Governed Context Compression: Measured Abstention as a First-Class Property

TekMyra compresses context for language models under a safety contract (locked-span preservation, refusal as a first-class outcome, and verification before emission) and counts every refusal against its own headline.

The TekMyra Team (LaconIQ) Correspondence: hello@tekmyra.ai August 31, 2026

Open core: Apache-2.0 github.com/laconiq-ai/tekmyra

ABSTRACT

Context compressors are evaluated on how much they remove. We argue and measure a prior question: what a compressor should be *allowed* to touch. TekMyra is a compression router: content is classified and routed to codecs under a safety contract with locked-span preservation, refusal as a first-class outcome, and verification before emission. Before anything is emitted, a verifier confirms that every locked span (an account number, a citation, a file path, a monetary amount, a policy identifier) is represented exactly once in the output; if the check fails, the compressor retries on a safer route, and if the retry fails too, it raises and emits nothing. A compressor that silently drops a dollar figure is worse than no compressor, because the output still looks fine.

On the public corpus we report: (1) a headline of **62.1458% effective byte reduction** over 138 externally sourced fixtures (2,708,761 → 1,025,379 bytes) with the **6 refusals stated beside it** (4.35% of fixtures, 3.55% of bytes, contributing zero saving), independently re-derived from production primitives by a second engineer; (2) per-cell figures on their honest denominators, including the ones that are thin (diff 0.0189%, the log codec 0.008%) and why they are published anyway; and (3) a comparison against LLMLingua-2 under a pre-registered design in which the context-window asymmetry (their model's 512-token native window against a corpus with median 3,416 tokens; exactly 1 of 138 fixtures fits) was declared and measured before any result existed. The corpus bytes and their attribution are reproducible without the trained artifacts; the rates are not.

On the shipped benchmark corpora we report reached-set ratios of **0.7409** on `synthetic` (28 reached of 28 eligible, 0 refused; +25.86% token reduction over the eligible set; 68/68 locked spans preserved) and **0.2929** on `long_context_v1` (26 reached of 40 eligible, **14 refused**; the 14 are counted at zero saving in the +48.44% corpus token reduction beside it; 704/704 locked spans on this build's denominator). The central claim is not raw compression supremacy: under the pre-registered comparisons of §4.6, TekMyra leads LLMLingua-2 on effective byte reduction over the 138 public fixtures (62.1458% vs 61.4640%) while LLMLingua-2 leads on cl100k token reduction (62.6016% vs 58.1101%), and, handed our own span map, its preservation exceeds ours. The claim is that the deployable question is what happens to the content a compressor should not touch, and that a system which measures and reports its own abstention is a different, auditable kind of instrument. One limitation is stated here rather than left to be discovered: the trained artifacts every headline rate requires ship as a separate release asset, and a clean checkout of the repository measures a no-op compressor.

CONTENTS

---

1 Introduction and motivation

Enterprise context includes protected spans: identifiers, legal text, monetary amounts, PII-adjacent content whose exact wording carries the meaning. A compressor that treats all bytes as equally deletable cannot be deployed against regulated content, because its failure mode is silent: a compressed prompt that is missing an account number does not look broken, it looks shorter. The model answers fluently from it, and the error surfaces far downstream, or never. Compression without governance is unaccountable deletion.

Most compression tools shorten first and repair after. TekMyra inverts that: it works out what must survive *before* it removes anything, and when it cannot guarantee survival it declines. The content passes through unchanged or the run refuses outright, and the run record says which mechanism declined and why. Routing, refusal, and verification-before-emission are the contract. Declining is a recorded outcome, not an error, and it counts against every headline number in this paper: a build that refuses more would otherwise score better for refusing, so refusals sit in the denominator.

This reframes what the headline figure is for. Under a compression headline, a refusal reads as a failure rate; under a governed-context headline it is the control working. The system declines what it cannot verify, and the refusal is measured, addressed (refusals have addresses, such as incident reports and high-risk threads) and reported. What no gate can currently prove, ours included, is that declining was *right*: that requires labelled ground truth with a recall figure, which is named as open work in §4.9, not claimed.

For context, without a survey: TekMyra sits adjacent to token-pruning prompt compressors of the LLMLingua family [1, 2], selective-context methods, retrieval-side summarization, and in-router compression inside model-serving stacks. §4.6 reports measured, pre-registered comparison work against LLMLingua-2, the strongest published baseline available to us; this paper otherwise makes no claims about other systems.

1.1 Three commitments

A paper about a compression system is easy to write badly: pick the friendly cases, report the good average, leave out what was skipped. This paper was written under three rules instead.

- **Falsifiable.** Every claim is stated so that a specific observation would refute it: a protected span that changed, an output shipped after verification failed. Comparisons follow pre-registered designs committed before any figure existed (§3.5), and unfavourable results are published by us first, which is what makes the favourable ones credible.
- **Verifiable.** Reported figures are derived from emitted run records; every figure carries its corpus, its denominator, its basis, the refusal count in the same table as the ratio, and, where a committed baseline exists, the commit that produced it. The method for re-deriving the figures is written out in §5.
- **Honest.** Runs where TekMyra declined to compress are included in every headline figure, not filtered out of the denominator. A system that abstains often and reports only its successes is reporting a fiction.

Claims of this paper, stated so they can be proved wrong

`C1` **Span preservation.** Every output TekMyra emits represents each locked span exactly once: verbatim, as an approved typed redaction marker, or as a token that resolves to the original. Refuted by one emitted output in which a locked span is absent, altered or duplicated.

`C2` **Fail-closed verification.** No output ships after verification fails on both the initial route and its fallback; the run raises `TerminalVerificationError` and emits nothing. Refuted by a shipped output whose record carries a failed final verification.

`C3` **Monotone risk.** The learned router may raise the risk tier but never lower it, and protected-span dispositions tighten monotonically with risk and never loosen. Refuted by any run record in which a learned decision lowered a tier or loosened a disposition.

`C4` **Honest denominators.** Every corpus-wide rate published in §4 counts refusals in its denominator at zero saving. Refuted by recomputing any published figure from the shipped fixtures and records and finding that reproducing it requires excluding refused runs.

The rest of the paper is organized around those claims: §2 describes the machinery that makes C1–C3 hold, §3 the measurement discipline behind C4 (including the audit that proves the discipline's own controls can fail), §4 the measured results with their honesty notes, §5 how to re-derive them, §6 what is deliberately not in the open repository, and §7 licence and provenance, including a disclosure of AI assistance in the codebase's development.

