GOOSVC: Version Control for Content Creation with Generative AI

David Grünert[1,2], **Alexandre de Spindler**[1] and **Volker Dellwo**[2]
[1]Zurich University of Applied Sciences, Winterthur, Switzerland
[2]Department of Computational Linguistics, University of Zurich
{grud,desa}@zhaw.ch

Abstract

This paper introduces GOOSVC[1], a version control system for content creation using generative AI. As generative AI models become integral to creative workflows, managing iterative changes, branching, and merging of content is challenging. Current version control systems are not designed for these workflows, which involve multiple AI assistants exchanging text, images, or other artifacts. In this paper, we identify the core requirements for such a system and show how GOOSVC meets them. Our system provides full traceability and versioning of both artifacts and conversation states, allowing seamless integration of multiple AI assistants into creative workflows.

1 Introduction

Generative AI has rapidly evolved into a powerful creative partner in domains such as marketing, design, data science, and creative writing (Davenport and Mittal, 2022). Individuals and teams often rely on large language models (LLMs) or multimodal AIs to brainstorm ideas, refine concepts, and generate or revise content (White et al., 2023). Despite these successes, complex workflows, in which users combine multiple AI assistants, pose significant challenges. Studies confirm that creative work requires fluid human–AI co-creation (McGuire et al., 2024; Rezwana and Maher, 2023). Such co-creation is rarely linear: users frequently need to revise, branch, or revert to earlier prompts, and must manage a growing collection of text, images, audio files, or other artefacts in the process (Cygnis, 2024; Kumar and Suthar, 2024; Coca-Cola, 2023). Because of the non-deterministic nature of generative AI, but also for legal reasons (European-Commission, 2020), AI-assisted workflows must offer end-to-end traceability of all generated artefacts *including* all AI interactions.

[1]https://goosvc.com

While conventional version control systems (VCS) such as Git track file changes and allow branching and merging, they are not designed to handle dynamic AI-generated outputs or iterative dialogue histories. Existing generative AI platforms (e.g., ChatGPT, Microsoft Co-Pilot, Google Gemini) store chat logs and generated files, however, they lack robust mechanisms to manage branching workflows, merge parallel conversation threads, or revert selectively to earlier states. Versioning of AI models has been discussed in prior work (Vadlapati, 2024), but to our knowledge no system explicitly supports full-versioning of generative AI interactions alongside the content they create—particularly when multiple AI assistants are used in parallel. This gap often forces users to adopt fragmented workflows, where they manually copy AI outputs, store them in external tools, and struggle to piece together a coherent project history.

In this paper, we address this gap by introducing a novel version control approach that captures both AI-driven conversations and their resulting artefacts within a single, integrated framework. Specifically, our work makes the following contributions.

- We identify the core challenges for an integration of AI assistants into iterative, multimodal workflows. From these observations, we derive the requirements for an AI-focused VCS.
- We propose a new VCS that treats every user prompt, AI response, and generated artefact as part of a unified version history, enabling branching, merging, and reverting at both the project and conversation levels while offering end-to-end traceability for each artefact created.
- We demonstrate the system's practicality through a data science application that generates synthetic datasets using GOOSVC as underlying VCS.

The remainder of this paper is organized as follows. Section 2 introduces a detailed use case to motivate the requirements for versioning generative AI workflows. We then review related work in generative AI interfaces and traditional VCSs in Section 3, highlighting their limitations. Section 4 describes our proposed system's architecture, data model, and merging strategies. Section 5 showcases a real-world demonstration of our approach in synthetic dataset creation, and finally Section 6 provides concluding remarks and outlines directions for future work.

2 Content Creation Use Case

Consider a creative director tasked with producing an advertisement clip for a new, innovative product. The process begins by defining a target persona and mapping out their journey, capturing key emotional touchpoints and decision-making moments. Next, the director articulates the product's value proposition and envisions how it can transform the persona's experience, weaving these elements into a compelling narrative. From that, multiple iterations of storylines are developed and refined—from initial concepts to detailed scene descriptions. The final storyboard emerges as a composite artefact that combines descriptive texts with illustrative images and may also include audio or video elements for more immersive storytelling.

