Allegheny College
Computer And Information Science Department

Junior Seminar Research Report

cotrace - Semantic Analysis and Summarization of Collaborative Document Revisions

by

Hemani Alaparthi

ALLEGHENY COLLEGE

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

Instructor: **Professor Gregory M. Kapfhammer**
Technical Leader: **Student Second Reader**

Abstract

A well-researched student project completed during the Junior Seminar.

Table of contents

List of Figures

List of Tables

1 Introduction

This research presents CoTrace, a browser extension and backend system designed to automatically summarize version history changes in Google Workspace documents and detect individual contributor impact within collaborative writing environments. CoTrace operates by analyzing revision histories across Google Docs and Google Sheets, extracting edit patterns, attributing changes to specific contributors, and generating human-readable summaries of how documents evolved. The tool combines local browser-side processing with a secure backend to normalize changes, detect contribution patterns, and measure the significance of individual edits without requiring manual review of version histories.

To understand the necessity of tools like CoTrace, it is essential to recognize the challenges inherent in collaborative document management. As collaborative authoring has become standard practice across academia, software development, business, and creative industries, teams increasingly work asynchronously on shared documents through platforms like Google Workspace. These collaborative workflows generate extensive version histories that capture every edit, but this granular data creates an accessibility problem: while version histories contain complete records of who changed what, extracting meaningful insights about document evolution and individual contributions remains labor-intensive and time-consuming.

Existing version control systems present this information in ways optimized for technical precision rather than human comprehension. Google Docs displays version histories chronologically but requires manual review to understand the significance of changes or their cumulative effect. GitHub provides detailed diff information but primarily serves software development use cases and requires technical expertise. No current tool effectively summarizes multi-author edits, synthesizes change patterns into narrative form, or measures contribution impact in ways that are accessible to non-technical collaborators.

CoTrace addresses this gap by introducing automated change summarization and contributor detection specifically designed for the Google Workspace ecosystem. The browser extension integrates directly into Google Docs and Google Sheets, providing users with immediate access to version analytics, per-author contribution metrics, and semantic summaries of how specific edits shaped document evolution. The backend processes revision data, applies natural language processing to categorize change types, and implements contribution scoring algorithms to measure individual impact. This approach transforms version history from a difficult-to-parse record into actionable insights that improve collaboration and accountability.

1.1 Motivation

1.1.1 The Collaborative Work Documentation Problem

The motivation for CoTrace stems from a fundamental challenge in collaborative work environments: the difficulty of understanding document evolution and measuring individual contributions in real time. As asynchronous collaboration has become standard practice, particularly in distributed teams, the volume of edits generated in shared documents has grown exponentially. This creates what we term the "version history accessibility problem"—complete edit records exist, yet extracting meaningful narratives about how work progressed and who contributed substantively remains labor-intensive.

This problem manifests across multiple domains. In academic settings, collaborative writing groups struggle to track which team members addressed which sections or resolved

identified issues. In business environments, managers can spend hours manually reviewing version histories to understand what work was completed and by whom. In software documentation and content teams, identifying who introduced errors or which edits resolved critical issues requires extensive manual review. Current tools fail to bridge the gap between granular edit data and high-level understanding of contribution patterns.

The significance of this challenge extends beyond convenience. Accurate measurement of individual contributions affects real-world outcomes including performance evaluations, grade assignment in collaborative projects, promotion decisions in corporations, and recognition of intellectual labor. When contribution measurement relies on incomplete information or subjective assessment, the resulting decisions may be biased, unfair, or inaccurate. CoTrace addresses this by providing objective, automated measurement of contribution patterns in collaborative documents.

1.1.2 Personal Motivation

The development of CoTrace was shaped by direct personal experience with the limitations of existing collaboration tools. In multiple academic group projects, a recurring frustration was that version histories contain granular data about who changed what, yet no mechanism exists to transform that data into a fair summary of contribution. For instance, a student who rewrites a flawed argument contributes more meaningfully than one who corrects punctuation, yet simple edit-count metrics treat both identically. Witnessing unequal credit distribution in these settings made the problem feel personally urgent and worth addressing through a purpose-built tool.

1.1.3 Professional Motivation

From a professional standpoint, the rise of remote and hybrid work has made this problem more acute. Tools like Google Workspace have become the default infrastructure for distributed teams, yet the analytics layer of those tools has not kept pace with how teams actually work. Project managers and team leads commonly lack the instrumentation to answer basic questions about how a deliverable evolved or which team members drove the most substantive changes. CoTrace directly addresses this gap by bringing contribution intelligence into the environments where collaborative work already happens, without requiring teams to adopt new workflows or platforms.

1.1.4 Labor Productivity and Accountability

Beyond understanding document evolution, there is a broader motivation related to labor productivity and accountability. In collaborative environments, when contribution patterns are opaque, some team members may contribute significantly while others contribute minimally, yet all receive equal credit. This lack of visibility affects those responsible for evaluating performance, as they lack objective data to assess individual impact.

Research in organizational behavior consistently shows that perceived unfairness in credit attribution reduces motivation and team cohesion [7]]. Tools that make contribution visible serve not only evaluative purposes but also act as accountability mechanisms that encourage equitable participation. CoTrace is designed with this dual function in mind: it is both an analytics instrument and a transparency mechanism that benefits all stakeholders.