---

2 System architecture

This section describes the code in the open-core repository rather than the system as designed; where the shipped build does less than the commercial one, that is stated in the section where a reader would otherwise

assume parity (§2.7, §6). Every mechanism named here is implemented by a file the reader can open; the repository's `ARCHITECTURE.md` names them individually, and this paper keeps the same convention.

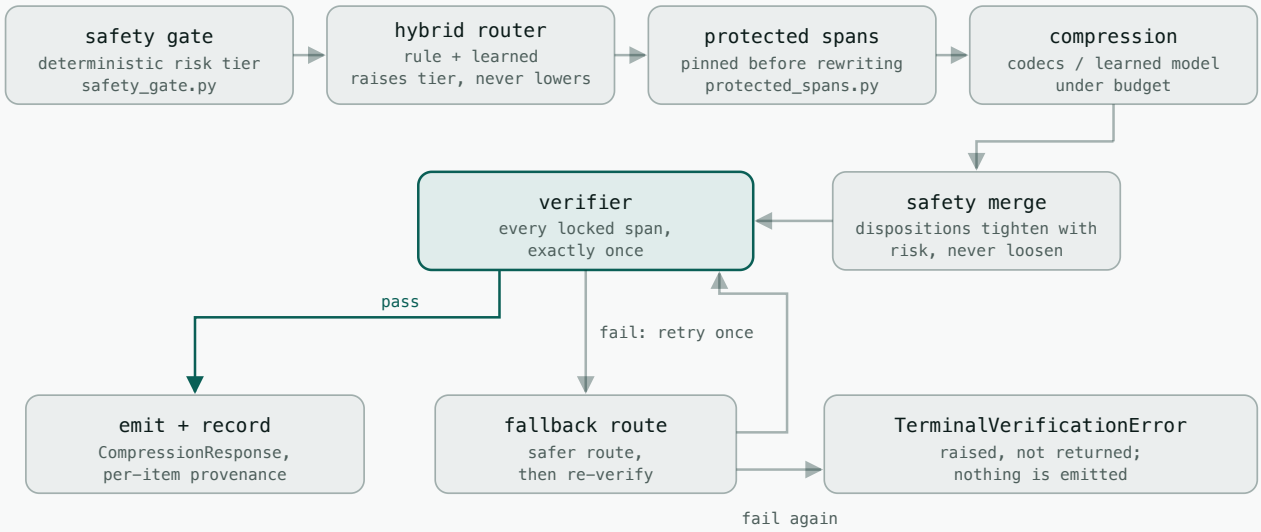

Figure 1: the request path. The verifier is the gate: a pass emits output with its record; a failure retries once through a fallback route and re-verifies; a second failure raises `TerminalVerificationError` and no response is constructed (`runtime.py`, verification at line 230, fallback at lines 258–283, the raise at lines 285–295).

2.1 Route classes and the two routers

The first decision about any content is what kind of content it is. Two routers answer that question. `rule_router.py` classifies deterministically by content shape. `learned_router/` is the learned classifier (features, heads, calibration, model, tokenizer window), and `hybrid_router.py` combines the two. The learned router may **abstain** from a decision; when it does, the rule router's decision stands and the record carries the tag `LEARNED_ABSTAINED` with a reason (`hybrid_router.py:164–165`). Disagreements resolve conservatively, and every routing decision carries provenance: which router decided, on what basis, with model version and artifact digest (`router_provenance.py`).

Routing interacts with risk in one direction only. A deterministic safety gate assigns a risk tier; the learned router proposes a route and **may raise the risk tier but never lower it**; a lossy route proposed at elevated risk is overridden to a reversible one. The route taxonomy includes safety routes (`reversible_retrieval_mode`, `legal_security_safe_mode`, `governance_safe_compression`, and PII redaction-before-compression) whose job is to transform conservatively, pass content through untouched, or refuse, rather than to maximize reduction.

Not every route in the taxonomy has a working compressor behind it, and the code says so rather than pretending. Routes in `UNIMPLEMENTED_CODEC_ROUTES` return `_unimplemented_result`, which fails closed:

they are roadmap entries rather than working compressors. At the time of writing, the sixteen routes split into six with working codecs, five that pass through untouched (`SAFE_TERMINAL_ROUTES`, §2.6), and five unbuilt. Both sets are derived from the codec map rather than declared, so the count can be asked of the code:

```
COUNT THE ROUTES RATHER THAN TRUSTING THE SENTENCE (AFTER THE §5.1 INSTALL)

python -c "from tekmyra.codecs.selector import _CODEC_MAP, SAFE_TERMINAL_ROUTES as s, \
UNIMPLEMENTED_CODEC_ROUTES as u; \
print(f'{len(_CODEC_MAP)} routes: {len(_CODEC_MAP)-len(s)-len(u)} working, {len(s)} passthrough, {len(u)} unbuilt')"
```

2.2 Protected spans

`protected_spans.py` detects spans whose exact wording carries the meaning: identifiers, amounts, dates, quoted terms. Detected spans are pinned before any rewriting is considered; later stages receive them as constraints rather than suggestions. Detection is regex- and rule-driven inside `protected_spans.py` itself. The vocabulary modules that sit beside it in the package (`pii_vocabulary.py`, `rights_vocabulary.py`, `corpus_vocabulary.py`) look like its inputs and are not: they serve fixture generation and measurement, not detection. The only first-party modules `protected_spans.py` imports are `contracts`, `content_hash` and `fixtures`, a claim `ARCHITECTURE.md` backs with a one-line AST check the reader can run against the source.

2.3 Codecs and the learned compressor

Two engine families do the shortening, selected by route:

- **Deterministic codecs** (`tekmyra/codecs/`): `json_codec.py`, `log_codec.py`, `code_codec.py`, `diff_codec.py`, `stacktrace_codec.py`, with `base.py`, `safe_mode.py` and `selector.py`. Given the same input they produce the same output every time, with no model involved.
- **Learned compressor** (`tekmyra/learned_compressor/`): learned sentence selection for prose, running `segmenter.py` → `features.py` → `model.py` (or the transformer variants, executed through `tekmyra/onnx_runtime.py`) → `selector.py`, under a budget from `tekmyra/compression_budget.py`.

The learned model is a versioned release artifact with a fingerprint (`version` fields and the sha256 capture in `router_provenance.py` are what make that checkable), and it is **not in the repository**. §5.2 gives the procedure for fetching it and §6 states why it is closed. Every number in this paper states which execution path produced it: the benchmark figures in §4 were measured via the joblib/scikit-learn path, with no `onnxruntime` installed.

