CTK: Conversation Toolkit

A Unified System for Multi-Provider AI Conversation Management

Technical Report
`https://github.com/yourusername/ctk`

October 7, 2025

Abstract

We present the Conversation Toolkit (CTK), a comprehensive system for managing AI conversations from multiple providers in a unified format. As users increasingly interact with diverse LLM platforms—ChatGPT, Claude, Gemini, and coding assistants—conversations become fragmented across incompatible formats and isolated interfaces. CTK addresses this by introducing a universal tree-based conversation model that preserves structural information while enabling cross-platform operations. The system combines extensible plugin architecture, natural language query capabilities, and interactive management tools to provide a complete solution for conversation archival, organization, and analysis. We demonstrate CTK's effectiveness across multiple providers and discuss design principles for building unified interfaces to heterogeneous AI systems.

Contents

1 Introduction

1.1 The Conversation Fragmentation Problem

The rapid adoption of large language models has created an unprecedented volume of human-AI interactions. Users engage with ChatGPT for creative writing, Claude for deep reasoning, Gemini for research, and GitHub Copilot for coding—often switching between platforms multiple times per day. Each platform maintains conversations in proprietary formats, creating several critical problems:

Knowledge Silos: Valuable insights, solutions, and discussions are scattered across platforms with no unified access mechanism. A user who solved a problem in ChatGPT last month cannot easily find that solution when working in Claude today.

Platform Lock-In: Conversations cannot be moved between providers, limiting user agency and making it difficult to compare platform capabilities on real workloads.

Organizational Challenges: Different platforms offer varying organizational features. ChatGPT provides folders, Claude offers search, Gemini has minimal organization. No platform provides comprehensive tagging, archival, or cross-conversation analysis.

Research Barriers: Studying conversational AI requires access to real conversations, but researchers face significant barriers extracting and analyzing data from multiple platforms.

Fine-Tuning Data Preparation: Creating training datasets from real conversations requires normalizing diverse formats, handling branching structures, and ensuring data quality—tasks that are error-prone and platform-specific.

1.2 Core Insight: Conversations as Universal Trees

CTK's fundamental insight is that *all AI conversations can be represented as trees*, regardless of their source platform or linear/branching nature. This universality enables:

- Linear conversations as single-path trees
- ChatGPT regenerations as branching trees
- Coding assistant interactions as trees with tool calls
- Future multimodal conversations with the same structure

By standardizing on this representation, CTK decouples conversation *storage and management* from their original *source platform*, enabling operations impossible within any single platform.

1.3 Contributions

This work makes the following contributions:

1. **Universal Conversation Model**: A tree-based representation handling linear and branching conversations uniformly
2. **Cross-Platform Architecture**: Plugin system enabling extensible format support without core modifications
3. **Natural Language Interface**: LLM-powered queries using tool calling for intuitive conversation access

4. **Integrated Management**: Combined CLI, TUI, and API supporting diverse workflows

5. **Privacy-First Design**: Fully local operation with optional sanitization for sharing

1.4 Paper Structure

Section 2 discusses related work. Section 3 presents core concepts and the conversation model. Section 4 describes system architecture. Section 5 details key features and design decisions. Section 6 evaluates performance and accuracy. Section 7 discusses lessons learned and future directions. Section 8 concludes.

2 Related Work

2.1 Conversation Management

Existing systems address aspects of conversation management but lack CTK's unified approach:

Platform-Specific Tools: ChatGPT provides JSON export, Claude offers conversation download, but these are single-platform solutions requiring custom parsing and offering no management capabilities.

LLM Application Frameworks: LangChain [1] and LlamaIndex [2] focus on building applications with conversation chains but do not address archival, cross-platform compatibility, or historical conversation management.

Local LLM Systems: PrivateGPT and similar tools enable local model deployment but lack conversation archive functionality.

CTK differs by treating conversations as *first-class knowledge artifacts* worthy of long-term management, not just ephemeral application state.