We assume that the creative director uses an authoring tool for this task. In principle, it is possible to use AI assistants for each of these work steps. For example, a large language model can help brainstorm ideas for the persona and the journey, propose story lines and a multimodal AI can generate images and text for the storyboard. Additionally, specialized generative models can produce audio or video prototypes. Based on this use case, we will now identify typical procedures and derive the requirements that are placed on an underlying VCS used by the authoring tool.

2.1 Iterative Development of Artefacts

In creative workflows, it is common to iteratively develop artefacts. For instance, the creative director may want to refine persona sketches in multiple iterations. When using an authoring tool offering AI assistance for this task, the director may need to adjust the prompts to elicit more detailed responses or to clear up misunderstandings. When the authoring tool wants to send these prompt to a chat-based assistant, such as ChatGPT via API, any request must include the complete chat history. Therefore, the chat history for every generated artefact must be stored. To support this, a VCS must:

R1 Provide a mechanism to version artefacts together with their chat history.

2.2 Using Multiple Assistants

In creative workflows, it is common to use multiple AI assistants. These assistants may be used independently of each other or in a collaborative way. For instance, the creative director may use a large language model to develop a storyline in collaboration with a multimodal AI to generate images for the storyboard. Or they may use multiple instances of the same AI model with different roles to investigate different perspectives like the view of the customer and the view of the service provider onto the product.

To support iterative development with multiple assistants, the authoring tool must store the chat history and the generated artefacts for every assistant separately. This is necessary to keep the chat context clean for every assistant and to prevents unintended cross-contamination of different chat contexts. Furthermore, as long as there is no interaction between the assistants, using separate contexts allows to revert one chat to a previous state without affecting the others. However, to allow collaborative use of assistants, it must be possible to share artefacts between the contexts. To support this, a VCS must:

R2 Provide a mechanism to create and manage chat contexts for multiple assistants that contain artefacts and chat histories.

R3 Provide a mechanism to share artefacts between chat contexts.

2.3 Reverting to Previous Versions

In creative workflows, it is common to revert to previous versions to revise decisions or to correct mistakes. For instance, the creative director may want to revert to an earlier persona sketch or revisit a previous storyboard to incorporate a discarded scene. For this task, an AI-assisted authoring tool should support two types of revert: reverting the complete project to a previous state and reverting a single chat to a previous state. When reverting a project, all artifacts and the related chats must be reverted. When reverting a single chat, only artifacts

generated in this chat must be reverted. However, the causality must be maintained between the chat and the project. For instance, if an artefact is reverted that has been used elsewhere in the project. To support this, a VCS must:

R4 Provide a mechanism to revert a project to any previous version including all chats and their artefacts.

R5 Provide a mechanism to revert a chat including the generated artifacts to any previous version while preserving the causality between the chat and the project.

2.4 Creating Variants of Workflows

In creative workflows, it is common to create multiple variants to explore different ideas or evaluate the impact of changes. For instance, the creative director may want to generate several versions of a storyboard to compare different visual styles or experiment with alternative personas. When using chat-based assistants via the provided API, the authoring tool must keep track of alternative paths because any request must include the complete chat history. To support this, a VCS must:

R6 Provide a mechanism to start alternative paths from any previous version including all artefacts and their associated chat histories without losing the progress made so far.

2.5 Combining Parallel Workflows

It is common in creative workflows to parallelize work on different parts of a project to increase efficiency. For instance, the creative director may want to distribute the work on different parts of the storyboard among several team members. To get the final storyboard, the authoring tool needs to combine the results of parallel workflows. During this merge, two types of conflicts may arise: Conflicts on artefacts occur when the same artefact is changed in multiple branches. Conflicts on chat histories occur when the same chat was continued in multiple branches. Furthermore, these merges will often include more than two branches. To support this, a VCS must:

R7 Provide a mechanism to merge any number of parallel workflows including their artefacts and the associated chat histories offering methods to resolve conflicts on artefacts and on chats.

2.6 Defining Stages in Workflows

In creative workflows, it is common to define stages to structure the creative process. For example, the creative director may want to define stages for the definition of the personas or the definition of their journeys. Stages help simplify to revert to defined milestones, to create variants and to parallelize workflows. To achieve that, an authoring tool must guarantee that the stages are unique at any time within the project history. This must be ensured when stages are added but also when parallel workflows are merged. To support this, a VCS must:

R8 Provide a mechanism to define stages in the version history and to keep these stages unique with the project history also when merging parallel workflows.