2.4 The safety gate and the merge

`safety_gate.py` classifies risk (`classify_safety(features) → SafetyDecision`), and `safety_merge.py` applies the merge that locks spans (`apply_safety_merge`). Each protected span receives a disposition (preserve, redact, anchorize, or human-review); a protected span becomes a **locked span** at this point, and

the counts in §4 and claim C1 are on the locked set. Dispositions tighten monotonically with risk and never loosen. Neither the gate nor the merge makes the final ship-or-decline call; that belongs to verification.

2.5 The fail-closed verifier

`verifier.py` re-checks every pinned span against the compressed output. `entity_fidelity.py` and `numeric_fidelity.py` hold the span-class checks; `sentence_boundaries.py` guards structural cuts. The contract is the one from the abstract: every locked span must be represented exactly once: verbatim, as an approved typed redaction marker, or as a token that resolves to the original.

`runtime.py` verifies (line 230). On failure it retries through a fallback route if one exists (lines 258–283) and re-verifies. **If the fallback also fails, it raises** `TerminalVerificationError` (lines 285–295). It does not quietly return the original: it refuses to construct a response at all, because the alternative is returning output the verifier has rejected. Callers must handle that exception. This is why claim C2 holds: a verifier failure is not a soft degradation, it is terminal by design, and a verifier failure can never ship a degraded prompt.

2.6 Three ways TekMyra declines, and two counts

"It abstains" is one phrase covering three distinct mechanisms with three different owners. They fail differently, so they are kept apart. Throughout the title, the abstract and §1, *abstention* carries the broad sense: declining to transform, by any of the three mechanisms below. From this section onward *abstention* is the narrow benchmark count defined here, disjoint from *refusal*; §4.4's "abstentions are zero" is a statement about the narrow count alone.

1. **The router abstains.** The learned router declines to classify; the rule router's decision stands. Owner: `hybrid_router.py:106–165`. Visible in the record as `LEARNED_ABSTAINED` plus a reason.
2. **The codec declines to transform.** The five routes in `SAFE_TERMINAL_ROUTES` (`passthrough`, `pii_redaction_before_compression`, `governance_safe_compression`, `reversible_retrieval_mode`, `reject_or_human_review`) share one `_passthrough` codec that returns its input verbatim, and the prose codec also returns its input untouched on HIGH and CRITICAL risk tiers. Owner: the codec layer, driven by the risk tier from `safety_gate.py`.
3. **Verification fails.** The terminal case of §2.5. Owner: `runtime.py`.

The benchmark keeps two counts from those mechanisms, computed from disjoint populations. A **refusal** is a fixture that raised `TerminalVerificationError`: it is appended to the failure list with its failed checks and never reaches the results table at all (`benchmark_compressor.py:287–310`). An **abstention** is a fixture that *completed* on a passthrough route (`reversible_retrieval_mode`), that is, completed untouched, and is filtered out of the results rows afterwards (`abstention_rows`). The report prints them on separate lines with separate labels. This paper does not roll the **abstention** count up to a single mechanism from the list above, because the benchmark does not record which one produced it; the refusal count is mechanism 3 by definition. Both outcomes count as saving nothing in every corpus-wide figure (§3.2), and the distinction is load-bearing when reading §4.4.

2.7 Locked spans and the anchor contract

When a protected span is anchorized rather than preserved verbatim, the content removed on a reversible route is given an addressable identifier rather than simply being gone. `anchor_interface.py` defines that contract, `AnchorStoreProtocol` (line 49), and five modules consume it: `runtime.py`, `verifier.py`, `safety_merge.py`, `router_provenance.py`, `benchmark_compressor.py`.

The store the open repository ships **resolves nothing**. `NonResolvingAnchorStore` (line 72) mints identifiers and then cannot resolve them: `get()` returns `None` for every identifier, including ones it minted itself (lines 117–119). The consequence is a behaviour, not a gap: this build refuses more than the commercial one, because it cannot ask "was that dropped span recoverable?" and so treats every drop as unrecoverable. It is not an empty store: a store that happened to be empty would behave differently once populated. It is a store that *cannot* resolve, which is the stronger position. §4.4 measures what this seam costs and §6 gives its full statement.

Per-item records are assembled by `runtime.py` against the typed contracts in `contracts.py`: the `CompressionSegment` and the `CompressionResponse`. The per-run telemetry emitter and the HTML console are not in the repository; they are internal operations tooling, excluded by category (§6). The contracts give the shapes and the runtime shows the assembly; the writer and the renderer are the integrator's to supply.

---

3 Measurement methodology

3.1 Where the figures come from

Nothing in §4 is measured by an observer standing outside the system. Every figure is arithmetic over records the system wrote while it ran: the benchmark harness aggregates per-item records at run time, and the paper reports those aggregations with their method. There is no second source, no adjustment, and no sampling step where inconvenient runs could quietly drop out. The repository states this as a rule about where numbers are allowed to live at all:

Table 1: where numbers are allowed to live (from `ARCHITECTURE.md` §7)

SURFACE	WHAT IT MAY SHOW
Run records	the facts, as emitted
The benchmark harness	aggregations derived from records at run time
The paper	measured results with method, traceable to records
The website	derives nothing; it may quote the one committed baseline figure verbatim, with its denominators

A figure that cannot name its corpus, its denominator, its basis and the execution path that produced it does not go in a table. Every figure in §4 carries all four.

3.2 Refusals stay in the denominator

The central discipline is refusal accounting. When verification fails terminally, the fixture is recorded as a refusal with its failed checks and counted at **zero saving**: the refusal stays in the corpus denominator at ratio 1.0. The reason is an incentive argument: a refusal saves nothing and must be counted as saving nothing, because a build that refuses more would otherwise score better for refusing. A system whose safety mechanism improves its own headline is a system whose headline has stopped measuring compression.

That incentive claim is measured rather than asserted: the headline harness's audit (§3.6) includes a computed control for exactly this mechanism. The control is a constructed corpus of ten hand-set rows, five of them refused, the refused half carrying half the input tokens: the headline reads 30.0% while the on-accepted diagnostic reads 60.0%; refusing *costs* the headline. (An earlier revision of this paper printed 0.0% for that control, transcribed from a fixture whose output total was mis-set; the fixture correction accompanies this revision, and the control's assertion, headline strictly below diagnostic, held under both.)

The same rule covers abstentions and passthroughs: a fixture that completes untouched contributes its full size to the denominator and nothing to the numerator. Declining, in every one of the three mechanisms of §2.6, is visible in the corpus-wide rate.