1.1.5 Current Limitations of Existing Solutions

Version control and change tracking tools address different aspects of the collaboration problem but fail to solve the contribution measurement challenge comprehensively. Google Docs, for example, provides version history but presents changes chronologically and requires manual synthesis to extract meaning. GitHub excels at code change tracking but is optimized for technical workflows and includes steep learning curves for non-developers. Spreadsheet tools like Google Sheets lack sophisticated change tracking altogether. None of these systems provide semantic summarization of changes or automated contribution scoring designed for non-technical users.

Furthermore, existing tools are primarily designed for retrospective analysis rather than real-time insight. Users must interrupt their workflow, navigate to version history interfaces, and spend time reviewing changes. This friction means that comprehensive understanding of document evolution often occurs only when necessary rather than continuously during collaboration.

The CoTrace project builds upon existing version control tools by introducing a layer of intelligence specifically optimized for understanding collaboration dynamics in Google Workspace environments. Rather than replacing existing tools, CoTrace augments them with summarization and contribution detection capabilities designed for the non-technical collaborator.

1.2 Current State of the Art

1.2.1 Version Control and Change Tracking Systems

Version control represents a mature field with well-established tools and methodologies. Git, the dominant version control system, provides tracking of source code changes through commits, branches, and detailed diff information. While Git excels at managing code repositories and enabling collaboration among developers, it requires technical expertise to interpret diffs and understand how changes relate to broader project goals. The learning curve for non-technical users remains steep, limiting its applicability in domains like academic writing, business documentation, and creative collaboration.

Google Docs and similar cloud-based collaborative editors provide version history functionality accessible to non-technical users. These systems automatically save versions and allow users to view changes over time. However, their version history interfaces present information chronologically and show only basic metadata about who edited when and the chunks of edits done. They lack semantic understanding of changes—users cannot easily ask questions like "what was the main point of this version?" or "which edits addressed the peer review feedback?" without manually reading through edits.

1.2.2 Gaps in Existing Solutions

Despite progress in individual areas—version control, contribution measurement, and change summarization—no existing tool comprehensively addresses the problem of understanding collaborative document evolution and measuring contribution impact for non-technical users. The existing landscape consists of:

- **Version control systems** (Git, Google Docs history) that provide complete but raw information

- **Contribution measurement tools** that focus on code repositories or require manual input
- **Collaboration analytics platforms** that focus on communication or time tracking rather than contribution to document content

CoTrace represents the first integrated system designed specifically to address this gap by combining automated change summarization, contribution detection, and non-technical accessible interfaces for the Google Workspace ecosystem.

1.3 Goals of the Project

The primary objectives of CoTrace are to automate version history summarization, detect and measure individual contributions, provide transparent collaboration analytics, and enable fair assessment of collaborative work.

First, automate version history summarization. CoTrace generates concise, semantic summaries of document changes that capture what evolved between versions without requiring users to manually review edit histories. Rather than presenting raw diffs or chronological lists, the tool uses NLP and semantic analysis to synthesize changes into human-readable narratives. For example, recognizing that multiple edits collectively 'addressed peer review feedback' rather than listing each edit separately.

Second, detect and measure individual contributions. CoTrace attributes edits to specific contributors and implements algorithms to measure the significance of individual contributions. This goes beyond simple edit counts to assess the substantive impact of changes, including whether edits resolved key issues, introduced errors, or synthesized feedback into coherent document structure.

Third, provide transparent collaboration analytics. By aggregating contribution data across multiple documents and time periods, CoTrace creates analytics dashboards that make collaboration patterns visible. These visualizations enable teams to understand who contributed to what, when contributions occurred, and the relative impact of different team members.

Fourth, enable fair assessment of collaborative work. By providing objective, detailed records of individual contributions, CoTrace supports more accurate and equitable decision-making in contexts including performance evaluation, grade assignment, and recognition of intellectual labor. The tool reduces reliance on subjective assessments or incomplete information when measuring contribution to group work.

1.4 Technical Approach

CoTrace is architected as a distributed system combining client-side browser extension functionality with cloud-based backend processing. This architecture ensures that computationally intensive tasks are handled server-side while maintaining user interactivity through real-time browser integration. The system processes Google Workspace revision histories through multiple analytical layers: first extracting raw revision data via the Google Drive API, then normalizing and structuring that data, applying natural language processing to categorize changes, and finally computing contribution metrics based on content similarity analysis and edit attribution.

The browser extension serves as the user-facing interface, providing real-time visualization of contribution metrics alongside document editing without requiring users to navigate

to external applications. The extension captures editing context, manages user authentication, and communicates with both Google's APIs and CoTrace's backend services. The backend implements the core analytical algorithms including revision normalization, change categorization through NLP, and contribution scoring based on algorithmic assessment of edit significance rather than simple metrics like word count or edit frequency.

1.5 Ethical Implications

The development of tools that measure and quantify collaboration raises important ethical considerations regarding fairness, transparency, and accountability in group work assessment. CoTrace addresses several key ethical dimensions including equitable contribution measurement, prevention of misuse for surveillance, algorithmic fairness in contribution scoring, and data governance.