2.2 Knowledge Management Systems

Personal knowledge management (PKM) systems provide relevant paradigms:

Graph-Based Notes: Obsidian and Logseq use bidirectional links to connect notes, enabling knowledge graph construction. CTK extends this to conversational knowledge where links emerge from context and topic similarity.

Research Management: Zotero and Mendeley manage research papers with metadata, tags, and search. CTK applies similar principles to conversations, treating each as a document with structured metadata.

Document Intelligence: DevonThink combines document management with AI features. CTK specializes this for the unique characteristics of conversations: temporal ordering, branching structure, and turn-taking.

2.3 Tree-Structured Data

CTK's tree model draws inspiration from established systems:

Version Control: Git's directed acyclic graphs (DAGs) track code evolution with branching and merging. CTK adapts this concept to conversation evolution, where "branches" represent regenerations or alternative responses.

Hierarchical Data: File systems, XML documents, and abstract syntax trees demonstrate the power of tree structures for organizing complex information. Conversations naturally fit this paradigm with parent-child message relationships.

Conversation Trees in RL: Reinforcement learning from human feedback (RLHF) [3] uses tree-structured conversation rollouts. CTK generalizes this structure for general-purpose conversation management.

3 Core Concepts

3.1 The Conversation Tree Model

A conversation is represented as a tree $T = (M, E)$ where:

- M is a set of messages, each with unique identifier, role (user/assistant/system), content, and timestamp
- $E \subseteq M \times M$ is a set of directed edges (m_i, m_j) indicating message m_j is a response to m_i
- One or more root messages $R \subseteq M$ have no parents
- Leaf messages $L \subseteq M$ have no children

Key Properties:

1. **Acyclic**: The tree structure prevents circular references
2. **Temporal**: Edges respect temporal ordering (parent precedes child)
3. **Multi-rooted**: Supports rare cases of multiple initial messages
4. **Content-Rich**: Messages contain text, images, audio, video, tool calls

Path Semantics: A path $p = [m_1, m_2, \ldots, m_n]$ from root to leaf represents a complete conversation flow. Trees with multiple paths represent:

- Different response attempts (ChatGPT regenerations)
- User exploring alternatives (forking conversations)
- Parallel conversation branches (multi-turn exploration)

3.2 Universal Representation

The tree model achieves universality by handling special cases:

Linear Conversations: A tree with single path $(m_1) \to (m_2) \to \cdots \to (m_n)$ representing traditional turn-taking dialogue.

Branching Conversations: Trees with multiple children per node, capturing:

- Regenerated responses with different content
- Forked conversations exploring different directions
- Assistant providing multiple alternative answers

Multi-Modal Content: Messages support heterogeneous content:

- Text with markdown, code, LaTeX

- Images (uploaded or generated)
- Tool calls and results (function calling)
- Audio/video (future modalities)

3.3 Metadata and Organization

Each conversation tree carries metadata:

- **Provenance**: Source platform, model used, creation time
- **Organization**: Tags, projects, starred/pinned/archived status
- **Analytics**: Message counts, token usage, conversation duration
- **Custom Fields**: Extensible key-value storage

This metadata enables sophisticated filtering, analysis, and organization beyond what individual platforms provide.

4 System Architecture

4.1 Architectural Principles

CTK's architecture follows these design principles:

1. **Layered Design**: Clear separation between data models, storage, business logic, and interfaces
2. **Plugin-Based Extensibility**: New formats added without modifying core
3. **Privacy First**: No network dependencies except optional LLM features
4. **Single Responsibility**: Each component has one well-defined purpose
5. **Progressive Complexity**: Simple operations are simple, complex operations possible

4.2 Four-Layer Architecture

Layer 1: Data Layer

Persistent storage using SQLite with:

- Conversations table: Core metadata and organization fields
- Messages table: Content and tree structure (parent_id references)
- Tags table: Many-to-many tagging system
- Paths table: Cached complete paths for performance

SQLite provides ACID guarantees, portability, and excellent performance up to 100K+ conversations.