3.3 A denominator the system cannot choose

Refusal accounting is a special case of a general rule: **any published rate needs a denominator the system cannot choose.** A benchmark denominator is the full fixture set as shipped (on `long_context_v1`, all 40 fixtures), and the system cannot shrink it by refusing, abstaining, or passing through: every path it can take leaves the denominator intact. Where a corpus ships fixtures outside the routes under test, the denominator is the eligible subset: fixed before the run, stated in the table (§4.3), and still not a set the system can shrink at run time.

For the public corpus the binding is committed. The denominator is 138 fixtures, 2,708,761 bytes, and the evidence artifact (`tests/baselines/public_corpus_baseline.json`) carries the figure, its denominators, the corpus file counts and the sha256 digest of each corpus, binding the visible fixture bytes to the measured baseline at commit `14c63bc`. The headline was additionally re-derived independently: a second engineer, with their own byte accumulation against production primitives, matched it to four decimal places (2026-08-10).

The complementary figure, the *on-accepted diagnostic*, divides by the set of fixtures the system accepted, which is precisely a denominator the system chooses. That is why it is labelled a diagnostic and is never the headline: it answers "how well does the compressor do on the work it agreed to take?", which is useful for engineering and useless as a promise. Wherever both appear in §4, the refusals-included figure is the headline and the on-accepted figure is subordinate to it.

3.4 Basis discipline

Figures in this paper are on stated bases. The compressor benchmark (§4.3) reports a **reached-set ratio** (the share of content kept, averaged over fixtures the compressor reached) alongside corpus-wide **token** reduction with refusals in the denominator; fidelity results are in points on a declared span class. Figures on different bases are not the same kind of number, are never compared directly here, and never share a table without their basis named; §4 keeps each basis in its own table.

The public-corpus figures of §4.1–§4.2 are on a **byte basis** and are not comparable to any ratio or token figure in this paper.

3.5 Pre-registration, and publishing the unfavourable result first

Comparisons follow pre-registered designs committed *before* any figure existed, and unfavourable comparisons are published by us first, because that is what makes the favourable ones credible.

Concretely: the LLMLingua-2 context-window asymmetry was declared at commit `c645049`; the three-method comparison design is PRODV1-52 at `4275a80`; the symmetric two-point budget design is PRODV1-54 at `c036bc1`; the round-four fidelity protocol was committed at `99bf815`. Where a pre-registered falsification clause fired against our own claim, the result is in this paper (§4.6) with the harness's conclusion quoted rather than softened.

3.6 Can the controls fail? The falsifiability audit

A control that cannot fail certifies nothing, so every harness underwriting a published figure is audited for controls that cannot fail. The audit harness and its artifacts live in the private measurement tree; this section states its scope and results exactly, and names what a clone can and cannot re-run.

For the round-four fidelity harness (§4.6): its seven stop conditions decompose into **thirteen checkable claims**: the honest unit, since a single condition can be half-live. The pre-repair mutation baseline found **three** of them proven incapable of failing and **two** indistinguishable on failure, and that failing baseline was committed as a red table *before* any repair landed (`mutation_harness_red_baseline.json` in the private measurement tree); it is a recorded artifact, not a post-hoc claim. The integrated tree measures **eleven live plus two distinguishable, zero vacuous**, with the green table committed beside the run it certifies (`9a6be96`). Both tables are records of the private history; the public repository is a snapshot and does not carry them.

For the headline harness (no structured stop suite to mutate): 85 constructs were enumerated and the 13 load-bearing ones fault-injected, each shown to move under its own defect. The scope is stated exactly: 13 of 85; the remaining 72 are guards, defaults and plumbing, classified by enumeration and not individually injected. All eight computed controls are live, including the refusal-accounting control of §3.2, the empty-denominator control (which yields `None`, never 0.0), and the beaten-on-ratio reporter, which flips against us under inverted inputs. One finding is disclosed rather than buried: four provenance entries were *declared* (fixed strings reading as facts about the run) rather than derived, so the run could not falsify them. Nothing

was factually wrong, and the class was repaired before release: mode fields are now derived from the runner's actual construction (an unrun runner reports NOT CONSTRUCTED rather than an intention), and the one remaining declared field carries the label "DECLARED BELIEF, NOT A MEASUREMENT", because the harness deliberately declines to construct the other modes rather than attempt-and-catch to satisfy a provenance rule. Post-repair, with the newly derived provenance fields joining the injectable set (a derived field can be fault-injected where a declared string cannot), the audit reads 15 injected: 13 computed injections that move under their own defects, plus 2 checks that record a fixed truth rather than a computed one ("MeetingBank status line" and "truncate.refused"), listed as structural rather than proven falsifiable.

3.7 Attributing the saving: `codec_transformed`

A route name does not tell you whether compression happened, and a per-route token delta is not evidence that it did. Two things produce a reduction: the codec, and the safety merge's anchorization; the merge's saving lands on whatever route the fixture happened to take. During development, two routes were found reporting non-zero token reductions while their codec was a no-op on every fixture: anchorization credited to compression. That finding is an engineering note recorded in a source comment, not a shipped artifact: no result file in the repository carries it, and reproducing it needs the separately fetched model artifact.

The fix is in the record rather than in the prose. Each item carries `codec_transformed`, computed by comparing the codec's own input against its own output; both `apply_codec` call sites (initial and fallback) are fed `merge_result.transformed_text`, so the comparison is exact on either path. When reading the numbers in §4, or numbers from your own runs, `codec_transformed` is the field that separates "the compressor worked" from "something upstream shortened the text".

---

4 Results

All rates in this section were measured with the reference artifact bundle installed (`.cache/compressor/1.0.0` and `.cache/router/1.0.0`; §5.2), via the joblib/scikit-learn execution path with no `onnxruntime` installed. That qualifier is load-bearing and is repeated in §4.10 rather than left to the reproducibility section: **a clean checkout of the repository measures a no-op compressor** and cannot produce these rates.

