

LENS-X: A Declarative Specification Language for Position-Locked Generation

Language specification v0.1, with runtime semantics and measured guarantee levels

Thomas Garren

SOPHIA XT LLC

19 September 2026 · Language v0.1

sophiaxt.com

CC BY 4.0. Code referenced herein is Apache 2.0.

Abstract

LENS-X is a declarative language for constraining the output of a discrete-sequence diffusion model. A `.lensx` document names a base model, optionally an adapter and a retrieval corpus, and then declares which output positions are fixed and what occupies them. A runtime resolves the document into a forced-anchor decode: locked positions are written before the first denoising step and excluded from the unmasking loop, so the model never has the opportunity to overwrite them. The result is a position-level guarantee on identifier-grade tokens, the pieces of an answer that must be exact, while the model remains free over the text around them. This document specifies the document model, the lock semantics, the resolution order a conforming runtime follows, and the two guarantee levels the runtime is required to report. It states what has been measured, which is that preservation is exactly 1.0 on a masked-diffusion backend and below 1.0 on a hosted autoregressive backend, and it states what has not been measured, which is everything about the quality of the text written around a lock.

1. Status of this document

This is the v0.1 language specification. The reference runtime is published separately as an Apache 2.0 Python package, with its source at github.com/Choro Zion/lens-xt. The language is LENS-X; the runtime package is named `lens-xt`. A document written against this specification carries the extension `.lensx`.

The v0.1 runtime is a pre-release. Names in the document model are settled enough to write against, but a v0.2 MAY rename optional keys, and the sections below marked reserved are declared here so that implementations agree on what not to use, not because they work yet.

This document replaces the version dated 2026-05-09. That version named a model checkpoint that was never released, carried delivery dates that the shipped runtime has already overtaken, and spent four pages on market positioning that

had no business in a specification. All of it is gone. Section 10 reports what is actually measured.

2. The problem

Some tokens in a generated answer must be exact. A drug name, a dosage string, a case citation, a part number, a column name, an API symbol: if the model paraphrases one of these, the answer is not slightly worse, it is wrong in a way that reads as authoritative. The usual defences are weak in a specific way. Instructions in a prompt are advisory. Schema-constrained decoding constrains the shape of the output rather than its content. Grammar-constrained decoding restricts the token set at each step but cannot say that position 50 holds this exact token. Post-generation validation catches the failure after the tokens are written and leaves the caller to retry.

A masked-diffusion model offers something the autoregressive stack does not. Generation starts from a fully masked sequence of known length and unmask positions over a fixed number of denoising steps. Positions are not produced in order, so a position can simply be filled in before the first step and marked as never eligible for unmasking. The model conditions on it at every step and cannot touch it. The guarantee is structural rather than statistical.

Reaching that guarantee means editing the unmasking loop, which is the wrong place for application code to live. LENS-X separates what is constrained from how the constraint is enforced. An author writes the constraint; a runtime resolves it against whichever backend is selected and reports honestly how strong the resulting guarantee is.

3. Document model

A LENS-X document is UTF-8 YAML, or the equivalent JSON for interchange. The recommended media type is `application/x-lensx+yaml`. One document specifies one generation. It has eight top-level keys, two of which are required.

```

version:    "0.1"      # required, the language version
base:      { ... }    # required, model selection
adapter:   { ... }    # optional, adapter selection
retrieval: { ... }    # optional, content fetched before generation
locks:     [ ... ]    # optional, position-locked content
generation: { ... }  # optional, decoder parameters
validation: { ... }  # optional, post-generation checks
output:    { ... }    # optional, response shaping

```

A runtime MUST reject a document whose *version* it does not implement rather than attempting a best guess. A runtime MUST reject an unknown top-level key. Silently ignoring a key the author wrote is the failure mode this rule exists to prevent: an author who misspells *locks* should get an error, not an unconstrained generation that looks fine.

```

base:
  model: "cassandra-t1"      # registry name
  precision: "bf16"          # optional runtime hint

```

The reference registry entry used throughout this document is Cassandra T1, a 1.33-billion-parameter masked-diffusion language model released under Apache 2.0: 28 transformer layers, hidden size 2,048, a 32,768-token vocabulary, grouped-query attention with 16 query heads and 4 key-value heads, at the released epoch-5 checkpoint. Any discrete-sequence diffusion model can be registered in its place. The language has no dependency on this particular one.