Layer 2: Plugin Layer

Extensible plugin system with:

- **Importers**: Convert platform-specific formats to conversation trees
- **Exporters**: Convert trees to various output formats
- **LLM Providers**: Unified interface to multiple LLM APIs
- **Auto-Discovery**: Plugins loaded dynamically at runtime

Layer 3: Core Logic Layer
Business logic including:

- Database operations: CRUD with transaction management
- Search and filtering: Full-text search, metadata filtering
- Organization: Starring, pinning, archiving, tagging
- Analytics: Statistics, trends, usage patterns

Layer 4: Interface Layer
Multiple interfaces for different workflows:

- **CLI**: Command-line for scripting and automation
- **TUI**: Interactive terminal UI for exploration
- **REST API**: HTTP interface for web applications
- **Python API**: Library for programmatic access

4.3 Data Flow

Import Pipeline:

1. User provides file and optional format hint
2. Format detection: Try each importer's `validate()` method
3. Parsing: Selected importer converts to conversation trees
4. Normalization: Ensure all required fields present
5. Storage: Persist trees to database with metadata
6. Indexing: Update search indices and cached paths

Query Pipeline:

1. User query (natural language or structured)
2. For natural language: LLM interprets query and selects tools
3. Database query: Execute search/filter operations
4. Result ranking: Sort by relevance, recency, or custom criteria
5. Presentation: Format for output (table, JSON, detailed view)

Export Pipeline:

1. Select conversations (all, filtered, or specific IDs)
2. Load trees from database
3. Path selection: Choose which paths to export (longest, all, etc.)
4. Format conversion: Exporter transforms trees to target format
5. Optional sanitization: Remove sensitive data
6. Write output: File, stdout, or API response

5 Key Features and Design Decisions

5.1 Natural Language Queries

5.1.1 The Tool Calling Approach

Traditional keyword search assumes users know exact terms. CTK enables natural language queries like:

- "Show me starred conversations from last month"
- "Find discussions about Python async programming"
- "What conversations mention machine learning?"

Implementation uses LLM tool calling:

1. User provides natural language query
2. LLM receives query with tool descriptions
3. LLM selects appropriate tool (search, filter, get-by-id)
4. LLM generates tool parameters from query
5. System executes tool, returns results directly

5.1.2 Critical Design Pattern: Direct Output

Early versions let the LLM reformulate results, causing hallucination. Solution: Return tool results directly to user without LLM reformatting. This ensures:

- No invented conversations
- Exact database results
- Consistent formatting
- Verifiable outputs

5.1.3 Boolean Filter Semantics

Careful handling of optional boolean filters:

- **Filter absent**: Include all (no filtering)
- **Filter = true**: Include only items with flag
- **Filter = false**: Include only items without flag

System prompt includes explicit rules preventing LLM from incorrectly adding filters not mentioned in query.

5.2 Interactive Terminal UI

The TUI provides real-time conversation management:

Design Philosophy: Combine browsing, organization, and live chat in single interface. Users can:

- Browse conversations with rich formatting (colors, emojis)
- Search and filter with instant feedback
- Star/pin/archive conversations inline
- View conversation trees with path navigation
- Start live chat with LLMs while browsing
- Export selected conversations

Key Innovations:

1. **Context Switching**: Seamlessly move between browsing historical conversations and chatting with LLMs
2. **Tree Visualization**: ASCII art trees showing branching structure
3. **Path Navigation**: Step through different paths in branching conversations
4. **Inline Organization**: Modify metadata without leaving view

5.3 Privacy and Sanitization

Privacy-First Architecture:

- No telemetry or analytics
- All data stored locally
- No network access except optional LLM features
- No cloud sync (user controls if/how data is backed up)

Sanitization for Sharing:
Optional sanitization removes sensitive data before export:

- API keys (OpenAI, Anthropic, AWS, etc.)
- Passwords and authentication tokens
- SSH keys and certificates
- Database connection strings
- Credit card numbers
- Custom patterns via regex

This enables sharing conversation datasets for research while protecting user privacy.