1.5.1 Fair Assessment and Accountability

One primary ethical consideration is ensuring that CoTrace contributes to fair and equitable assessment of collaborative contributions rather than enabling biased decision-making. The tool is designed with the understanding that contribution measurement should be transparent and explainable. All contribution scores are accompanied by clear explanations of the methodology used, enabling human reviewers to validate or contextualize algorithmic assessments. The tool intentionally avoids assigning binary judgments ("good contributor" vs "poor contributor") and instead provides granular data that allows human evaluators to apply context and nuance to decision-making.

Furthermore, CoTrace acknowledges that different forms of contribution carry different significance in different contexts. The tool supports multiple contribution metrics rather than reducing collaboration to a single score. This allows users to understand the full range of how individuals contributed and to weight different types of contributions appropriately for their specific context.

1.5.2 Prevention of Misuse for Surveillance

As with any tool that provides visibility into work behavior, CoTrace must address the potential for misuse as a surveillance mechanism. The tool is designed to ensure it remains transparent to all collaborators. Users can see that CoTrace is analyzing their document, and the extension's operation is visible rather than hidden. Documentation clearly specifies intended use cases and discourage surveillance-oriented applications.

Additionally, CoTrace is designed to work within shared documents where all parties have editing access, preventing scenarios where one person monitors others' work without their knowledge. The tool does not enable tracking of user activity outside the shared document or monitoring of individuals who have not consented to measurement.

1.5.3 Algorithmic Fairness in Contribution Measurement

CoTrace implements contribution scoring algorithms, which creates a responsibility to ensure these algorithms do not inadvertently disadvantage specific groups or individuals. The tool is designed to avoid biases that might favor certain types of contributions over others—for example, avoiding systems that privilege quantity of edits over quality of impact, or that disadvantage contributors with different work styles.

The project acknowledges that what constitutes valuable contribution varies across contexts. In academic writing, theoretical framing may be more significant than formatting work, while in technical documentation, clear organization may be as important as content creation. CoTrace supports contextualized assessment rather than imposing a single definition of valuable contribution.

1.5.4 Data Governance and Privacy

While CoTrace measures collaboration patterns rather than tracking privacy-sensitive data, the tool handles sensitive information including work history, authorship attribution, and potentially confidential document content. The project commits to handling this data according to established privacy principles, including data minimization, purpose limitation, and user consent. Data retention policies ensure that contribution data is not stored beyond the operational requirements of the summarization and analytics features.

1.6 Research Contributions

This project contributes to the field of collaborative work systems and human-computer interaction in several concrete ways. **First, CoTrace demonstrates that automated summarization of collaborative document changes is feasible and useful for non-technical users, addressing a meaningful gap in existing version control tools.** By operationalizing NLP-based change categorization within a browser extension, the project shows that semantic document intelligence can be delivered without requiring users to leave their editing environment. **Second, the project provides an implemented system for measuring individual contributions to collaborative work, contributing empirical evidence about what metrics are meaningful and practical in real-world settings.** The contribution scoring algorithms developed for CoTrace extend prior academic work on revision analysis into an operational context for the first time. **Third, by open-sourcing CoTrace and its algorithms, the project enables future research into contribution detection, attribution systems, and fair assessment mechanisms for collaborative work.** The system architecture, API design, and contribution detection methodology are documented in sufficient detail to serve as a foundation for follow-on research in collaborative intelligence and human-computer interaction.

2 Related work

Since collaborative document editing has become standard practice in academic and professional settings, understanding how teams work together and measuring individual contributions has grown increasingly important. However, tools that help users quickly comprehend document evolution and fairly measure contribution impact remain limited. This gap presents a valuable opportunity for new research and innovation in collaborative work systems. As teams increasingly rely on asynchronous collaboration through platforms like Google Workspace, the need for automated change summarization and contribution measurement becomes critical. Additionally, understanding the current landscape of collaboration analytics tools and their limitations provides important context for how CoTrace addresses unmet needs in the ecosystem.

Challenges in collaborative work can be broadly categorized into two problems: understanding document evolution and measuring contribution fairness. Understanding document evolution refers to the difficulty of comprehending what changed in a document, why it changed, and how those changes shaped the final product. Measuring contribution fairness refers to the challenge of objectively assessing individual contributions to collaborative work without relying on incomplete information or subjective judgment. This paper focuses primarily on these two interconnected challenges and how semantic change summarization combined with contribution detection can address them in ways existing tools do not.

2.1 Rise of Collaborative Work and Remote Teams

Since the widespread adoption of cloud-based document editors following Google Docs' introduction in 2006, collaborative writing and asynchronous document editing have become fundamental to how teams work. Early collaborative systems focused on real-time co-editing capabilities, allowing multiple users to edit simultaneously and see each other's cursors [6]. This represented a significant advancement over email-based document sharing, which created version confusion and made coordination difficult.