4.1 Public-corpus headline: 138 fixtures, byte basis

Table 2: public-corpus results · byte basis · refusals in the denominator at ratio 1.0 · commit `14c63bc`

QUANTITY	VALUE	NOTE
Corpus	138 fixtures	67 `public_domain_v1` (Congressional Record), 36 `public_code_v1`, 29 `public_diff_v1`, 6 `public_json_v1`; all externally sourced, rights-inventoried; 2,708,761 bytes
Effective reduction (bytes)	62.1458%	the headline; 2,708,761 → 1,025,379 bytes; denominator includes refused content at ratio 1.0
On-accepted reduction	64.4356%	diagnostic; accepted-only denominator, one the system chooses, so never the headline (§3.3)
Refusals	6 of 138	4.35% of fixtures, 3.55% of bytes, reported here rather than in a footnote; each contributes zero saving
Positive control	81.8222%	`legal_security_safe_mode`/`long_prose` cell, 49 fixtures; the harness measures a large rate where one is achieved
Verification	independent	second engineer, own byte accumulation against production primitives; exact match to four decimal places (2026-08-10)

The gap between 62.1458% and 64.4356% is the honest cost of counting refusals: about 2.3 percentage points of headline, paid so that the safety mechanism cannot improve the score by firing more often. A reader who sees the on-accepted figure quoted alone, anywhere, is seeing a denominator the system chose.

4.2 Per-cell results on honest denominators, byte basis

The blended corpus mean is not the product number. Below are the positive control and every cell that earns almost nothing, reported on what each accepts with refusals beside it, at the same precision as the flattering ones; cells not shown fall between them. Cells are route × content-type pairs and do not map one-to-one onto the four `public_*_v1` corpora:

Table 3: per-cell effective byte reduction · byte basis · same run as Table 2 · refusals in the denominator

CELL	EFFECTIVE BYTE REDUCTION	DENOMINATOR HONESTY
`legal_security_safe_mode` / `long_prose`	81.8222%	49 fixtures; the non-zero positive control, which shows the near-zero cells below are measurements rather than measurement failure
`code`	2.08%	3 of 11 fixtures refused; refusal is the safety contract working, and it is priced in this figure
`diff`	0.0189%	24 fixtures; effectively nothing on real diffs; reported despite being unfavourable
`log_compressor` / `code`	0.008%	12 code-typed fixtures (the public corpus ships no log-typed fixtures); the folding codec found almost nothing here, and the corpus artifact records this as a generator finding: real logs repeat heavily where these fixtures vary; reported despite being unfavourable
`pii_redaction` (three cells)	zero	3, 4 and 4 fixtures; all three cells at zero; a zero on a redaction-bearing safety route is the contract priced into the table, not a defect

Rates are as measured; they are not restated at a common precision because rounding a published figure is a change to it.

A related mechanism is worth stating because it flatters every compressor that does not report it: corpus size. Applied directly to all six real JSON documents the deterministic JSON codec measured 9.43%, against 30.53% on the synthetic corpus (ratio basis, 2026-08-11 baseline), a 21-point fall we report because small uniform synthetic documents flatter overhead-removal codecs. Under production routing the shipped baseline records 17.5079% for this codec on real documents, over the single fixture the router sends to it (the other five route to PII redaction and human-review at zero saving); and the six real JSON documents are effectively three independent shapes, four being near-duplicates. Two denominators, both stated.

4.3 Compressor benchmark: ratio and token basis

With the reference artifact bundle, on the two benchmark corpora shipped in the repository, via the joblib/scikit-learn path:

Table 4: compressor benchmark · ratio/token basis · refusals in the token denominator at zero saving

CORPUS	FIXTURES	MEAN SIZE	RATIO ON REACHED FIXTURES	TOKENS	LOCKED SPANS
`synthetic`	28 eligible of 301 shipped (28 reached, 0 refused)	250 chars (over 301 shipped)	0.7409	+25.86%	68/68
`long_context_v1`	40 eligible of 40 shipped (26 reached, **14 refused**)	23,644 chars	0.2929	+48.44% / +70.40%*	704/704

The ratio column is the share of content kept, averaged over the fixtures the compressor reached; it is a reached-set diagnostic, not a headline (§3.3). Tokens is the refusals-included figure: token reduction with refusals in the denominator at zero saving, over the eligible set; *the second `long_context_v1` figure is the reduction on the prose route alone. Ratio and token figures are never comparable to byte-basis figures (§3.4). Both rows are routed blind, and some fixture labels name a codec that is declared but never built: 20 of synthetic's 28 eligible fixtures (`summarizer`) and 20 of `long_context_v1`'s 40 (`extractive_keypoint_compressor`). Those fixtures were compressed by `prose_token_compressor` rather than the codec their labels specify. An improvement on them means "compressed with a different codec than the labels intended"; the benchmark derives and prints this caveat per corpus beside the number. Rates are on the eligible denominator: a fixture is eligible when its content type has a codec on its route in this build; eligibility is a fixed property of the shipped corpus, decided before any run, and no path the system takes at run time can change it. `synthetic` ships 301 fixtures of which 28 are prose-eligible; `long_context_v1` ships 40, all eligible.

4.4 What the 14 refusals mean

The 14 refusals on `long_context_v1` are refusals, not abstentions; both words appear in this system and they are different outcomes (§2.6). What follows was measured on both builds at the same commit, with the anchor store as the only variable.

- **Abstentions in the §2.6 sense are zero in both builds.** An abstention is a fixture that completes untouched. None did.
- **The 14 are refusals, and they exist only in this build.** The commercial build completes all 40. The refused fixtures are the same 14 either way, and the 26 this build completes are a strict subset of the commercial build's 40.
- **They fail three checks at once** (`locked_spans_preserved`, `facts_preserved`, `anchors_resolvable`), not the anchor check alone. This build cannot resolve anchors (§2.7, §6), so spans the commercial build recovers are unrecoverable here, and verification refuses rather than shipping the loss.
- **They are not "routed to" `reversible_retrieval_mode`.** That is where they land after their first route fails verification; the router sent them to `legal_security_safe_mode` (6) and `governance_safe_compression` (8). Routing is identical in both builds (same routes, same ratios on every shared fixture), so the anchor store changes what survives verification, never where anything was sent.

This build is not worse at compressing. It reaches the compressor on exactly the same 26 fixtures and produces the identical 0.2929. It refuses 14 that the commercial build completes, for a stated reason, and counts them as saving nothing. A system that compresses 26 of 40 and refuses 14 rather than ship an unverifiable result is making a different promise from one that compresses all 40; that difference is what the open/closed seam costs, and it is stated here rather than smoothed over.

4.5 Locked spans: name the build before quoting the ratio

Both builds report locked-span preservation of 1.000 over different denominators: **704/704** here, **1785/1785** commercial, on `long_context_v1`. The gap is the 1,081 spans belonging to the 14 refused fixtures, which leave this build's count with the fixtures. Span *detection* is identical across both builds on every shared fixture. On the `synthetic` corpus this build preserves 68 of 68. Preservation of what we were told to keep is a provenance fact, not a recall claim; §4.9 states that distinction in full.