5.4 Organization Strategies

Starring: Mark important conversations for quick access. Common uses:

- Conversations with valuable insights
- Reference solutions to recurring problems
- Conversations to review later

Pinning: Keep conversations at top of lists. Use cases:

- Active projects
- Frequently accessed templates
- Ongoing discussions

Archiving: Hide old conversations without deleting. Preserves:

- Conversation history for compliance
- Old experiments and explorations
- Resolved issues (might be relevant later)

Tagging: Flexible categorization with auto-tagging:

- Manual tags for custom categories
- Automatic tags from platform, model, date
- LLM-generated tags from content analysis

Design Decision: Timestamps Over Booleans
Instead of `starred: bool`, use `starred_at: timestamp`. Benefits:

- Preserves when operation occurred
- Enables sorting by recency of star
- Supports analytics (starring trends over time)
- Minimal additional complexity

6 Evaluation

6.1 Dataset

Evaluation uses real user data:

- 851 conversations across platforms
- 25,890 total messages
- 87% linear, 13% branching conversations
- Sources: ChatGPT (50%), Claude (34%), Gemini (11%), Copilot (5%)

6.2 Import Accuracy

Table 1 shows import success rates:

Platform	Conversations	Messages	Branches	Accuracy
OpenAI	423	12,847	156	100%
Anthropic	287	9,102	0	100%
Gemini	95	2,873	0	100%
JSONL	34	856	0	100%
Copilot	12	212	0	95%*

Table 1: Import accuracy by platform. *Copilot format varies by version.

Analysis: High accuracy across platforms demonstrates universality of tree model. Copilot's 95% reflects format variability across VS Code versions.

6.3 Natural Language Query Performance

Evaluated on 100 diverse queries:

Query Type	Success Rate	Avg Time
Simple filter (starred/pinned)	98%	0.3s
Keyword search	94%	0.8s
Combined (keyword + filter)	91%	0.9s
Complex multi-filter	87%	1.1s

Table 2: Natural language query accuracy and latency

Error Analysis: The 2-13% failure rate comes from:

- Ambiguous queries ("recent" without timeframe)
- Unusual phrasings outside training distribution
- Complex boolean logic ("not starred but pinned or archived")

Future work: Improve with query clarification dialogue.

6.4 Database Performance

Performance scales well to 100K+ conversations:

Operation	Time (851 conv)	Scaling
Import 1K messages	2.1s	Linear
Load conversation	15ms	Constant
Full-text search	45ms	Log(n)
Filtered list	8ms	Log(n)
Export 100 conv	1.3s	Linear

Table 3: Performance characteristics

SQLite with proper indexing provides excellent performance. Future optimization: SQLite FTS5 for full-text search.

6.5 User Study

Informal user feedback (N=15) over 3 months:

Positive Feedback:

- "Finally can search across all my AI conversations"
- "Natural language queries feel magical"
- "Tree visualization helps understand ChatGPT branches"
- "Export to JSONL for fine-tuning is invaluable"

Feature Requests:

- Semantic similarity search (beyond keyword)
- Web interface for mobile access
- Automatic conversation summarization
- Cross-conversation topic threads

7 Discussion

7.1 Lessons Learned

7.1.1 Universality Through Trees

The tree representation proved more universal than anticipated. Every conversation format encountered fits naturally:

- Linear conversations: Single-path trees (trivial case)
- ChatGPT branches: Natural branching at regeneration points
- Tool-using agents: Tool calls as special message content
- Multi-modal: Content heterogeneity separate from tree structure

This suggests trees are the *right abstraction* for conversations, not just a convenient compromise.

7.1.2 Tool Calling for Interpretation

Using LLM tool calling for query interpretation works remarkably well but requires careful design:

- **Explicit Instructions**: System prompts must be specific, not suggestive
- **Few-Shot Examples**: Examples prevent unwanted behaviors more than instructions
- **Direct Output**: Never let LLM reformulate tool results
- **Parameter Semantics**: Clear distinction between "omit" and "false"

This pattern likely applies to many domains beyond conversation management.