The COVID-19 pandemic further accelerated the adoption of cloud-based collaborative tools, with organizations forced to shift to remote and hybrid work models. Google Workspace, Microsoft 365, and similar platforms became essential infrastructure for teams across academia, business, and creative industries. However, as teams increasingly worked asynchronously across time zones, a new problem emerged: understanding document evolution and measuring individual contributions became significantly more difficult. Unlike real-time co-editing where users can observe changes as they happen, asynchronous collaboration generates extensive version histories that are difficult to synthesize and understand.

Research in computer-supported cooperative work (CSCW) has extensively studied how collaborative tools affect group dynamics and productivity [9]. A key finding is that awareness of others' work and understanding what collaborators are doing, why they are doing it, and what they have accomplished significantly improves coordination in distributed teams. However, most awareness mechanisms focus on real-time presence and activity rather than historical understanding of work evolution and contribution patterns.

2.2 Version Control and Change Tracking Systems

Version control systems represent a mature field with well-established tools and methodologies, yet each approach involves a fundamental trade-off between technical power and

accessibility.

2.3 Git

Git, the dominant version control system, provides *granular* tracking of source code changes through commits, branches, and detailed diff information [1]. While Git excels at managing code repositories and enabling collaboration among developers, it requires technical expertise to interpret diffs and understand how changes relate to broader project goals. The learning curve for non-technical users remains steep, limiting its applicability in domains like academic writing, business documentation, and creative collaboration. In contrast to CoTrace, which is designed specifically for non-technical collaborative document users, Git assumes a level of developer fluency that excludes much of its potential audience.

2.4 Google Docs

Google Docs and similar cloud-based collaborative editors provide version history functionality that is accessible to non-technical users. These systems automatically save versions and allow users to view changes over time. However, their version history interfaces present information chronologically, show only basic metadata about who edited when and chunks of the edits made in the document . They lack semantic understanding of changes. For example, users cannot easily ask questions like "what was the main point of this version?" or "which edits addressed the peer review feedback?" without manually reading through edits. Where Git sacrifices accessibility for analytical depth, Google Docs sacrifices analytical depth for accessibility, and CoTrace targets exactly this gap by providing semantic understanding without requiring technical expertise.

2.5 Microsoft Word and Google Workspace Collaboration Features

Microsoft Word's track changes feature and similar tools within Google Workspace provide comment-based collaboration but focus on presentation of raw changes rather than synthesis or impact measurement. These tools document that changes occurred but do not explain their significance or contribution impact. While platforms like Google Workspace have introduced more collaborative writing features over time, such as suggestion mode, comment threads, and real-time co-editing, they remain focused on facilitating the editing process itself rather than analyzing or summarizing the outcomes of that process. The fundamental limitation across all of these tools is the same: they surface what changed but provide no mechanism for understanding why it mattered or how much any individual contributor shaped the final document.

Taken together, these tools reveal a consistent pattern: the more accessible the tool, the less analytical power it provides. CoTrace is designed to break this trade-off by combining the accessibility of cloud-based editors with the contribution insight that currently only exists in highly technical systems. Addressing this requires advances in automated change summarization, which the following section examines.

2.6 Contribution Measurement and Attribution

Research into contribution measurement has primarily focused on software engineering contexts. Tools and frameworks exist for measuring code contributions, including metrics based

on commit counts, lines of code changed, and issue resolution [3]. However, these metrics are known to be imperfect proxies for actual contribution value and can be misleading when applied without context. A single critical bug fix may be worth more than thousands of lines of code that are never used. Unlike CoTrace, which aims to measure the semantic significance of contributions rather than their raw volume, these software engineering approaches reduce contribution to quantities that are easy to count but difficult to interpret meaningfully. Translating this insight into the document collaboration domain requires natural language processing capabilities that have only recently matured sufficiently to make this feasible.

2.7 Automated Change Summarization and Natural Language Processing

Recent advances in large language models have enabled more sophisticated approaches to understanding and summarizing code and document changes. There has been research on using neural networks to automatically generate commit messages summarizing code changes [5]. Their work demonstrated that models could learn to summarize diffs in semantically meaningful ways, providing a foundation for automated change understanding.

Building on this work, subsequent research has explored using large language models for understanding broader development practices. The emergence of transformer-based models like GPT have made it possible to analyze text at scale and generate coherent summaries of complex changes [4]. However, most existing work focuses on source code rather than general document collaboration, a distinction that matters because document changes involve rhetorical and structural shifts that code-focused models are not trained to recognize.

In the document domain, research on automatic text summarization has made significant progress. Multi-document summarization and update summarization have been studied, but application to collaborative document editing remains limited. The specific challenge of understanding semantic changes in documents, for example where content is reordered, rewritten, or synthesized, presents unique difficulties beyond traditional abstractive summarization [10]. Existing summarization approaches typically assume independent documents rather than versions of the same document with cumulative changes. CoTrace addresses this directly by treating document versions as a sequence of evolving states rather than independent texts, enabling it to detect and summarize the semantic delta between versions in a way prior summarization research has not attempted. Ensuring this summarization is also fair in how it attributes contributions requires engaging with a broader body of research on measurement ethics.

2.8 Fairness and Ethical Considerations in Work Measurement

The history of performance measurement and evaluation reveals important lessons for contribution measurement systems. While systematic measurement can enable fairer assessment, it can also create perverse incentives, encourage gaming of metrics, and reduce human work to oversimplified quantification [8].