3.2 adapter

Selects a low-rank adapter to load over the base model. The default projection set is value and output only. That default comes from a measured result: in anchor-token-masked fine-tuning, a value-and-output-only adapter outperformed a full adapter at half the trainable parameters, and a query-and-key-only adapter fell below the random-mask control. The scope

3.1 base

Names the model the document decodes against, by registry name rather than by path, so a document is portable across machines that have registered the same model under the same name.

of that result is stated in the report it comes from and is not restated as a general claim here.

```

adapter:
  source: "registry/cardiology-v3.lora"
  rank: 16
  apply_to: ["v_proj", "o_proj"] # default
  blend_weight: 1.0

```

An adapter list with per-entry blend weights is reserved. Composition of multiple adapters is not part of v0.1 conformance and no claim is made about which blend weights remain stable.

3.3 retrieval

Names LTMi-XT, Layered Topological Memory Indexing, bundles to query before generation, and how to rank what comes back. Retrieved loci become available to locks as *retrieval[N]*, indexed by rank from zero.

```

retrieval:
  bundles:
    - "corpora/cardiology.ltm"
  query: "${user_input}"
  top_k: 3
  scoring:
    breadcrumb_match: 0.5
    decay_weight: 0.2
    semantic_similarity: 0.3
  fallback_on_empty: error # or: continue, use_literal

```

Scoring weights are passed to the retriever and their meaning belongs to it, not to this language. *fallback_on_empty* decides what happens when retrieval returns nothing: *error* halts, *continue* proceeds with the retrieval-sourced locks dropped, and *use_literal* proceeds using the lock's own *content* as a substitute. The default is *error*, because a retrieval-grounded document that silently generates ungrounded text is worse than one that fails.

4. Positions and locks

4.1 The position model

Output is a fixed-length token sequence. Its length is *generation.total_length*, or a runtime default when that is absent. Position 0 is the first generated token and positions are zero-indexed. A lock is a contiguous range of positions whose token ids are decided by the document rather than by the model.

4.2 Range forms

Form	Meaning
[start, end]	Absolute positions, end exclusive.
[start, auto]	Starts at start, length decided by the resolved content.
[auto, end]	Right-aligned so the content finishes at end.
at(position)	A single token at position.
head(N)	The first N positions.
tail(N)	The last N positions.

A range resolved with *auto* depends on tokenizer output, so its concrete extent is known only after stage 4 of resolution. A runtime MUST report the concrete extent of every lock in its result, because an author debugging an overlap needs the resolved numbers, not the declared ones.

```
locks:
  - type: locus
    range: [0, auto]
    source: retrieval[0]

  - type: literal
    range: [88, auto]
    content: "Always consult a healthcare provider."

  - type: locus
    range: [50, 65]
    source: locus("medical:dosage:nitroglycerin:standard")
```

4.4 Disjointness

Locks MUST be disjoint. Two locks whose resolved ranges overlap are an error, raised at resolution time. A runtime MUST NOT merge overlapping locks, order them by precedence, or let a later lock win. There is no correct silent answer to the question of which of two conflicting requirements the author meant.

4.5 Lock budget

Total locked positions MUST NOT exceed *total_length* minus 8. A document that locks the entire output is not generating anything, and the runtime says so rather than returning a concatenation of its own inputs as if a model had written it.

4.6 Tokenization invariance

Lock content is tokenized with the base model's tokenizer. The guarantee is over token ids: the ids at locked positions in

```
generation:
  total_length: 192
  unmask_steps: 12
  temperature: 0.8
  top_p: 0.9
  rep_penalty: 1.3
  beta: 0.5           # quantile offset for unmask scheduling
  seed: 0             # optional; fixes sampling
  noise_schedule: "pde_cosine"
```

These are decoder parameters and a runtime passes them through. A runtime MUST reject a document whose declared

4.3 Lock sources

- **literal**: text given in the document, tokenized and locked as written.
- **locus**: an LTMi-XT locus addressed by its breadcrumb path. The locus statement is the locked content.
- **retrieval[N]**: the locus at rank N from the retrieval section.
- **lensx_compose**: the output of another LENS-X document, locked into this one. Reserved; see section 6.

the output equal the ids produced by tokenizing the lock source. It is not a guarantee over the detokenized string. With a byte-pair tokenizer, the characters around a lock can in principle merge differently than they did when the lock content was tokenized alone, so the exact substring may not appear verbatim in the decoded text even though every locked id is correct. For most vocabularies this distinction never shows up. Documents that cannot tolerate it set *decode_strict: true*, which adds a re-tokenization check of the decoded output against the lock content and fails the run when they disagree.