4.6 Comparison: LLMLingua-2, pre-registered

We compare against LLMLingua-2 (`microsoft/llmlingua-2-xlm-roberta-large-meetingbank`) [2, 3], the strongest published baseline available to us, under the pre-registered designs of §3.5.

The asymmetry, declared before the run

The baseline model has a 512-token native context. Under `cl100k_base` our public corpus measures min 482 / median 3,416 / max 18,230 tokens: **exactly 1 of 138 fixtures (0.7%) fits their window.** LLMLingua-2 chunks internally and produces valid output, so the comparison is sound, but it compares a method inside its native regime against one that must chunk to see the input at all. This asymmetry cuts toward us, which is exactly why it was committed to the pre-registration before any result existed (commit `c645049`). Their published 2–5× profile comes from corpora sized to their window [2]. Both statements are true and both are stated: this is not a fair fight, and it is an accurate measurement of what happens on real long documents.

Rate-controlled vs quality-controlled

LLMLingua-2 is rate-controlled: you request a ratio, you receive it, quality is the dependent variable (documented default rate 0.5); its public checkpoint is trained on the NC-licensed MeetingBank corpus [3]. TekMyra is quality-controlled: it delivers what the safety bar permits and refuses when it cannot. The pre-registered rounds measure both bases over the same 138 public fixtures, and the two bases disagree for a mechanistic reason worth stating: a token-pruner removes token-dense, byte-cheap material (function words, punctuation), while sentence selection removes byte-rich prose, so byte and token rates order the two systems differently. Round one (committed at `0a561fc`): LLMLingua-2 leads on cl100k token reduction, 62.6016% to 58.1101%. On effective byte reduction, TekMyra leads, 62.1458% to 61.4640% (round-two artifact `force_tokens_round2/20260811_021158`); the byte basis was committed as this paper's headline denominator before any comparison figure existed, not chosen after. Round two's force-tokens arm, the experiment a hostile reviewer would run: handed our derived span map, LLMLingua-2's preservation exceeds ours (99.4% vs 97.3% all-spans; 100% vs 96.5% must-preserve) while still leading on tokens. That is the pre-registered flip condition, published as triggered: the preservation advantage is a property of the span map,

not of our compressor. What remains ours by measurement: the map itself, the refusal accounting (we refuse 6 of 138 and count them at zero saving; LLMLingua-2 refuses none by design), and fail-closed verification. On the held-out `long_context_v1` split the route-level artifact ratios are 0.8235 (ours, over the 3 of 8 held-out fixtures the prose route reached) against their 0.8109 at target 0.82 over all 8: a 0.0126 gap on a small n, printed with its denominators; both sides are recorded in the private measurement tree (the `llmlingua2_held_out` runs), theirs measured via their library's own pinned checkpoint. An earlier revision of this paper compared their figure to our codec-stage ratio (0.8114), a stage metric retired by the anchor-accounting audit because it cannot see anchor overhead; the artifact-level figures are the ones printed here. We publish the decomposition rather than the blend: different control contracts, so each number above carries its basis, and the systems differ in what they are willing to touch.

Three-method comparison: named absence, and an unfavourable finding

The full three-method results table (ours / LLMLingua-2 / truncation floor, 138 fixtures) has not landed at publication: the PRODV1-52 harness is in review, its pre-registration committed, and the comparison ships as in-progress rather than as an implied result. One result from that harness is reported now, because it is unfavourable and its falsification clause fired without being asked: **on the locked secrets-and-identifiers class, LLMLingua-2 matches us exactly, and so does plain truncation, which has no preservation logic at all.** The harness's own conclusion, quoted rather than softened: the cut has no discriminating power on this corpus, and our central differentiator is "unevidenced on this corpus rather than refuted", with n too small to distinguish those. This is a pre-registered falsification clause producing a real negative about our own claim; we publish it first, state what would evidence the differentiator (a corpus whose locked-class items truncation actually destroys, i.e. locked spans positioned beyond naive budget boundaries, at adequate n), and treat building that corpus as named open work. The symmetric two-point budget comparison (PRODV1-54, pre-registered at `c036bc1`) has not been run at this revision; like PRODV1-52 it is named here rather than left to be noticed absent.

Fidelity under the derived-span map (round four)

Complete: all seven stop conditions pass on all 138 fixtures. The pre-registered question: does the *derived* span map outperform a random list of identical shape at equal token cost; that is, is it *which* tokens are forced that matters, not *how many*?

Table 5: fidelity by arm · all-spans class, `cl100k`, 138 fixtures · points

ARM	ALL	MUST-PRESERVE	LOCKED
Ours (derived map)	95.9402	95.5266	100.0
Control B (declared regex)	89.3162	86.6729	89.4737
Control C (random, size-and-length-matched)	82.3362	83.9702	94.7368

The measured ordering is the one the thesis predicts: the derived map leads the matched random control by **13.60 points**, with the declared-regex control between them. Whether that lead clears a margin is not answered here. A framing constraint from the artifact's own verdict machinery is **MARGIN_ABSENT**: the blind margin procedure has not returned, so measurements are reported and *no verdict exists*; the table and its ordering are printed, and no pass/fail claim against a margin is made, because per the artifact's own words, none may be manufactured.

Two-column honesty on the truncated subset (13 documents at the baseline library's force-token cap of 100; a model property of the pinned checkpoint, not a setting): forced-list fidelity 85.5769 on the gold denominator; 69.4763 against arm B's own full natural list. Both columns ship; the favourable one is not chosen for us. A locked-class caveat, printed rather than discovered: on the locked class the random control (94.74) outscores the declared-regex control (89.47), small-n behaviour consistent with the three-method finding above that the locked class under-discriminates on this corpus. Protocol committed at `99bf815`; evidence recorded in the private measurement tree at round4/integration `9a6be96` (`round4_maps_compared_20260811`); run at `e202409`, offline enforced, 481.9 s.

4.7 Corpus composition and attribution

The 138 public fixtures divide by provenance: **67 fixtures** are US-Government public-domain material (the Congressional Record), and **71 fixtures** are attribution-bearing, drawn from exactly six open-source projects: `rich`, `express`, `nlohmann` and `pytest` under MIT, and `httpx` and `golang` under BSD-3-Clause. Every fixture names its exact source and licence in its own provenance block; the repository's `NOTICE` file carries two sections: the public-domain material and the full attribution with licence text for the six projects (§7.2).

