal acyclic finite-state automata,” *Computational Linguistics*, vol. 26, no. 1, pp. 3–16, 2000.
- [14] C. D. Paice, “Method for evaluation of stemming algorithms based on error counting,” *Journal of the American Society for Information Science*, vol. 47, no. 8, pp. 632–649, 1996.
- [15] D. A. Hull, “Stemming algorithms: A case study for detailed evaluation,” *Journal of the American Society for Information Science*, vol. 47, no. 1, pp. 70–84, 1996.
- [16] F. N. Flores and V. P. Moreira, “Assessing the impact of stemming accuracy on information retrieval—a multilingual perspective,” *Information Processing & Management*, vol. 52, no. 5, pp. 840–854, 2016.
- [17] M. Miłkowski, D. Weiss, and contributors, “Morfologik stemming: Finite-state automata and morphological dictionaries,” GitHub repository; Morfologik 2.1.9 used in the reported benchmark inventory, 2026, accessed Sep. 2, 2026. [Online]. Available: <https://github.com/morfologik/morfologik-stemming>
- [18] K. Batsuren, O. Goldman, S. Khalifa, N. Habash, W. Kieraś, G. Bella, B. Leonard, G. Nicolai, K. Gorman *et al.*, “UniMorph 4.0: Universal morphology,” in *Proceedings of the 13th Conference on Language Resources and Evaluation*. European Language Resources Association, 2022, pp. 840–855.