2.7 Summary

These procedures derived from our use case reveal several critical challenges for a version control system that is used by an authoring tool for creative workflows. First, there is an urgent need for **dual versioning with contexts** (**R1**, **R2**, **R3**) that maintains a direct link between evolving artefacts and the underlying AI-driven conversations that produce them. Second, the ability to **revert projects and chats** (**R4**, **R5**) is crucial for revising decisions and recovering from mistakes. When reverting single chats, the system must maintain causality between the chat and the project. Third, the iterative nature of creative work requires robust **automatic branching** (**R6**) to explore alternatives without losing previous progress. Fourth, while parallelization can enhance efficiency in creative workflows, **merging parallel paths** (**R7**) requires a mechanism to combine multiple branches and resolve conflicts on artefacts and chats. Finally, the ability to define and maintain **project stages** (**R8**) in the creative process is essential for structuring complex workflows and project planning.

Every requirement above targets a distinct dimension of AI-assisted content creation, ensuring coverage of iterative development, collaboration among multiple assistants, safe reversion, parallel exploration, and structured milestone definition. Together, they form an orthogonal set that comprehensively addresses the challenges identified in this use case. Overall, these requirements address the nonlinear, multimodal, and iterative nature of generative AI use cases introduced in Sect. 1.

3 Background

In this section, we first assess how existing generative AI tools and interfaces manage iterative creative workflows and highlight their current limitations in terms of traceability and prompt reuse. We then turn to VCS as a potential source for more advanced branching and merging concepts. We go on to evaluate how well these approaches fulfil the requirements described in section 2, and finally identify the key gaps that motivate our solution.

3.1 Generative AI Interfaces

Contemporary generative AI front-ends such as ChatGPT, Microsoft Co-Pilot, Google Gemini and Anthropic Claude have transformed content creation by delivering multimodal outputs and allowing users to refine prompts on the fly. Despite these strengths, they offer limited support for complex, iterative workflows that require branching, merging and robust versioning. While some interfaces allow users to revisit or modify previous prompts, each output is still treated as an isolated event, and complex artefacts or composite data sets are not intrinsically linked to the conversation history that produced them. This missing link makes end-to-end traceability difficult: although users can see the final product, they have no systematic way of exploring the creative process that led to that outcome.

Popular AI front-ends often confine interactions to a single chat context, making it difficult to collaborate across multiple AI models or to run parallel explorations of the same artefact. In particular, if AI models from different providers are involved, their interaction cannot be documented. The functions available within the web interface, such as storing or editing a chat history especially from different models, are vendor-specific, manual processes. When these models are invoked programmatically via an API, the application developer must transmit the entire conversation history again for every request and in addition manage the branching logic. As a result, key requirements such as **R6** (starting alternative paths), **R7** (merging parallel paths), **R8** (defining stages in workflow), and **R1** (versioning artefacts with their chat history) remain uncovered. Consequently, creators either do without the possibility of branching and merging or resort to inefficient workarounds such as copying intermediate outputs, duplicating prompts, and manually tracking versions outside the AI tool.

In summary, while modern generative AI systems excel at generating rich content, they offer minimal native support for iterative, branching workflows. This limitation hinders the kind of controlled exploration and traceability that creators increasingly need when integrating AI into complex projects.

3.2 Version Control Systems

VCSs have long been essential for tracking and managing changes across software projects and other text-based repositories. Traditional systems, such as Subversion and Git, typically provide several core capabilities. First, they maintain a chronological sequence of changes, known as linear versioning, which preserves a historical record of modifications. Second, they allow branching, so that work can proceed in parallel lines of development, making it possible to explore experimental features or maintain distinct configurations. Third, these systems include merging functionality, enabling divergent branches to be reconciled into a unified project state. Finally, they allow projects to maintain different variants through mechanisms that can track concurrent releases or alternate product lines.

While these mechanisms provide a solid blueprint for managing project histories, they were never designed to track interactive conversations or dynamically generated content from AI models. Such conversations must be versioned in a manner similar to code commits, yet cannot be handled by line-based diffs. Conventional VCSs distinguish between text and binary files, both of which are inadequate for storing conversational histories. This is because requirements such as **R1** (versioning artefacts with their chat history), **R2** (managing multiple chat contexts), **R3** (sharing artefacts between chat contexts), and **R5** (reverting selected chats while preserving causality) are not met.

