TORQON

Token-Optimized Retrieval & Query Orchestration Network

Infrastructure Report & Technical Overview

Executive Summary

Torqon is a context optimization infrastructure layer designed to improve how Large Language Models process memory, conversation history, and long-running workflows.

Modern AI applications typically rely on raw context injection, where entire conversation histories are repeatedly transmitted to models regardless of relevance. This creates substantial inefficiencies in:

- token consumption
- latency
- contextual focus
- long-session consistency
- memory continuity

Torqon addresses these limitations by introducing a structured context management pipeline that:

- selectively retrieves relevant memory
- compresses historical context
- dynamically controls token budgets
- prioritizes high-signal information
- assembles optimized prompts before inference

Instead of treating context as static text, Torqon transforms context into a structured, retrievable, ranked, and budget-controlled resource.

The result is:

- reduced token overhead
- improved contextual consistency
- cleaner reasoning chains
- scalable long-session interaction
- lower operational inference cost

Torqon is currently being developed as the core infrastructure layer powering Meridian.

---

1. Industry Problem

Current State of AI Applications

Most AI products today operate using a simple wrapper architecture:

```
User Input
   ↓
Frontend
   ↓
LLM API
   ↓
Response
```

These systems typically:

- resend raw conversation history
- rely entirely on the model to infer relevance
- repeatedly transmit redundant context
- scale token usage linearly with conversation growth
- degrade in quality during long sessions

As interactions become more complex, these limitations compound rapidly.

Core Infrastructure Challenges

1. Token Inefficiency

Long-running conversations produce exponential token growth.

Example:

Conversation Length	Approximate Token Usage
10 messages	3K–5K tokens
50 messages	15K–25K tokens
100+ messages	40K+ tokens

This directly increases:

- inference cost
- latency

- prompt noise
- retrieval ambiguity

---

2. Weak Long-Term Memory

LLMs do not possess persistent memory by default.

Without external context infrastructure:

- previous decisions are forgotten
- project continuity degrades
- architectural consistency breaks
- models hallucinate outdated information

---

3. Context Pollution

Raw conversation history contains:

- irrelevant discussion
- duplicated information
- casual interactions
- obsolete decisions
- low-signal content

Models must internally infer:

> “What matters?”

This is computationally inefficient.

---

4. Scaling Constraints

Increasing context window size alone does not solve the problem.

Larger windows still:

- include irrelevant information
- increase operational cost
- reduce contextual precision
- create weaker attention allocation

---

2. Torqon Thesis

Torqon is built on the principle that:

> AI output quality is heavily dependent on context quality.

Rather than increasing raw context volume, Torqon optimizes:

- relevance
- structure
- continuity
- prioritization
- compression
- retrieval precision

Torqon acts as a middleware intelligence layer positioned between the user and the model.

---

3. High-Level Architecture

```
User Input
   ↓
Intent Classification
   ↓
Memory Retrieval (RAG)
   ↓
Conversation Summarization
   ↓
Recent Context Selection
   ↓
Token Budget Enforcement
   ↓
Prompt Assembly
   ↓
LLM Inference
   ↓
Response
   ↓
Memory Extraction & Storage
```

---

4. Core Infrastructure Components

4.1 Intent Classification Layer

Objective

Determine whether a query actually requires contextual memory.

Without classification:

- memory is injected unnecessarily
- token overhead increases
- irrelevant context pollutes prompts

Classification Categories

Project / Contextual

Requires prior discussion or workspace continuity.

Examples:

```
Continue this architecture.
Improve the memory layer.
What stack are we using?
```

General / Standalone

Can be answered independently.

Examples:

```
Explain recursion.
What is a vector database?
```

Multi-Layer Classification System

Layer 1 — Fast Heuristic Detection

Uses:

- contextual references
- conversation depth
- pronoun analysis
- structural cues

Low latency and inexpensive.

Layer 2 — Lightweight LLM Classification

If ambiguity exists:

- a lightweight model performs fallback classification
- output remains deterministic

Result:

```
project
```

or

```
general
```

This minimizes unnecessary RAG execution.

4.2 Retrieval-Augmented Generation (RAG)

Objective

Retrieve only relevant memory instead of transmitting entire conversation history.

Traditional AI Memory Approach

```
Attach everything.
```

Torqon Retrieval Strategy

```
Retrieve only what matters.
```

Example Memory Objects

```
[Stack] Supabase + OpenRouter
[Goal] Reduce token consumption
[Decision] Switched from Pinecone to pgvector
[Constraint] Maintain low latency under 4K context
```

Retrieval Pipeline

```
Query
   ↓
Embedding Generation
   ↓
Vector Similarity Search
   ↓
Scoring & Ranking
   ↓
Top Relevant Memory
```

Similarity Thresholding

Torqon rejects weak contextual matches.

Example:

Similarity Score	Action
0.92	Retrieve
0.84	Retrieve
0.63	Reject

This prevents low-quality context injection.

4.3 Summary Compression Engine

Objective

Compress long conversations into high-density summaries.