4.8 Reading the refusals as product behaviour

Under a compression headline, 6 of 138 refused reads as a failure rate; under a governed-context headline it is the control working. The system declines what it cannot verify, and the refusal is measured, addressed, and reported; refusals have addresses, such as incident reports and high-risk threads.

4.9 Honesty notes, kept in the paper

What these results do not claim

- Locked-span preservation is a provenance fact (we did not destroy what we were told to keep), *not* a recall claim that we caught everything that should be hidden. We have no false-negative measurement and therefore claim no recall. "Zero PII survival" is not claimed anywhere in this paper.
- Neither our gate nor any baseline's can currently prove that a given refusal was *right*: that requires labelled ground truth with a recall figure, which is named as open work, not claimed.
- The synthetic-corpus figures that preceded the present corpora were measured on paragraph-sized snippets and are superseded; small uniform synthetic documents flatter overhead-removal codecs.
- The learned router does not participate on deterministically-routable content; prose figures are compressor figures, not router figures.
- Downstream answer-quality effects of compression are real and were measured in early paired runs: compressed correctness at or below raw on every provider lane tested (WP-1, 2026-07-15). The current pipeline's downstream figures have not been re-measured; that prior measurement is stated with its date rather than silently carried forward.

4.10 The limitation, stated where the numbers are

What a clean checkout measures

A clean checkout of this repository **measures a no-op compressor**. The trained artifacts, `.cache/compressor/1.0.0` and `.cache/router/1.0.0`, ship as a separate release asset, and every headline rate in this section requires them. Without them, the benchmark refuses to produce a number rather than reporting a flattering 1.000, and a direct API call degrades to passthrough: safe for protected spans, useless as a measurement (§5.4). The headline rates of this section are unobtainable from a clean checkout.

What a clean checkout *can* reproduce without the artifacts: the public-corpus fixture bytes, their digests, and the attribution coverage (§5.3). What it cannot: any figure in Tables 2–5 (Table 5's evidence is recorded in the private measurement tree, §4.6).

5 Reproducibility

This paper publishes with the open-core release: the benchmark harness, the fixture corpora, and this paper are public together. The mutation harness that proves our controls can fail is in the private measurement tree (§3.6); its scope and results are stated there rather than re-runnable here. Diff the paper against the repository; that is what it is for.

That includes the 138-fixture public corpus behind §4.1: it ships in this release, rights-inventoried, with its two-section attribution in `NOTICE` (§7.2).

5.1 Install against the constraints file

Dependencies are declared as ranges so TekMyra can live inside an application; ranges float, so the published figures were measured against the exact versions recorded in `constraints-reproduce.txt`. To reproduce the numbers rather than just use the library:

INSTALL FOR REPRODUCTION

```
pip install -c constraints-reproduce.txt -e '.[dev]'   # library + pytest, pinned for reproduction
```

How much the constraints matter is measured rather than asserted, against the published reference bundle: on scikit-learn 1.8.0 and 1.9.0 the `long_context_v1` headline is **0.2929 both times**, identical to four decimals. Below the declared floor the artifact does not degrade, it refuses to start: the bundle's router was exported under scikit-learn 1.8 or newer, whose `LogisticRegression` no longer carries the removed `multi_class` attribute, so 1.5.2, 1.6.1 and 1.7.2 all crash at model load with an `AttributeError` (an earlier export of the model did run on 1.5.2; the floor moved when the artifact did, which is why `pyproject.toml` declares `scikit-learn>=1.8`). The constraints file is a durability guarantee within the declared range, not the only thing holding the figures up.

5.2 The reference artifact bundle

Trained model artifacts are not in the repository and never will be; they are the commercial half (§6). The reference bundle is published as a release asset, `tekmyra-reference-artifacts.tar.gz`, with its SHA-256 printed in the release notes. Check the digest before extracting:

FETCH AND VERIFY THE REFERENCE ARTIFACTS

```
curl -L -O https://github.com/laconiq-ai/tekmyra/releases/latest/download/tekmyra-reference-artifacts.tar.gz
shasum -a 256 tekmyra-reference-artifacts.tar.gz   # compare against the release notes
tar xzf tekmyra-reference-artifacts.tar.gz         # unpacks .cache/compressor/1.0.0 and .cache/router/1.0.0
```

The digest is published beside the asset rather than pinned in the documentation, so that a reader checks the artifact they actually downloaded against the record for that release rather than against a constant a later release would silently falsify. Artifact directories resolve in a stated order (`$TEKMYRA_ARTIFACT_ROOT`, then `.cache/` under the working directory, then `.cache/` beside the installed package), and a failure prints every location it tried.

The artifacts cannot be retrained from the repository; that is stated here rather than left to be discovered. The training corpus and the trainer are deliberately not published, under the same rule that keeps trained artifacts closed. What a cloner can verify is receipt-of-our-bytes (the bundle is verifiable as a receipt; compare its digest) and every benchmark figure produced from those bytes; what a cloner cannot do is re-derive the weights. No tool in the repository claims otherwise.

5.3 The commands

REPRODUCING WHAT IS HERE

```
python -m tekmyra.benchmark compressor --fixture-dir benchmarks/fixtures/synthetic
python -m tekmyra.benchmark compressor --fixture-dir benchmarks/fixtures/long_context_v1
python -m pytest tests                    # the suite, green without a model
python scripts/verify_open_core.py        # the seam, proved rather than asserted
```

On the public corpus, two further checks. `tests/test_public_corpus_notice.py` runs on a clean checkout: it verifies that the public-corpus fixture bytes match the committed baseline and that every third-party source is covered by `NOTICE`. It does **not** reproduce the 62.1458% headline: that also requires the trained artifacts, which the reference bundle supplies as the versioned `.cache/compressor/1.0.0` and `.cache/router/1.0.0` directories the code loads. (The baseline additionally records a digest over the full private `.cache/router` directory, of which the bundle is the loadable subset; the README's evidence section enumerates which recorded digests verify against the published tree.) With the reference artifacts installed, the rate itself is checked by the ratchet test:

PUBLIC CORPUS: BYTES, ATTRIBUTION, AND THE RATE

```
python -m pytest tests/test_public_corpus_notice.py    # public bytes + attribution, clean checkout
python -m pytest tests/test_public_corpus_ratchet.py   # the headline rate, artifacts required
```

5.4 Refusal is the default

On a bare clone with no trained artifact, the default benchmark invocation does not emit 1.0, or 0, or a quiet pass-through: it **refuses**, naming the missing artifact directory, printing every path it searched, and offering the explicit escape hatch (`--allow-rules-only`), and it states that an escape-hatch run cannot emit a

headline compression ratio. Under that flag the run is honest about what it is: the headline is **withheld**, reported as neither zero nor one; token reduction is 0.00%, and 68 of 68 locked spans are still preserved across 28 reached fixtures. The safety machinery works without a model; only the compression claim is withheld until a model is present. With the reference bundle in place, the published figures reproduce. This is the governed-context thesis demonstrated by the repository's own default behaviour rather than asserted by its authors.