In light of these gaps, where generative AI tools lack integrated branching, merging, and revert capabilities, and where conventional VCS fail to capture conversational histories, we propose a specialised versioning framework tailored to AI-driven creative workflows. Our approach unifies conversation and artefacts tracking, addresses branching across multiple assistants, and integrates robust merging mechanisms, laying the groundwork for end-to-end traceability and prompt reuse in generative AI projects.

4 Versioning Approach

In this section, we present our new versioning approach and show how the requirements listed in 2.7 are addressed. Our implementation GOOSVC (Grünert, 2025) is designed to be used in a production environment considering operational aspects such as performance, scalability, and security. As shown in Figure 1, users will typically not interact directly with GOOSVC. The goal is to simplify the integration of AI assistants into workflows, enabeling seamless branching, reverting, and merging, and to offer traceability across both the prompts and the generated content.

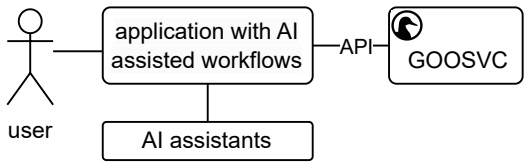

Figure 1: System overview

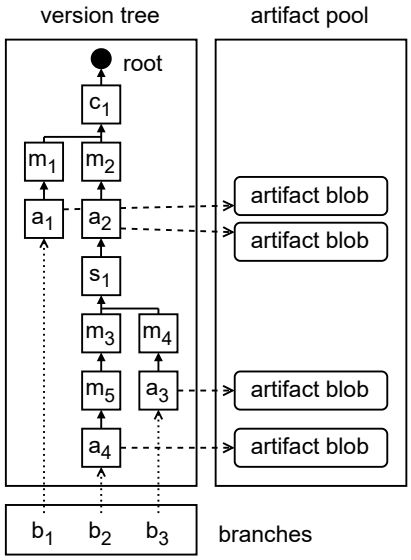

Figure 2: Data model example showing a version tree with three branches (b_1 to b_3) containing chat- (c), message- (m), stage- (s) and artifact-nodes (a) with references to the artifact pool.

Figure 2 presents the core elements of the data model. It consist of an immutable node-based version tree, where every interaction is recorded as a distinct node with associated metadata (e.g., parent node, time stamp, author, committer, and type specific data), an immutable pool for storing generated artifacts, and branches to keep track of all available paths. Nodes have one of the following types:

- **Chat Nodes**: Add a new chat context to the project for interactions with an AI. Every chat context has a unique ID later used by messages and artifacts to declare their affiliation. Chats can be started either from scratch or with a parent by referring to a message node. Chats with parents are interpreted as a continuation from the referenced message.
- **Message Nodes**: Store prompt-response pairs that capture the conversational exchange with an AI. All messages must be associated with a chat context.
- **Artifact Nodes**: Store artifacts (text, images, audio) by referencing the actual data in the artifact pool and defining metadata such as the filename. All artifacts must be associated with a chat context.
- **Stage Nodes**: Define named project milestones. Stage names must be unique within any path of the version tree.
- **Merge Nodes**: Document a merge operation of parallel branches. Merge nodes are not shown in Figure 2. Details are described in section 4.4.

4.1 Dual Versioning with Contexts

When inserting nodes into the version tree, the position of the new node must be defined. This position can either be a branch or an existing node. When a branch is used, the new node is appended to the branch. When a node is used, the system inserts the new node as a child of the given node. Depending on the parent's position, this will either create a new branch or continue a branch (see 4.3).

The system establishes a link between AI interactions and their generated artifacts by capturing both within the same version tree. All prompt-response pairs are stored as message nodes in the version tree, referencing a chat context and thereby forming a complete lineage of the conversational history. Similarly, every artifact is represented as an artifact node, referencing both the chat context used to create it and the artifact in the pool. When storing or changing artifacts, the system adds a new artifact node containing all metadata such as file name, path, scope (chat, global) and the operation (add, update, rename, delete). Artifacts in the pool are immutable, therefore, when artifacts are added or updated, a new artifact is added to the pool.

