

Qorx Local Context Resolution: a handle-resolved runtime model for local AI context

Marvin Sarreal Villanueva

Metro Manila, Philippines

Version 0.2 preprint manuscript

May 1, 2026

DOI: 10.5281/zenodo.19953308

License: CC BY-ND 4.0

Software version: Qorx 1.0.3

Abstract

Large language model workflows often resend local context: source files, logs, notes, retrieved chunks, and previous tool output. Qorx defines a smaller runtime boundary. A workflow carries a source file, bytecode file, handle, or evidence pack, then resolves the needed evidence locally under a declared mode and budget. This paper defines Qorx Local Context Resolution (Q-LCR), describes the carrier/resolver model, and gives an evaluation plan. This manuscript is a technical preprint. It is not peer reviewed.

1. Definition

Qorx Local Context Resolution, or Q-LCR, is the runtime process of resolving a small carrier into local evidence under a declared task mode and evidence budget.

A carrier can be .qorx source, .qorxb bytecode, a qorx:// handle, or a compact evidence pack. The carrier is not the evidence itself. The resolver reads local state and returns a proof page, or refuses when it cannot support the request.

This distinction matters because a handle alone does not make hidden local data known to a model. The resolver path is part of the system.

2. Statement of need

Developer-agent workflows often need the same local facts more than once. A repository answer may depend on source files, generated indexes, prior tool output, a cache entry, or a signed receipt. Sending all of that state into a model prompt is simple, but it is hard to audit and it can waste context budget.

Qorx is for the narrower case where the evidence already exists locally and a workflow needs a small, inspectable way to ask for it. Q-LCR treats the handle or bytecode as a carrier, not as the evidence itself.

3. Motivation

RAG showed that retrieval can ground generation in external documents rather than relying only on model parameters. ReAct-style agents showed that language models can interleave reasoning with tool actions. MemGPT framed long-context management as an operating-system-like memory problem. Long-context studies such as "Lost in the Middle" showed that adding more tokens is not the same as making all evidence equally usable.

Qorx sits in this engineering space, but it makes a narrower claim. It does not train a model, replace RAG, or promise universal compression. It defines a local runtime boundary for addressed evidence.

4. Runtime model

Q-LCR has five parts:

1. Carrier: the small object moved through the workflow.
2. Resolver: the local runtime that interprets the carrier.
3. Local state: indexes, cache entries, receipts, provenance, and source text.
4. Budget: the maximum visible evidence allowed for the task.
5. Proof page: the returned evidence and citation surface.

The simplest .qorx program is:

```
QORX 1
@mode strict-answer
@ask what proves this repository contains the Qorx runtime?
@budget 700
```

The runtime can compile the source to bytecode:

```
qorx qorx-compile goal.qorx --out goal.qorxb
qorx goal.qorxb
```

5. Measured claims

Qorx should be judged on measured behavior, not vocabulary.

The current measurable claims are:

- parser and bytecode round-trip correctness;
- resolver support rate on evidence-seeking tasks;
- retrieval correctness against known local files;
- visible token reduction under a declared local estimator;

- latency of indexing and resolution;
- provenance verification for receipts and proof pages.

Redshift is local accounting:

$$\text{redshift} = \text{local_estimated_tokens} / \text{visible_estimated_tokens}$$

B2C accounting compares a baseline payload with the carrier or proof page:

$$\text{omitted_tokens} = \text{baseline_local_tokens} - \text{visible_carrier_tokens}$$

Provider invoice savings require provider billing evidence. Qorx local accounting is not enough.

6. Evaluation plan

The first serious evaluation should use at least three repositories:

- a small Rust CLI;
- a medium web application;
- a mixed documentation/code repository.

For each repository, run paired tasks:

- baseline: paste or retrieve the usual large context;
- Qorx: use `.qorx`, `.qorxb`, or `qorx://` carriers and resolve locally.

Record:

- task success;
- support rate;
- exact evidence citations;
- visible input tokens;
- latency;
- resolver errors;
- provider-side cost only when routed billing evidence exists.

The evaluation should publish the task list, source revision identifiers, Qorx commands, result files, and failure cases.

7. Related work

Retrieval-Augmented Generation introduced a retrieval-backed generation model for knowledge-intensive NLP tasks. Qorx differs by keeping retrieval and evidence resolution in a local runtime instead of defining a model architecture.

ReAct showed the value of tool use as part of language model workflows. Qorx treats the resolver as a tool/runtime boundary with explicit carriers.

MemGPT used an operating-system memory analogy for long-context agents. Qorx

uses a simpler runtime boundary: handle, resolve, pack, cite, or refuse.

"Lost in the Middle" showed that long context can still be hard for models to use. Qorx does not solve long-context reasoning by itself. It tries to reduce the amount of irrelevant context placed in the visible prompt.

8. Limits

Q-LCR does not mean that a remote model understands a local repository from a short handle. It means the local runtime can resolve that handle when asked.

Qorx also does not claim first use of the broad phrase "context resolution". This paper defines Qorx Local Context Resolution as a specific AI/developer tooling runtime model.

Q-LCR is not a standard by itself. A standard would need independent implementations, outside review, and a separate standards process.

9. Availability

The reference implementation is Qorx 1.0.3. The source code is available at <https://github.com/bbrainfuckk/qorx> under AGPL-3.0-only.

The preprint record is available through Zenodo DOI 10.5281/zenodo.19953308.

The software citation record is separate and available through Zenodo DOI 10.5281/zenodo.19875352.

10. References

- Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." arXiv:2005.11401, 2020. <https://arxiv.org/abs/2005.11401>
- Shunyu Yao et al. "ReAct: Synergizing Reasoning and Acting in Language Models." arXiv:2210.03629, 2022. <https://arxiv.org/abs/2210.03629>
- Charles Packer et al. "MemGPT: Towards LLMs as Operating Systems." arXiv:2310.08560, 2023. <https://arxiv.org/abs/2310.08560>
- Nelson F. Liu et al. "Lost in the Middle: How Language Models Use Long Contexts." arXiv:2307.03172, 2023. <https://arxiv.org/abs/2307.03172>