In modern contexts, the rise of algorithmic management and surveillance in workplaces has raised ethical concerns about worker autonomy and privacy [8]. While CoTrace aims to improve fairness in contribution assessment, it must be designed to avoid becoming a surveillance tool that enables excessive monitoring or undermines worker autonomy. Transparency in algorithmic decision-making is increasingly recognized as important for fairness

and user trust.

Contemporary efforts to address fairness in assessment include developing "altmetrics" that provide multiple perspectives on impact rather than single scores [**?**], and implementing transparent rubrics that make assessment criteria explicit [**?**]. These approaches align with CoTrace's design philosophy of providing multiple contribution metrics rather than reducing collaboration to a single score, a philosophy that distinguishes CoTrace from the single-metric approaches common in software engineering contribution tools.

2.9 Academic Integrity and Writing Process Analysis Tools

Recent tools have emerged to assist educators in understanding student writing processes and detecting anomalies that may indicate plagiarism or AI generation. Revision History, widely adopted in educational contexts with over 200,000 users, tracks editing sessions, detects copy/paste events, and identifies unusual writing patterns such as sudden large text additions or inconsistent editing velocities [2]. These tools focus on flagging suspicious patterns to prompt teacher investigation.

While Revision History effectively identifies unusual patterns, it operates at the level of detection rather than understanding. It can flag that a large block of text was added suddenly, but it does not explain what that text is, how it relates to previous content, or whether it represents substantive contribution. Similarly, while it tracks edit counts, it does not measure the significance of individual contributions, as a single critical edit is treated the same as a minor typo fix. This creates a trade-off similar to the one seen in version control: Revision History provides useful signals but no semantic insight, while CoTrace provides semantic insight without positioning itself as an integrity enforcement tool. CoTrace builds on this foundation by shifting from anomaly detection to semantic understanding of changes and contribution measurement.

2.10 Gap and Contribution

The existing literature reveals a clear gap at the intersection of collaborative document editing, automated change understanding, and fair contribution measurement. While mature work exists in each individual area, and emerging tools address academic integrity concerns, no integrated system specifically addresses the problem of understanding collaborative document evolution and measuring fair contribution in ways that reduce manual analysis burden while promoting equitable assessment.

Existing tools fall into distinct categories: (1) version control systems provide complete but raw information requiring manual synthesis; (2) writing process analysis tools focus on detecting anomalies rather than understanding meaning; and (3) contribution measurement research emphasizes software engineering contexts rather than general document collaboration.

CoTrace fills this gap by specifically targeting the collaboration fairness and change comprehension problem. Rather than asking "Is this work authentic?" (the concern of academic integrity tools), CoTrace asks "What actually changed in this document, and who contributed what?" The system combines insights from version control, natural language processing, contribution measurement research, and fairness-aware algorithmic design into a unified system tailored to the Google Workspace ecosystem and designed for both educational and team collaboration contexts.

This project contributes by: (1) demonstrating that automated, semantically meaningful summarization of document changes is feasible and valuable for non-technical users, reducing the time and effort required to understand document evolution; (2) providing an implemented system for detecting and measuring individual contributions to collaborative documents in ways that highlight substantive impact rather than edit counts; and (3) grounding contribution measurement in fairness-aware design principles that prioritize transparency, multiple metrics, and user understanding over simple quantification, principles that apply equally to classroom group projects and distributed team work.

3 Method of approach

This chapter describes the technical implementation of cotrace, a Chrome Extension designed for semantic summarization of collaborative document changes. The system architecture, algorithms, technologies, and development methodologies used to build cotrace are detailed below.

3.1 System Architecture

cotrace follows a client-server architecture pattern optimized for browser-based deployment and real-time collaboration analysis. The system consists of three primary layers: the Chrome Extension frontend, a local Node.js backend server, and a set of external APIs comprising Google Drive, Google OAuth 2.0, and the Anthropic Claude API. Each layer has a clearly scoped responsibility, and the separation between them reflects deliberate design decisions around security, performance, and extensibility.

3.1.1 Architecture Overview

The frontend extension runs within the Chrome browser context and is responsible for all user-facing interactions: document detection, revision browsing, per-author contribution display, and chat-based querying. It communicates exclusively with a local backend server running at `http://localhost:3000`, which manages the resource-intensive operations of content retrieval from Google Drive and AI-powered semantic analysis via the Claude API. This hybrid architecture was chosen for two primary reasons. First, it keeps all authentication tokens, OAuth credentials, and chat history on the user's machine, ensuring that no sensitive data is transmitted to or stored on a third-party server. Second, it offloads computationally expensive API orchestration, including pagination through revision histories and prompt construction for Claude, to a process running outside the browser's sandboxed extension context, which has stricter resource and network constraints.

The system processes collaborative document changes through five sequential stages:

1. Detecting the active Google Doc, Sheet, or Slide by inspecting the current browser tab URL
2. Fetching revision metadata and paginated revision content from the Google Drive API v3
3. Computing a unified diff between any two selected revision snapshots
4. Generating a semantic summary of those differences using the Claude API
5. Persisting the resulting chat history locally in Chrome Storage for continuity across sessions

This end-to-end pipeline transforms raw version history, which Google exposes as timestamped binary snapshots with no inherent semantic labeling, into human-readable, author-attributed summaries that a collaborator can query conversationally.