Every node of the version tree represents a version of the project. This version is defined as the union of all nodes along the path from that node back to the root. When retrieving a selected project version, the system can either return the complete project or only data from a specific chat context. If the path contains multiple artifacts with the same combination of path and filename, the latest node masks all older ones. For any version of a project, only one artifact for a given path-filename combination is visible. If the last node for a given path-filename combination has the operation delete, there is no such artifact in the respective version. Path and filename are both defined by the workflow application. Similar to popular VSC, The complete set of all artifacts can be checked out to the local file system for any version of the project.

While messages are always limited to one chat context, the scope of artifacts can be set. If scope is set to *chat*, the artifact is only visible within the chat context. If set to *global*, the artifact is visible in all chats. Setting the scope of an artifact to global allows to share artifacts between chats. This dual versioning approach with contexts ensures that every creative decision is fully traceable, enabling users to audit the entire workflow and understand the context behind every artifact generated within the project.

4.2 Revert Projects and Chats

Our system supports two types of revert: reverting the entire project to a previous state and reverting a single chat to a previous state. When reverting a single chat, only the artifacts within the chat's context and their associated chat history are reverted. In contrast, reverting a project to a previous state will revert all chats and their artifacts.

Reverting a single chat is achieved by inserting a new chat node that references the previous state as parent. New messages added to the chat after the revert must then be associated with this new chat. Figure 3 shows an example of reverting a chat. Subfigure a) shows the project before the revert containing two chats: c_1 (m_1, m_3) and c_2 (m_2, m_4). Subfigure b) shows the project after reverting the chat c_1 to message m_1 and adding an additional message m_5. Reverting is achieved by adding c_3, referencing m_1 as parent. The chat is then extended with message m_5. After this revert, the project contains three chats: c_1 (m_1, m_3), c_2 (m_2, m_4), and c_3 (m_1, m_5).

Reverting a project is achieved by branching off from the previous state. The new branch is then used to continue the project. Figure 3 Subfigure c) shows the project after reverting the entire project to message m_1 and adding the additional message m_5 to chat c_1. The project is branched off from m_1, and c_1 is extended with m_5. The project in branch b_2 contains chat c_1 with the messages m_1 and m_5.

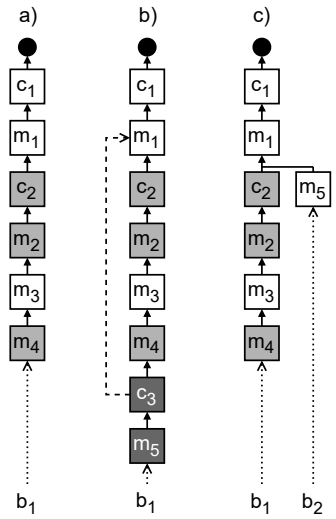

Figure 3: a) Original project with two chats (c_1, c_2). b) Project after reverting **chat** c_1 to m_1 and adding additional message m_5. c) Project after reverting **project** to m_1 and adding additional message m_5. Nodes with the same background color belong to the same chat context.

4.3 Automatic Branching

In our system, branching is used for three different purposes: Variants, reverting, and parallelization. Variants are used to explore different ideas or evaluate the impact of changes. Finally, one variant is selected to continue with. Reverting is used to go back to a previous state and continue working from there by branching off. Parallelization is used to increase efficiency by working simultaneously on different parts of a project. The results of parallel workflows are merged to create the final output (see 4.4).

The creation of these branches is not always a conscious decision. Often, they emerge naturally as the creative process unfolds. To capture this organic branching, our system automatically creates new branches whenever a node diverges from an existing path. Instead of using names for branches, the system uses unique identifiers. These identifiers are used to reference a branch when appending nodes or merging branches.

4.4 Merging Parallel Paths

Merging creative workflows is used to combine the work of a parallelized sections of a project. Such a merge may include more than two branches. Furthermore, the merge does not necessarily include the complete branch up to its head. In the example shown in Figure 4, branch b_4 is merged with n_8 from b_2 and n_6 from b_3.