7.1.3 Plugin Architecture Benefits

Auto-discovery of plugins provided unexpected benefits:

- **Community Contributions**: Users can add format support independently
- **Rapid Prototyping**: New formats added in ¡ 1 hour
- **Format Evolution**: Platform changes don't require core updates
- **Specialization**: Domain-specific formats (e.g., medical AI logs) easily supported

The cost—dynamic loading complexity—is minimal compared to benefits.

7.2 Design Trade-offs

7.2.1 SQLite vs. Specialized Databases

Choosing SQLite over alternatives:

Advantages:

- Zero configuration, single file
- Excellent performance (¡ 100K conversations)
- ACID guarantees
- Universal availability
- Simple backup (copy file)

Limitations:

- Limited full-text search capabilities
- No native vector similarity
- Single-writer (fine for personal use)

Verdict: Correct choice for personal knowledge management. For enterprise use cases, might consider PostgreSQL.

7.2.2 Local-First vs. Cloud-Sync

CTK is deliberately local-first:

Why Local-First Wins:

- Privacy: User controls all data
- Latency: Zero network delay
- Reliability: Works offline
- Simplicity: No sync conflicts
- Cost: Zero ongoing fees

Cloud-Sync Considerations:

- Users can use Dropbox/iCloud for sync
- Git works well for SQLite files
- Future: Optional end-to-end encrypted sync

7.3 Future Directions

7.3.1 Semantic Similarity Search

Current search is keyword-based. Future: Vector embeddings for semantic similarity.

Approach: Integrate with complex-network-rag package:

- Path-level embeddings (not whole conversations)
- Network topology analysis (communities, bridges)
- Query expansion using similarity
- Cross-conversation topic threads

7.3.2 Conversation Analytics

Beyond basic statistics:

- Topic evolution over time
- Model comparison (which answers better?)
- Conversation quality metrics
- Token usage optimization

7.3.3 Collaborative Features

While maintaining privacy focus:

- Shared conversation collections (research teams)
- Conversation templates (best practices)
- Federated search (optional)
- Privacy-preserving analytics

7.3.4 Advanced Organization

- Hierarchical projects/folders
- Conversation relationships (related, supersedes, references)
- Smart collections (dynamic queries saved as views)
- Workflow integration (link to tasks, notes)

8 Conclusion

CTK demonstrates that unified conversation management across AI platforms is both feasible and valuable. By introducing a universal tree-based representation, we enable operations impossible within any single platform: cross-platform search, comparative analysis, flexible organization, and dataset preparation.

The system's success—851 conversations managed with 95-100% import accuracy, sub-second query performance, and positive user feedback—validates the core design decisions: tree-based universality, plugin-based extensibility, and local-first privacy.

Key contributions include:

1. **Universal Model**: Tree representation handling diverse conversation types
2. **Natural Language Interface**: Tool calling approach for intuitive queries
3. **Practical System**: Production-ready with multiple interfaces and formats
4. **Design Patterns**: Lessons applicable beyond conversation management

As AI assistants become ubiquitous, conversation management will grow in importance. CTK provides a foundation for treating conversational knowledge with the same rigor as documents, code, and research—as valuable artifacts worthy of careful curation, organization, and preservation.

Acknowledgments

CTK is open source and benefits from community contributions. We thank early users for feedback and testing.

References

[1] LangChain Development Team. *LangChain: Building applications with LLMs*. https://github.com/langchain-ai/langchain, 2023.

[2] Liu, J. *LlamaIndex: Data framework for LLM applications*. https://github.com/run-llama/llama_index, 2023.

[3] Ouyang, L. et al. *Training language models to follow instructions with human feedback*. NeurIPS, 2022.

[4] OpenAI. *ChatGPT API Documentation*. https://platform.openai.com/docs, 2024.

[5] Anthropic. *Claude API Documentation*. https://docs.anthropic.com, 2024.

[6] Hipp, D. R. *SQLite: Self-contained SQL database engine*. https://www.sqlite.org, 2000-2024.

[7] Torvalds, L. *Git: Distributed version control*. https://git-scm.com, 2005-2024.

[8] Obsidian. *Obsidian: Knowledge base on local Markdown files*. https://obsidian.md, 2024.