Example

Instead of transmitting:

```
40–50 raw messages
```

Torqon generates:

```
Goal: Build AI coordination infrastructure
Stack: Supabase + OpenRouter
Focus: Context optimization and memory retrieval
Recent Decision: Added deterministic token budgeting
```

Benefits

- preserves continuity
- significantly reduces token usage
- improves contextual precision
- maintains long-session state

4.4 Token Optimization Infrastructure

Objective

Maintain strict control over context window utilization.

Problem

Every model has a finite context limit.

Overflow creates:

- failed requests
- truncation
- degraded reasoning
- rising cost

Torqon Strategy

Torqon enforces:

```
Safe Context Budget: 3800 Tokens
```

Context Allocation

Context Section	Allocation
Summary	40%
Memory	30%
Recent Conversation	30%

Trimming Hierarchy

If overflow occurs:

1. Trim oldest recent messages

2. Trim lowest-ranked memory chunks
3. Preserve summary whenever possible

This ensures the highest-value context survives.

4.5 Deterministic Prompt Assembly

Objective

Construct structured prompts instead of raw conversational payloads.

Final Prompt Structure

```
System:
...

Summary:
...

Relevant Memory:
...

Recent Conversation:
...

User:
...
```

Infrastructure Benefit

Structured prompts:

- improve reasoning consistency
- reduce ambiguity
- increase focus
- reduce contextual drift

4.6 Memory Extraction Infrastructure

Objective

Convert conversations into reusable long-term memory.

Example

Conversation:

```
We replaced Pinecone with pgvector.
```

Stored memory:

```
[Decision] Using pgvector instead of Pinecone
```

Deduplication Layer

Torqon prevents redundant memory accumulation.

Example:

```
Using Supabase
I use Supabase
Stack includes Supabase
```

These are consolidated semantically.

---

5. Before vs After Torqon

Standard Wrapper Architecture

```
Send full conversation history.
```

Consequences:

- token inflation
- context noise
- weaker focus
- degraded continuity
- higher inference cost

Torqon Architecture

```
Send:
- compressed summary
- relevant memory
- recent high-signal interaction
- optimized prompt structure
```

Results:

- reduced token overhead
- improved reasoning quality
- stronger continuity
- lower latency
- cleaner contextual focus

---

6. Benchmark Example

Baseline System

```
Conversation Length: 50 messages
Token Usage: 7200+
```

Observed Behavior:

- repeated information
- generic responses
- loss of architectural consistency

Torqon System

```
Summary: 1200 tokens
Memory: 800 tokens
Recent Context: 1000 tokens
Total: ~3000 tokens
```

Observed Behavior:

- stronger contextual awareness
- reduced redundancy
- improved continuity
- lower operational cost

---

7. Scoring Infrastructure

Torqon ranks contextual memory using deterministic scoring.

```
final_score = (similarity × 0.7) + (recency × 0.3)
```

Purpose

Not all memory should be treated equally.

Recent implementation changes may outweigh older architectural assumptions.

This scoring system ensures:

- relevant information receives priority
- outdated memory decays naturally
- retrieval remains contextually aligned

---

8. Tokenization Infrastructure

Torqon uses:

```
js-tiktoken
```

with:

```
cl100k_base
```

for GPT-style token accounting.

Purpose

Enable:

- accurate prompt budgeting
- overflow prevention
- deterministic trimming
- efficient context allocation

---

9. Observability & Evaluation

Torqon includes full observability infrastructure.

Tracked metrics include:

- token consumption
- memory usage
- similarity scores
- classification decisions
- token reduction percentage
- contextual retrieval quality

Example Log

```
{
  "total_tokens": 3650,
  "summary_tokens": 1400,
  "memory_tokens": 850,
  "recent_tokens": 1100,
  "memory_used": true,
```

```
  "classification": "project"
}
```

---

10. Evaluation Framework

Torqon supports controlled benchmarking between:

```
Baseline vs Torqon
```

Measured outcomes:

- token reduction
- contextual consistency
- retrieval precision
- response quality
- long-session continuity

---

11. Strategic Positioning

Torqon is not designed as a conventional AI wrapper.

Traditional wrappers primarily:

```
route prompts → return outputs
```

Torqon instead manages:

- context prioritization
- memory retrieval
- prompt structure
- token economics
- long-session continuity

It functions as:

> a context optimization infrastructure layer for LLM systems.

---

12. Long-Term Vision

Torqon is designed to evolve toward:

- persistent AI workspaces
- adaptive memory systems
- multi-model coordination
- intelligent context routing
- scalable AI operating infrastructure

Future roadmap areas include:

- autonomous context planning
- dynamic prompt compression
- cross-session intelligence
- multi-agent orchestration
- persistent organizational memory

---

13. Conclusion

Torqon is a context optimization infrastructure layer engineered to improve how modern AI systems manage memory, continuity, and token efficiency.

Rather than relying on raw context injection, Torqon transforms conversational state into:

- structured
- retrievable
- ranked
- compressed
- budget-controlled context

By combining:

- intent classification
- retrieval systems
- deterministic scoring
- token budgeting
- structured prompt assembly

Torqon enables Meridian to move beyond traditional AI wrapper architectures and toward scalable AI infrastructure.

Its objective is not simply to generate responses.

Its objective is:

to ensure models receive the highest possible quality context at the lowest possible operational cost.