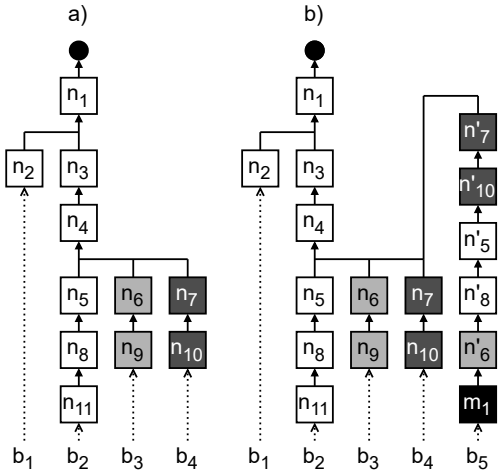

Figure 4: Example of a conflict-free merge with nodes n of unspecified type. a) Original project before the merge. b) Project after merging b_4, n_8 and n_6. Nodes with the same background color belong to the same branch before the merge.

Merging is achieved by replaying all nodes that follow the first common ancestor into a new branch to create a unified, sequential history. At the end of this sequence, an additional merge node is added to document the merge. In the example shown in Figure 4, branch b_5 contains this sequence and the additional merge node (m_1).

As introduced in 2.5, two types of conflicts my arise when merging parallel paths: Chat conflicts, when the same chat was continued in multiple branches, and artifact conflicts, when the same artifact was changed in multiple branches. Chat conflicts are resolved as follows: the system splits the dialogue automatically into two chats with a shared history before the first common ancestor. This approach ensures that the context of every conversation remains intact, even when the content diverges. In the example shown in Figure 5, branches b_2 and b_3 both continued the chat c_1. To resolve the conflict when merging b_2 and b_3, the system creates a new chat (c_2) with the common history of c_1 before the divergence. Message m'_4 and artifact a'_3 are then both added to c_2.

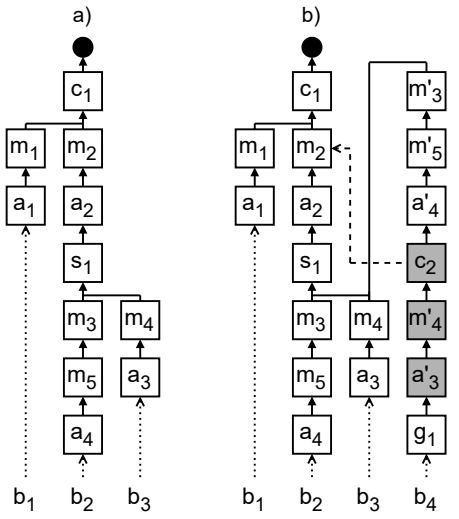

Figure 5: Example of a merge with conflicting chats a) Version tree before the merge. b) Version tree after merging b_2 and b_3. Nodes with the same background color belong to the same chat context.

The resolution of artifact conflicts depends on the file type. For text files, the system may automatically merge the changes, if independent sections were modified. For binary files, the system will rename the files. Both cases may require manual review of the result. In general, merge conflicts on artifacts should be avoided. For most situations, parallelized work only makes sense if the work is independent.

Thus, the merging mechanism effectively integrates parallel creative paths while resolving conflicts automatically. Instead of relying on standard diff-based methods, our approach tailors conflict resolution to the nature of the content, ensuring that the creative process remains fluid and efficient.

4.5 Project Stages

Stages function as immutable checkpoints within the project history. Therefore, stages are not associated with a chat context. Every stage marks a milestone that remains unchanged regardless of subsequent iterations, offering stable reference for reverting or branching the project. Stages are implemented as stage nodes in the version tree. The system ensures that stages are unique within any path of the version tree. The system refuses to add a stage if the name already exists. Also when merging parallel workflows, stages must be kept unique. To achieve this, the system refuses to merge branches containing any stages. Stages represent milestones for the entire project. Adding a

stage via a merge would contradict this concept. In summary, stages provide fixed anchors in the creative process, ensuring that pivotal moments remain preserved and clearly defined.

5 Demonstration

To demonstrate the flexibility and real-world utility of our version control approach, we applied it to a complex workflow that generates synthetic crime datasets for research. Data recorded during criminal investigations is often confidential and therefore unavailable for research. Existing datasets from other domains do not share the characteristics of crime-related data, which typically include telephone recordings, audio surveillance with varying quality, multilingual and emotional speech, and background noise containing relevant information. Moreover, higher-level analyses such as communication structure detection require the spoken content to match the context of actual criminal cases.