5.5 Proving the seam

`scripts/verify_open_core.py` is the check worth running. It builds the published surface from the allowlist (§6.3), proves every closed module is **unimportable** before measuring anything, runs the test suite inside the built tree, and refuses to print a number if any of that fails. Its output includes the by-route table, so the declining behaviour of §2.6 is something a reader can watch rather than take on trust. The suite itself passes without a model: tests that need a trained artifact skip, and each skip names every location that was searched.

One shipped script is explicitly not for the reader: `scripts/payload_manifest.py` is an internal check that proves every published file was derived from our private source at a stated commit: byte-for-byte against the git blob it came from, or re-derived by re-running the staging script. It needs the private repository's history, which the reader does not have. It ships because the staging rule and its checker ship together or neither is auditable; nothing in this paper rests on it.

---

6 What is deliberately absent

This is an open core, not the whole product; the commercial build is distributed separately. The absences below are not omissions; each is a decision, and the code is built so that its absence is *safe* rather than merely quiet. Each is stated below as the design decision it is.

6.1 The anchor store resolves nothing

When a protected span is anchorized, this build mints an anchor identifier and then cannot resolve it: `NonResolvingAnchorStore.get()` returns `None` for every identifier, including ones it minted itself. The encrypted, tenant-partitioned, retention-bounded, retrieval-audited store is the commercial half.

The consequence is that this build **refuses more than the commercial one**, and that is measured rather than claimed twice over: §4.4's 14 refusals on `long_context_v1` are reproducible with the shipped fixtures and the reference bundle, and on the live PII route the commercial build completes 30 of 30 rows while this one refuses 4 (a measurement on production traffic, which no shipped fixture contains). The latter is stated because withholding it would be worse, and flagged as what it is: our claim about a build the reader does not have, not a number the repository can reproduce. The mechanism, by contrast, is checkable in the open

code: the verifier's anchor-resolvability check calls `anchor_store.get()`, and this store's answer is always `None`, so the open half treats every dropped span as unrecoverable and refuses.

6.2 The other absences

- **No economics gate.** The ex-ante cost model that decides whether compressing is worth it (spend caps, credit balance, budget policy) is not here. Its data contract is present (`EligibilityOutcome`, `CompressionEconomics`, `EligibilityReason`), because the fixture schema references it, but nothing computes it.
- **No paid-provider adapters**, no tenancy or audit layer, no recovery server, and no internal operations tooling, including the per-run telemetry emitter and the HTML console of §2.7.
- **No trained artifacts or training corpora.** The reference bundle is fetched and verified separately (§5.2) and cannot be retrained from this repository.
- **Nothing here authenticates anyone**, because there is nothing here to authenticate to. This is a library and a benchmark harness.

6.3 How the published surface is decided

The seam is drawn by an allowlist over paths, computed rather than maintained: the import closure of the documented entry points (deferred, in-function imports included), plus what a contributor needs to build and test, minus categorical exclusions that override both. `scripts/open_surface.py` is that rule, and it ships with the exclusion list, which names every withheld module and why.

The alternative was measured before being rejected. A denylist publishes by default: the denylist version of this repository would have shipped 88 modules, including a live OpenAI adapter and a module of spend caps, none of which anyone had decided to publish. `scripts/verify_open_core.py` (§5.5) proves the allowlist holds (every closed module unimportable) before it will print a single number.

---

7 Licence and provenance

7.1 Licence

The open core (everything in the public repository) is released under **Apache-2.0**; see `LICENSE` and `NOTICE`. The commercial build, with the resolving anchor store and the rest of §6's absences filled in, is distributed separately and is not covered by this licence. Contributions are accepted under the DCO (`CONTRIBUTING.md`). Security reports do not go in the issue tracker; see `SECURITY.md`. This paper is published with the repository and the packaged release, so that every claim in it can be checked against code from the moment it is readable.

7.2 Fixture provenance

Every fixture in the `synthetic` and `long_context_v1` corpora is machine-generated by the scripts named in its own provenance block. They contain no third-party text and no real personal data; identifiers that look like personal data (social-security numbers, card numbers, addresses at `example.com`) are synthetic values written to exercise the protected-span detector. Some of these fixtures carry legacy provenance labels (`source: public_sample`, `license: MIT`, `license: proprietary_laconiq_mock`, `adjudicated_by: nextgen_router.adjudicator`) that predate the repository and describe the *shape* the generator was imitating rather than an upstream source: there is no upstream and no third-party attribution is owed. The labels are kept as they are because rewriting them would change the corpus digests the published baselines pin. `nextgen_router` in that last label is this project's own former name, before it was renamed to TekMyra; it names no component in the repository.

The four `public_*_v1` corpora are different by design: they contain text from the Congressional Record and six permissively licensed open-source projects. Of the 138 fixtures, 67 are US-Government public-domain material and 71 are attribution-bearing, across exactly six projects: `rich`, `express`, `nlohmann` and `pytest` under MIT; `httpx` and `golang` under BSD-3-Clause. Every fixture names its exact source and licence in its provenance block; `NOTICE` carries two sections (the public-domain material and the full per-project attribution with licence text), and `tests/test_public_corpus_notice.py` verifies that coverage on every clean checkout (§5.3).

7.3 On AI assistance

Substantial portions of this codebase were written with AI assistance, directed and gated by a human maintainer. The private development history carries `Co-Authored-By` trailers naming the assistants involved. Nothing published here rests on that: every figure in this paper names its corpus, its basis, and whether the reader can re-run it: the standard we would want applied to any codebase regardless of what typed it.

7.4 References

1. [1] H. Jiang et al. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. EMNLP 2023; arXiv:2310.05736.
2. [2] Z. Pan et al. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. Findings of ACL 2024; arXiv:2403.12968.
3. [3] Model card: `microsoft/llmlingua-2-xlm-roberta-large-meetingbank`, Hugging Face; the pinned checkpoint measured in §4.6.