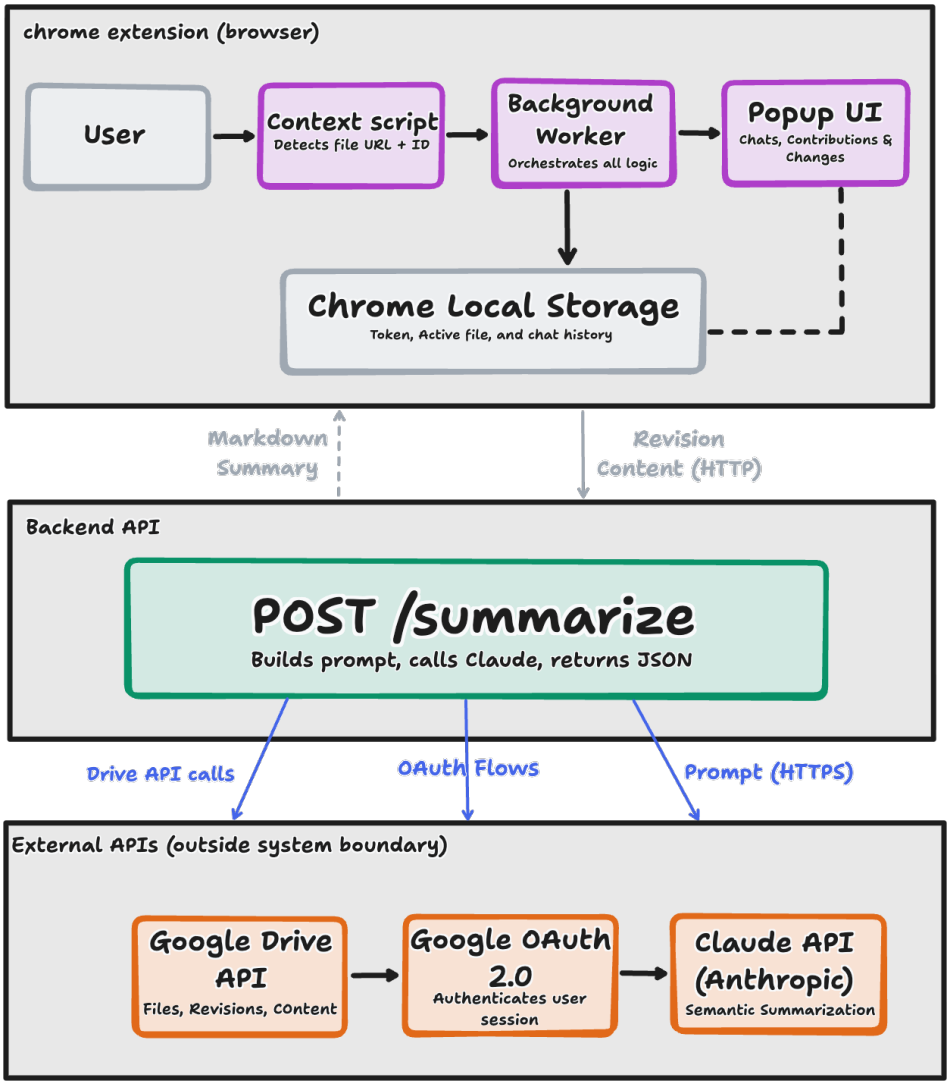

Figure 1: System Architecture Diagram

3.1.2 Design Rationale: Local Backend vs. Hosted Service

A central architectural decision was to run the backend server locally rather than hosting it as a cloud service. This choice directly addresses the privacy requirements of collaborative document analysis: users querying cotrace may be working with confidential documents, proprietary research, or commercially sensitive content. Routing document content through a third-party hosted server would introduce data retention risks and potential compliance concerns. By running the backend locally, cotrace guarantees that document content never leaves the user's machine outside of the direct API calls to Google and Anthropic that the user has explicitly authorized.

The tradeoff of this approach is that users must have Node.js installed and must start the

backend server manually before using the extension, a friction point that would be addressed in a production deployment by bundling the server as an Electron process or packaging it into the Chrome Extension itself using a service worker with persistent background execution.

3.1.3 Technology Stack

The frontend is built entirely on Chrome Extension APIs conforming to Manifest v3, the current extension standard introduced by Google in 2023. Manifest v3 replaces persistent background pages with ephemeral service workers, which are event-driven processes that terminate when idle and are restarted on demand. This architectural constraint has a direct consequence for cotrace's state management: because the background service worker cannot hold data in memory across invocations, all persistent state, including the OAuth access token, the currently active file metadata, and per-document chat history, must be serialized to Chrome's local storage API. Every module that requires this data reads it from storage at the point of use rather than relying on in-memory references. This design pattern, while adding a small latency overhead per operation, ensures the extension remains compatible with Manifest v3's resource lifecycle and avoids the class of bugs that arise from stale in-memory state in long-running sessions.