To address these challenges, we presented a workflow in (Grünert et al., 2024) that generates synthetic datasets from a case outline (see Figure 6). Specifically, it uses LLMs to produce transcripts of conversations and messages related to a hypothetical criminal case. This involves 22 different prompt templates and over 400 individual requests to LLMs. Next, these transcripts are annotated with emotions and timing aspects and then converted to audio files. Background noise and signal processing are subsequently applied to create realistic acoustic variations. The final dataset comprises text messages, audio files, and annotations (RTTM, TextGrid), making it suitable for research on speech analysis or communication structures.

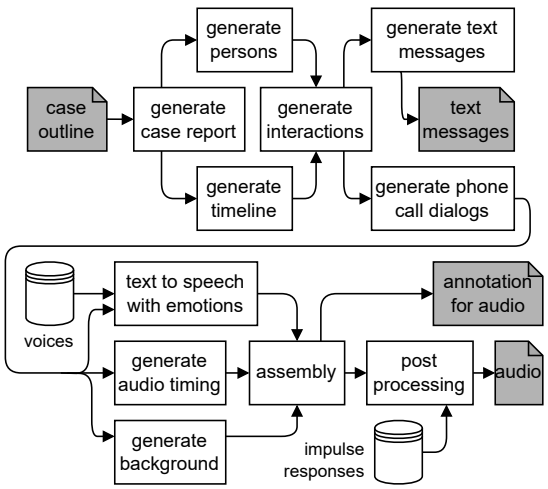

Figure 6: Case generation pipeline

Using GOOSVC, we developed an interactive web application that orchestrates every phase of this workflow while providing robust version control capabilities. One key advantage is the ability to manage distinct stages in the dataset generation process. Users can revert to any prior milestone (**R4**, **R5**) and make adjustments without having to restart the entire pipeline. For instance, if a user wants to revise how two suspects interact in the transcripts or modify the background noise level, they can branch off from the relevant stage, edit just the targeted prompts and parameters, and then regenerate only the affected outputs—preserving all other completed work.

Furthermore, for every artifact created, the system automatically stores the associated AI interactions. This provides end-to-end traceability (**R1**), allowing users to see which prompts and responses led to a specific audio track, transcript, or annotation. The same approach also supports branching out (**R6**) into parallel workflows—such as exploring different emotional tones for conversation—before merging them (**R7**), if needed. As a result, workflows deemed as successfull can be conveniently reused and adapted for new case outlines, emphasizing the flexible and iterative nature of GOOSVC.

6 Conclusion

In conclusion, we have introduced a novel version control approach tailored for generative AI-driven content creation. Our VCS captures both AI interactions and the resulting artifacts in a unified system offering branching, merging, and stable milestones. By addressing the key challenges of iterative creative workflows—such as maintaining traceability, managing parallel explorations, and resolving content-specific conflicts—our approach offers a robust framework that enhances reproducibility and flexibility. This work not only streamlines the creative process but also lays the groundwork for future enhancements in collaborative AI-driven design and content management.

References

Coca-Cola. 2023. Coca-cola invites digital artists to 'create real magic' using new ai platform.

Cygnis. 2024. Best practices for implementing ai workflow automation in enterprises.

Thomas H. Davenport and Nitin Mittal. 2022. How generative ai is changing creative work.

European-Commission. 2020. White paper on artificial intelligence-a european approach to excellence and trust.

David Grünert. 2025. Goosvc github repository. https://github.com/goosvc/goosvc.

David Grünert, Dominic Pfister, Alexandre de Spindler, and Volker Dellwo. 2024. Generating synthetic datasets for the validation and training of automatic speech analysis systems in the context of organized crime. 2nd VoiceID conference, Marbug, Germany.

Dinesh Kumar and Nidhi Suthar. 2024. Ethical and legal challenges of ai in marketing: an exploration of solutions. *Journal of Information, Communication and Ethics in Society*, 22(1):124–144.

Jack McGuire, David De Cremer, and Tim Van de Cruys. 2024. Establishing the importance of co-creation and self-efficacy in creative collaboration with artificial intelligence. *Scientific Reports*, 14(1):18525.

Jeba Rezwana and Mary Lou Maher. 2023. Designing creative ai partners with cofi: A framework for modeling interaction in human-ai co-creative systems. *ACM Transactions on Computer-Human Interaction*, 30(5):1–28.

Praneeth Vadlapati. 2024. Updagent: Ai agent version control framework for real-time updation of tools. *International Journal of Science and Research (IJSR*, 13(11):628–632.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.