5. Generation, validation and output

5.1 generation

locks do not fit inside *total_length* before it loads any weights, since that error is cheap to find and expensive to find late.

5.2 validation

Validation runs on the decoded output. Two validators are required for v0.1 conformance: *must_contain* and

must_not_contain, each over keyword lists or regular expressions. The rest are optional profiles.

```
validation:
  must_contain:
    - keywords: ["dosage", "consult"]
    - keywords: from_locus_required(retrieval[0])
  must_not_contain:
    - keywords: ["always safe", "no side effects"]
  must_match_schema: "schemas/medical-response.json" # optional
  must_be_valid_json: true # optional
  custom: # optional
    - plugin: "validators.compliance"
      function: "check_phi"
      severity: error
  on_failure: regenerate_once_then_error
```

A runtime that does not implement an optional validator MUST fail the document with an unsupported-validator error. It MUST NOT skip the validator and report success, which would hand the caller a result that passed checks nobody ran.

on_failure takes one of: *error*, *regenerate_once_then_error*, *regenerate_until_valid* with a *max_attempts* bound, *warn*, or *silent*.

5.3 output

```
output:
  format: text # or: json, markdown, raw_tokens
  include_provenance: true
  include_metrics:
    - anchor_preservation_rate
    - corpus_overlap
    - english_ratio
```

anchor_preservation_rate is the fraction of locked positions whose token id in the output equals the id the lock resolved to. It is the metric the guarantee in section 8.1 is stated in, and a conforming runtime MUST compute and return it whenever a document declares any lock, whether or not the document asked for it.

6. Variables and composition

String fields support ***\$*{name}** interpolation. Variables are bound by the caller at execution time, and a runtime MUST fail on an unbound reference rather than substituting an empty string.

```
# in the document
retrieval:
  query: "${user_input}"
locks:
  - type: literal
    range: [0, 20]
    content: "Re: ${ticket_id}"
```

```
# at the call site
result = LensX.run("support.lensx", variables={
  "user_input": "What is the standard dose?",
  "ticket_id": "TKT-2026-0042",
})
```

Interpolated values reach a tokenizer and, through retrieval, a query. A runtime MUST treat bound variables as data. It MUST NOT let an interpolated value introduce new locks, change a range, or alter any other structural field, whatever the value contains.

Document composition, a lock whose content is the output of another LENS-X document, is reserved. The key is *lensx_compose*, it is declared here so that implementations do not claim the name for something else, and it is not part of v0.1 conformance. Iterative refinement, in which a run feeds its own output back as context across several passes, is likewise reserved under the key *refinement*.

7. A complete document

```

version: "0.1"

base:
  model: "cassandra-t1"
  precision: "bf16"

adapter:
  source: "registry/cardiology-v3.lora"
  rank: 16
  apply_to: ["v_proj", "o_proj"]

retrieval:
  bundles:
    - "corpora/cardiology.ltm1"
    - "corpora/pharmacology.ltm1"
  query: "${user_input}"
  top_k: 3
  fallback_on_empty: error

locks:
  # the top retrieved fact opens the answer
  - type: locus
    range: [0, auto]
    source: retrieval[0]

  # a fixed closing line at a known position
  - type: literal
    range: [88, auto]
    content: "Always consult a healthcare provider."

generation:
  total_length: 192
  unmask_steps: 12
  temperature: 0.8
  top_p: 0.9

validation:
  must_contain:
    - keywords: ["consult"]
  on_failure: regenerate_once_then_error

output:
  format: text
  include_provenance: true

```

8. Runtime semantics

A conforming runtime resolves a document in this order. The order is normative: it is what makes an error surface at the

cheapest point at which it can be detected, and it is what makes two runtimes agree about which error a broken document produces.

