Fractal Workspace

Visual-first, node-based idea workspace for human + LLM collaboration

Tagline: Think in branches. Keep the truth. Scale memory beyond the context window.

Executive Summary (Elevator Pitch)

Fractal Workspace introduces a new interaction primitive for LLMs: a **visual, node-based workspace** where each node is a bounded chat instance with its own frozen context, a generated name, and a structured contribution to a lineage knowledge graph.

This approach addresses three persistent problems in LLM-assisted thinking:

1. Context pollution when exploring multiple ideas in parallel
2. Short-lived, brittle context memory and hallucinations as conversations grow
3. Practical performance limits on local or smaller LLMs (memory and speed)

By combining **branching**, **knowledge-graph summaries**, and **retrieval-augmented generation (RAG)**, Fractal Workspace enables reliable, scalable, and explainable ideation workflows — even on constrained local hardware.

The Problem (Why This Matters)

As large language models have become more capable, their limitations during *extended reasoning and ideation* have become increasingly apparent.

Humans naturally think in **branches**. When exploring ideas, we maintain parallel lines of thought and later merge or discard them. Traditional chat interfaces are strictly linear, forcing unrelated threads into a single context and degrading output quality over time.

LLMs also suffer from **context degradation**. As the context window fills, models become more prone to hallucination and factual drift. This is a well-documented phenomenon in neural text generation and abstractive summarization research.

Finally, **local inference constraints** remain a practical concern. Quantization techniques such as 4-bit and QLoRA make local deployment feasible, but memory bandwidth and latency still demand careful control over context size.

In short: current chat interfaces do not align with how humans think, nor with how LLMs remain reliable.

Reframing the Problem: Fractal Workspace

Fractal Workspace replaces the single linear chat with a **directed acyclic graph (DAG) of semantic states**.

Each **node** represents a bounded chat instance with:

- A frozen conversation log
- A generated short name
- A derived summary
- A local knowledge-graph delta (entities and relations introduced)
- A lifecycle state (active, frozen, deleted)

Edges represent semantic lineage rather than raw message flow.

For any LLM call, the effective context is constructed as:

System prompt

- Lineage summary
- Lineage knowledge graph
- Node summary
- Node graph delta
- Last-N node-local messages

Merges are explicit operations. A merge produces a new summary and graph delta, which are appended as derived artifacts to the target node. Forgetting becomes intentional, traceable, and auditable.

Implemented Features

- Visual infinite-canvas UI with first-class nodes
- Branching, drag-to-merge, copy, delete, freeze operations
- Event-sourced backend (append-only action log as source of truth)
- Node-level summarization with provenance tracking

- Knowledge-graph construction (entity–relation deltas per node)
- Retrieval-augmented generation (RAG) using embeddings
- Auto-splitting agent for overloaded nodes (experimental)
- Local-first LLM orchestration using Ollama
- Quantization-aware model selection per role
- Traceable deletion with optional downstream recompute

Example 1: Branching Prevents Context Pollution

Scenario: Designing a neonatal jaundice research pipeline.

- Center node: “Neonatal Jaundice Project” (goals and constraints)
- Branch A: “Data Pipeline” (data ingestion, bilirubin trends, calibration)
- Branch B: “Model Architecture” (CNN vs XGBoost vs time-series models)

In a linear chat, low-level preprocessing details from Branch A bleed into architectural reasoning in Branch B. In Fractal Workspace, Branch B inherits only the **explicit summary** of the center node. It never sees data-pipeline chatter unless explicitly merged.

When the user decides to merge, the system generates a concise merged summary and graph delta — the only information promoted to global context.

Example 2: Knowledge Graph + Summary Compression

After six sessions exploring multiple models and preprocessing strategies, a linear chat would require thousands of tokens of history.

In Fractal Workspace:

- Each node emits graph deltas such as:
    - Decision: use EfficientNet-based CNN
    - Experiment: augmentation A and B failed
- The summarizer compresses lineage into a short, decision-focused summary
- Provenance is preserved via node IDs

Future LLM calls receive only the distilled meaning plus pointers, not the full raw logs. Token usage stays low while auditability remains intact.

Retrieval-Augmented Generation (RAG)

RAG enables effectively unlimited long-term memory by storing embeddings of important artifacts and retrieving only the most relevant information at query time.

This approach:

- Reduces hallucination on knowledge-intensive tasks
- Keeps prompt sizes minimal
- Works well with local vector databases such as FAISS or Qdrant

Fractal Workspace integrates RAG as an optional augmentation layer whenever the agent detects missing context.

Why Hallucination and Forgetting Occur

- Abstractive models frequently hallucinate when compressing or inferring information
- Larger context windows mitigate but do not eliminate this issue
- Retrieval and structured memory are widely accepted mitigation strategies

These limitations are not theoretical — they are empirically documented across summarization and dialogue tasks.

Why Bigger Models Alone Are Not the Solution

Increasing parameter count and context window size is expensive and often impractical, especially for local inference.

Fractal Workspace focuses instead on **memory architecture**:

- Explicit structure
- Provenance tracking
- Controlled context construction

Quantization techniques like QLoRA make large models accessible, but disciplined context management is still essential. This system enforces that discipline by design.

Technical Architecture Overview

Clients (Browser UI)
↔ API (FastAPI)
↔ Event Store (Append-only)

Event Store → Derived Artifacts (Summaries, Graph Deltas)
Embeddings / Vector DB ↔ Retriever
LLM Layer (Chat, Summarizer, Graph Builder, Node Namer via Ollama)

Agents subscribe to events and trigger summarization, retrieval, and auto-splitting.

Demo Flow

1. Create center node and enter project description
2. Create two branches and demonstrate isolated reasoning
3. Introduce conflicting decisions and perform a merge
4. Show generated summary and knowledge-graph delta
5. Trigger retrieval for missing context
6. Demonstrate auto-split of an overloaded node
7. Inspect event log to show full traceability

Limitations and Tradeoffs

- Manual summarization in Phase-1
- Downstream recompute on deletion can be expensive
- RAG quality depends on embedding and chunking strategy
- Local inference still constrained by GPU memory and concurrency limits

---

Repository Structure (Phase-1)

backend/ — FastAPI event-sourced API
agent/ — auto-split and orchestration logic
llm-config/ — Ollama Modelfiles and role configs
docs/ — slides and documentation

How to Run

1. Start Ollama locally and load a supported model
2. Run backend using provided script
3. Open API docs and UI
4. Follow demo script

Final Note

This system aligns:

- How humans think
- How LLMs remain reliable
- How local inference constraints work in practice

Branching, structured memory, retrieval, and provenance are not optional features — they are necessary foundations for serious LLM-assisted reasoning.

Contributors:
Aryan Gupts 2nd year mechanical engg(IITR)
Suryansh Singh 2nd year mechanical engg(IITR)