The backend is implemented in Node.js using the Express framework for HTTP routing. Node.js was selected because its asynchronous, non-blocking I/O model is well-suited to the workload profile of the cotrace backend, which spends the majority of its execution time awaiting responses from external APIs rather than performing CPU-bound computation. A single incoming request to the `/summarize` endpoint may trigger multiple sequential and parallel outbound requests: first to Google Drive for revision metadata, then for the content of each revision snapshot, and finally to the Anthropic Claude API, all of which are I/O-bound. Node.js handles this concurrency model efficiently without the overhead of thread-per-request architectures that would be required in a synchronous server framework.

The AI summarization layer uses the Anthropic Claude API, specifically prompted to produce structured JSON output conforming to a schema that captures the authoring user, a semantic description of what changed, and the overall editing pattern (e.g., restructuring, expansion, clarification, or deletion). This structured output format was chosen over free-form prose generation because it enables the extension to render summaries in a consistent UI layout and supports future filtering of summaries by author or change type. The backend renders the structured JSON response into Markdown before returning it to the extension, where the `marked.min.js` library handles final HTML rendering in the popup.

3.1.4 Component Responsibilities

The Chrome Extension is decomposed into three functional subsystems. The content script (`content.js`) is injected into every Google Docs, Sheets, and Slides page and is responsible for a single task: detecting the file ID and document type from the current URL and writing this information to Chrome Storage. This tight scoping ensures the content script has the minimal permissions necessary: it reads the URL and writes to storage, nothing else, and this reduces the attack surface of the extension in the browser context.

The background service worker (`background-new.js`) acts as the central coordinator for all non-UI logic. It handles the Google OAuth 2.0 authorization flow, manages token storage and refresh, constructs and dispatches all outbound API requests to Google Drive and the local backend, and routes messages between the content script and the popup interface.

Concentrating this logic in the service worker rather than the popup ensures that API calls can complete even if the user closes and reopens the popup mid-operation.

The popup interface (`popup2-new.html` and its associated modules in `src/popup/`) is responsible exclusively for rendering and user interaction. It is organized into three views:

- A chat tab for natural language querying of document changes
- A contributions tab that breaks down revision authorship by collaborator
- A changes tab that renders a syntax-highlighted unified diff between any two selected versions

Separating the rendering layer from the data-fetching layer means the popup can be rebuilt or redesigned without modifying the underlying API integration logic.

3.1.5 System Boundary

cotrace operates entirely within the boundary of documents the authenticated user has permission to access through Google Drive. The extension requests only read-only OAuth scopes (`drive.readonly`, `drive.metadata.readonly`, `documents.readonly`, `spreadsheets.readonly`) and cannot modify, delete, or share any document. All data processing occurs either locally in the browser or on the user's local backend server, with the exception of the two explicitly user-authorized external calls: to Google Drive for revision content and to the Anthropic Claude API for semantic summarization. No telemetry, usage data, or document content is transmitted to any other endpoint.

3.2 Content Revision Architecture

The system implements a pagination-based retrieval mechanism to handle documents with extensive revision histories. Google Drive API returns revisions in paginated batches (default ~100 items per page). The `google-api.js` module implements pagination using `pageToken` to fetch all available revisions:

```
// Pagination logic for complete revision history
while (nextPageToken) {
  const response = await drive.revisions.list({
    fileId: fileId,
    fields: 'revisions(id,modifiedTime,lastModifyingUser,size)',
    pageToken: nextPageToken
  });
  revisions.push(...response.data.revisions);
  nextPageToken = response.data.nextPageToken;
}
```

This approach enables cotrace to handle documents with 100+ revisions, converting a practical limitation into comprehensive coverage of authorship and change patterns.

3.3 Change Analysis Algorithm

The core algorithm for change analysis operates in three phases: content retrieval, diff generation, and semantic summarization.

Phase 1: Content Retrieval - Accepts fileId, version IDs (revision1Id, revision2Id), and authentication token - Fetches full content for both revisions from Google Drive API - Returns content as text for comparison

Phase 2: Diff Generation The system implements unified diff format using the 'diff' npm package:

$$\text{diff}(v_i, v_j) = \text{createPatch}(\text{filename}, \text{content}_i, \text{content}_j)$$

The unified diff output includes: - 3-line context before and after each change - Lines prefixed with '+' for additions ($\rightarrow$ green highlighting) - Lines prefixed with '-' for deletions ($\rightarrow$ red highlighting)
- Lines prefixed with '@@' marking change location/extent

Phase 3: Semantic Summarization The backend generates a system prompt that guides Claude API to synthesize changes rather than enumerate them:

$$S = \text{Claude}(P, D, T_{max})$$

where: - P = system prompt emphasizing executive summaries and pattern identification - D = complete unified diff between two versions - T_{max} = token limit (1000 tokens for comprehensive analysis)

The Claude API response prioritizes: - Main topics/sections that changed - Editing patterns (restructuring, expansion, clarification, deletion) - High-level synthesis emphasizing important insights

3.4 Persistent State Management

Chat history persists locally in Chrome Storage using a file-keyed approach:

```
const chatKey = `chatHistory_${fileId}`;
const messages = await chrome.storage.local.get(chatKey);
```

Each message object contains: - `type`: "user" or "system" - `text`: message content - Implicit `timestamp`: from storage metadata

This design enables users to maintain context across browser sessions without transmitting message history to external servers.