Stage	What happens	Fails on
1. Parse	YAML or JSON to an abstract syntax tree; validate against the language schema; check the version.	Unknown version, unknown top-level key, malformed document.
2. Bind	Substitute variables into string fields.	Unbound variable reference.
3. Retrieve	Load the named bundles, run the query, bind the top-K results to retrieval[N].	Empty result under fallback_on_empty: error.
4. Resolve locks	Resolve each source to a string, tokenize with the base tokenizer, compute concrete positions, build the position-to-token-id map.	Overlapping ranges, budget exceeded, unresolvable locus.
5. Load	Load base weights, then adapters into their target projections.	Unknown registry name, tokenizer mismatch, missing adapter.
6. Decode	Initialise the sequence with locks pre-filled and marked ineligible; run the unmasking loop; locked positions are never re-evaluated.	Backend error.
7. Validate	Run validators in declaration order, then apply on_failure, which may return to stage 6.	Validator failure under on_failure: error; unsupported validator.
8. Format	Detokenize, shape per output.format, attach provenance and metrics.	Strict-decode mismatch when decode_strict is set.

Stages 1, 2 and 4 run without loading weights. A runtime SHOULD expose them as a validation mode so that a document can be checked in continuous integration on a machine with no GPU and no model files.

8.1 Guarantee levels

A runtime MUST report which of two guarantee levels applied to a run, and the level is a property of the backend, not of the document.

The distinction is the point of reporting the level at all. A caller that needs an exact identifier in the output cannot get one from a backend that only biases toward it.

Level	Backend	Mechanism	Preservation
DETERMINISTIC	Masked-diffusion, self-hosted	Locked positions are excluded from the set the sampler may write.	1.0 by construction
BEST_EFFORT	Hosted autoregressive API	Logit bias toward the locked tokens, then retry.	Can fall below 1.0

documents move between hosts.

9. Runtime interface

The language does not require a particular host API, but a runtime that offers one SHOULD keep these shapes so that

```
# Python
from lens_xt import LensX

lens = LensX("specs/medical-query.lensx")
result = lens.run(user_input="What is the standard dose?")

result.text           # decoded output
result.locked_sources # what each lock resolved to
result.metrics        # anchor_preservation_rate, and others
result.guarantee_level # DETERMINISTIC or BEST_EFFORT
result.validation_passed # bool
```

```
# Command line
lensx validate specs/*.lensx      # stages 1, 2 and 4 only
lensx explain specs/query.lensx   # resolved locks and positions
lensx run specs/query.lensx --var user_input="..." \
                                --show-provenance
```

```
# HTTP
POST /api/v1/lensx/run
Content-Type: application/json
```

```
{
  "spec":      "<document as YAML or JSON>",
  "variables": {"user_input": "..."},

```

```
  "options":  {"include_provenance": true}
}
```

A response carries the decoded text, the resolved locks with their concrete positions, the metrics, the guarantee level, the validation outcome and a trace identifier.

10. Implementation status and what has been measured

The reference runtime is published as a pre-release package under Apache 2.0. Parser, schema validator, lock resolver for literal, locus and retrieval sources, the masked-diffusion backend, a hosted-API backend, a Python interface, a command-line tool and an HTTP server are implemented. Adapter composition, document composition and iterative refinement are not.

The suite collects 259 tests. On a clean clone at main with no optional extras it reports 253 passed, 5 skipped and 1 deselected, with no failures: the skips are tests that need an optional tensor library and the deselection is a live integration test that needs weights the repository does not ship. With that optional extra installed, at commit 01e5eed on 16 September 2026, ten further tests run and it reports 256 passed, 2 skipped and 1 failed. The single failure is a test asserting that an internal variant registry holds exactly seven entries when five more have since been added beside them. It is a stale assertion in a test rather than a fault in the runtime, and it is recorded here because a reader who installs the extra will meet it.

Two assertions carry the guarantee in section 8.1. On the masked-diffusion path the suite asserts that *anchor_preservation_rate* is exactly 1.0. On the hosted-API path it asserts that the rate is below 1.0. The second assertion is the more useful of the two: it is the runtime committing, in its own test suite, to the claim that the weaker backend is weaker.

11. What this specification does not establish

Nothing about the quality of generated text. Preservation says the locked span comes out unchanged. It says nothing about whether the text written around it is accurate, coherent or useful. On a 1.33-billion-parameter base model at the released checkpoint it frequently is not. A locked citation inside a bad paragraph is still a bad paragraph.

No comparison against other constrained-generation approaches. No head-to-head measurement has been run against schema-constrained decoding, grammar-constrained decoding or any other method, so this document makes no claim that LENS-X is better than any of them at anything. The property it offers, a position-level guarantee over token ids, is a different property from the ones those methods offer, and a difference is not a result.

No end-to-end task measurement. The verticals discussed as motivation in section 2, clinical, legal, technical and schema-bound generation, are use cases the mechanism is designed for. No accuracy measurement on any of them has been published, and no adapter for any of them is claimed here as trained, evaluated or fit for use.

No measurement of the cost of locking. Fixing positions removes degrees of freedom from the decoder. Whether a heavily locked generation is worse around the locks than a lightly locked one, and where that trade-off turns, has not been measured.

12. Limitations

- **Fixed output length.** The position model requires a length decided before generation. Absolute ranges are meaningful only against that length, so a document written for one *total_length* does not transfer to another.
- **Tokenizer coupling.** A document is bound to the tokenizer of the model it names. Moving it to a model with a different vocabulary invalidates every absolute range. Runtimes check that the loaded tokenizer matches the one the document was written against and refuse the run otherwise.
- **Absolute positions are brittle to author.** Writing a lock at position 88 means knowing what occupies positions 0 to 87. The auto forms exist because of this, and a document that uses absolute ranges throughout is hard to maintain.
- **The guarantee does not survive the backend change.** The same document run against a hosted autoregressive API drops to best effort. Nothing in the language can raise a backend's capability; the level is reported so the caller can decide, not so the caller can ignore it.
- **The retrieval joint inherits the retriever's limits.** A lock sourced from retrieval is exactly as correct as the retrieval was. LENS-X guarantees that the retrieved statement appears unaltered, not that it was the right statement.
- **Pre-release.** The runtime is a pre-release with one known failing test under an optional install, and optional keys may be renamed before v0.2.

13. Open questions

- **Tokenizer binding.** Should a document be bindable to several tokenizers at run time, or locked to one at write time? v0.1 locks it at write time and the runtime checks the match. Relative ranges anchored to content rather than to integers would relax this, at the cost of a harder resolution stage.
- **Lock conflicts.** v0.1 makes overlap an error. A documented merge policy would let generated documents compose without a pre-pass, and would also let a generator emit a contradiction that the runtime resolves quietly. The error is the conservative choice and is where v0.1 stands.
- **Determinism across runs.** The intent is that a fixed seed, a fixed document and fixed variables produce identical output. Floating-point kernels on accelerators are not always bit-reproducible, so this has not been demonstrated across hardware and is stated here as intent rather than as a guarantee.
- **Adapter composition stability.** At what blend weights, if any, multiple adapters compose without degrading is an empirical question nobody has answered. It is the reason composition is reserved rather than specified.

14. License and citation

This specification is released under Creative Commons Attribution 4.0. The reference runtime is released under Apache 2.0.

```
@techreport{garren2026lensx,  
  author      = {Garren, Thomas},  
  title       = {LENS-X: A Declarative Specification Language for  
                Position-Locked Generation},  
  institution = {SOPHIA XT LLC},  
  year        = {2026},  
  url         = {https://sophiaxt.com/lens-x-spec}  
}
```

References

- [1] LENS-X reference runtime. Apache 2.0. github.com/Chorozion/lens-xt
- [2] LTMi-XT: A Retrieval and Training Record Format with Hash-Derived Lattice Addressing. SOPHIA XT LLC, 2026. <https://sophiaxt.com/lmi-xt-spec.pdf>
- [3] Garren, T. (2026). Anchor-Token Masking Produces an Out-of-Distribution Generalization Advantage in Retrieval-Grounded Masked-Diffusion Language Models. SOPHIA XT LLC. doi:10.5281/zenodo.20496783. The source of the value-and-output projection default in section 3.2.
- [4] Cassandra T1. Masked-diffusion language model, 1.33B parameters, Apache 2.0. <https://sophiaxt.com/models/cassandra>