3.5 Performance Optimization

For documents with extensive revision histories, the system implements two optimization strategies:

1. **Analytical Sampling**: When analyzing all changes, the system processes only the most recent 20 revisions using parallel fetching via `Promise.all()`, providing 10-15x performance improvement for documents with 100+ revisions while capturing recent activity.

2. **Smart Caching**: The extension always fetches fresh revision metadata before each user query, preventing stale data in long-running sessions.

3.6 Data Flow

The interaction between components follows this sequence:

1. **User Query** → Extension popup (chat.js)
2. **Token Retrieval** → Chrome Storage (authentication)
3. **API Call** → Backend server with fileId, query, authToken
4. **Revision Fetch** → Google Drive API returns metadata
5. **Content Retrieval** → Google Drive API fetches revision content
6. **Diff Generation** → Node.js creates unified diff
7. **AI Analysis** → Claude API generates semantic summary
8. **Response Display** → Extension renders results in popup UI
9. **History Storage** → Chrome Storage saves messages locally

3.7 Error Handling and Reliability

The system implements graceful degradation: - If Claude API fails, diff statistics (additions/deletions counts) are displayed without semantic summary - Missing or invalid authentication tokens trigger re-authentication flow - Network errors for Google Drive API calls display user-friendly error messages - Diff generation failures fall back to basic version metadata comparison

4 Experimental Results

This chapter describes your experimental setup and evaluation. It should also produce and describe the results of your study. The section titles below offer a typical structure used for this chapter.

Note for Junior Seminar: This chapter should be approximately 5-7 pages in length. Focus on presenting your key findings clearly. Include tables, charts, or graphs to visualize your results. Discuss what worked well and what didn't, and explain why.

4.1 Experimental Design

Describe how you designed your experiments or evaluation. What questions were you trying to answer? What metrics did you use? How did you collect your data?

Especially as it pertains to responsible computing, if conducting experiments or evaluations that involve particular ethical considerations, detail those issues here.

4.2 Evaluation

Present your results here. Use tables, figures, and charts to display your data. Discuss what the results show and how they relate to your research questions.

4.3 Threats to Validity

Discuss any limitations of your study or factors that might affect the validity of your results. This shows critical thinking about your work.

5 Conclusions and Future Work

Traditionally, this chapter addresses the areas proposed below as sections, although not necessarily in this order or organized as offered. However, the last section – "Ethical Implications" is required for this chapter. See the heading below for more details.

Note for Junior Seminar: This chapter should be approximately 5-7 pages in length. Summarize what you accomplished, reflect on what you learned, and discuss how this work could be extended in the future.

5.1 Summary of Results

Summarize the key findings and contributions of your research. What did you accomplish? What were the main outcomes of your project?

5.2 Future Work

Describe potential extensions or improvements to your work. What would you do next if you had more time? What questions remain unanswered?

5.3 Future Ethical Implications and Recommendations

Especially as pertains to the public release or use of your software or methods, what unresolved or special issues remain? What recommendations might you make? How could future work address ethical concerns raised by your research?

References

[1] 2025. *Git.* https://git-scm.com/

[2] 2026. *Revision History.* https://www.revisionhistory.com/

[3] Clei and Journal. 2021. Using Git Metrics to Measure Students' and Teams' Code Contributions in Software Development Projects. *[Name of Journal]* 24, 2 (2021). https://pdfs.semanticscholar.org/668f/bda57a831cab4c74e5c2a0023f7a88a69fbf.pdf

[4] Anon Hossain. 2025. *How Transformers Revolutionized NLP, AI and Generative Models.* https://medium.com/@anonhossain1710/how-transformers-revolutionized-nlp-ai-and-generative-models-62c7ed0d6c7f

[5] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. (oct 2017). https://doi.org/10.1109/ase.2017.8115626

[6] Hilary Johnson and Joanne Hyde. 2003. Towards Modeling Individual and Collaborative Construction of Jigsaws Using Task Knowledge Structures (TKS). *ACM Transactions on Computer-Human Interaction* 10, 4 (dec 2003), 339–387. https://doi.org/10.1145/966930.966934

[7] Jennifer Lerner and Philip Tetlock. 1999. Accounting for the Effects of Accountability. *Psychological Bulletin* 125 (1999). https://doi.org/10.1037/0033-2909.125.2.255

[8] David Manheim. 2023. Building Less-Flawed Metrics: Understanding and Creating Better Measurement and Incentive Systems. *Patterns* 4, 10 (oct 2023), 100842. https://doi.org/10.1016/j.patter.2023.100842

[9] Gregor Polančič et al. 2013. An Experimental Investigation Comparing Individual and Collaborative Work Productivity When Using Desktop and Cloud Modeling Tools. *Empirical Software Engineering* 20, 1 (sep 2013), 142–175. https://doi.org/10.1007/s10664-013-9280-x

[10] None Supriyono et al. 2024. A Survey of Text Summarization: Techniques, Evaluation and Challenges. *Natural Language Processing Journal* 7 (jun 2024), 100070. https://doi.org/10.1016/j.nlp.2024.100070