HAW HAMBURG

BACHELORARBEIT

Automatisierung der KI-basierten Generierung von audiovisuellen Medieninhalten

vorgelegt am 26. März 2026
Alexandre Kaul

Erstprüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Simon Dewert

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Fakultät Informatik und Digitale Gesellschaft
Finkenau 35
20081 Hamburg

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Frage, wie die KI-basierte Generierung von Sprechervideos durch ein modulares Websystem automatisiert und reproduzierbar gemacht werden kann. Ausgangspunkt war ein konkreter Bedarf aus einem Forschungsprojekt an der HAW Hamburg, in dem große Mengen an Nachrichtenvideos mit unterschiedlichem politischem Framing erzeugt werden sollten.

Das Problem lag dabei nicht im Fehlen geeigneter KI-Modelle, sondern in deren Orchestrierung. Das entwickelte System verbindet ein Node.js/Express.js-Backend mit ComfyUI als KI-Backend und mit einem webbasierten Frontend. Über eine REST-API können Jobs eingereicht und automatisch in einer Warteschlange verarbeitet werden. Für die Sprachsynthese wurden zwei Open-Source Text-to-Speech Modelle mit Voice-Cloning-Funktion verglichen und integriert, für die Videoanimation ein aktuelles Talking-Head-Modell eingesetzt. Eine optionale Nachbearbeitung mit FFmpeg ermöglicht das Hinzufügen von Intros und Wasserzeichen.

Abstract

This bachelor's thesis examines how the AI-based generation of news anchor videos can be automated and made reproducible using a modular web system. The starting point was a specific need arising from a research project at HAW Hamburg, in which large quantities of news videos with varying political framing were to be generated.

The problem was not the lack of suitable AI models, but in their orchestration. The developed system combines a Node.js/Express.js backend with ComfyUI as the AI backend and a web-based frontend. Jobs can be submitted via a REST API and automatically processed in a queue. For speech synthesis, two open-source text-to-speech models with a voice cloning feature were compared and integrated; for video animation, a current talking head model was used. Optional post-processing with FFmpeg allows for the addition of intros and watermarks.

Inhaltsverzeichnis

Verwendete Hilfsmittel

Im Rahmen dieser Bachelorarbeit habe ich „KI“ (ChatGPT, Claude) zur Korrektur von Rechtschreibfehlern sowie zur vereinzelten Verbesserung des Satzbaus genutzt. Die inhaltliche Ausarbeitung, wissenschaftliche Analyse und die vollständige Konzeption und Implementierung des Systems wurden ausschließlich von mir selbst durchgeführt.

Abkürzungsverzeichnis

API	Application Programming Interface
ASR	Automatic Speech Recognition
CLI	Command Line Interface
EJS	Embedded JavaScript
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
KI	Künstliche Intelligenz
LTS	Long Term Support
REST	Representational State Transfer
SQL	Structured Query Language
TTS	Text-to-Speech
UUID	Universally Unique Identifier
URL	Uniform Resource Locator
VRAM	Video Random Access Memory
WAV	Waveform Audio File Format
WSL	Windows Subsystem for Linux

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der Auslöser für diese Arbeit war ein konkretes praktisches Problem eines Doktoranden an der HAW Hamburg. Dieser entwickelt ein System zur automatischen Umformulierung von Nachrichtentexten mit unterschiedlichem politischem Framing. Unter politischem Framing versteht man die selektive Hervorhebung bestimmter Aspekte, um die subjektive Wahrnehmung zu einem Thema zu beeinflussen (Entman, 1993). Um die Wirkung verschiedener Framings nicht nur textlich, sondern auch audiovisuell zu untersuchen, sollten synthetische (Nachrichten-) Videos mit einem virtuellen Sprecher erzeugt werden und zwar in großer Stückzahl und reproduzierbar.

Die Herausforderung bestand dabei nicht in fehlenden KI-Technologien. Für Sprachsynthese (z.B. XTTS (Edresson Casanova, 2024)), Gesichtsanimation (z.B. SadTalker (Wenxuan Zhang, 2022)) und Videobearbeitung (z.B. FFmpeg[1]) stehen bereits Open-Source-Modelle bzw. Werkzeuge zur Verfügung. Das Problem liegt in der Orchestrierung. Ein manueller Workflow, der den parallelen Einsatz mehrerer KI-Modelle erfordert, ist für die Generierung einer hohen Anzahl von Videovarianten nicht praktikabel. Ein automatisierter, API-gesteuerter Workflow fehlte.

Kommerzielle Anbieter schieden aus praktischen Gründen aus. Die Nutzungsbedingungen schränken experimentelle Inhalte ein, die Kosten skalieren mit der Videomenge, und der Datenschutz ist bei externen Diensten nicht immer gewährleistet. Daraus entstand die Entscheidung, ein eigenes, lokal betriebenes System zu entwickeln, mit vollständiger Kontrolle über Modelle, Konfiguration und Ausgaben.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Konzeption, Implementierung und Evaluation eines Websystems, das die KI-basierte Generierung audiovisueller Medieninhalte automatisiert und reproduzierbar macht. Das System soll die technische Komplexität der zugrundeliegenden KI-Pipeline für die Nutzende verbergen und eine klare Schnittstelle für die Einreichung, Überwachung und den Abruf von Generierungsjobs bereitstellen.

Konkret wurde ein System entwickelt, das aus einem Node.js/Express.js Backend, einem webbasierten Frontend sowie einer Integration mit ComfyUI[2] als KI-Backend besteht. Über eine API können Nutzende Texte zusammen mit einem Sprecherprofil und optionalen Einstellungen übermitteln. Das System erstellt daraus vollautomatisch ein KI-generiertes Video und stellt dieses zum Download bereit.

Ein weiteres Ziel ist die Modularität des Systems. Durch eine klare Trennung von Frontend, Backend, Konfiguration und KI-System soll es möglich sein, das System später ohne großen Aufwand anzupassen, etwa durch das Hinzufügen neuer Sprecher, dem Wechsel der TTS-Engine und die Anpassung der

[1] https://www.ffmpeg.org/ (Letzter Zugriff: 09.03.2026)
[2] https://www.comfy.org/ (Letzter Zugriff: 12.03.2026)

Nachbearbeitung. Das Frontend ist in diesem Projekt als Beispielimplementierung gedacht, die die Nutzung der REST-API veranschaulicht. Für das eigentliche Forschungsprojekt wird das Frontend nicht benötigt, da dort die API direkt angesprochen wird.

1.2 Leitfrage der Arbeit

Ausgehend von der beschriebenen Motivation und Problemstellung ergibt sich folgende Leitfrage dieser Arbeit:

> „Wie kann die KI-basierte Generierung audiovisueller Medieninhalte durch ein modulares Websystem automatisiert und reproduzierbar gemacht werden?“

Zur Beantwortung dieser Frage wird ein Prototyp entwickelt, der verschiedene KI-Modelle zur Sprach- und Videogenerierung miteinander verbindet und über eine Web-API orchestriert.

Im Verlauf der Arbeit werden sowohl die Architektur des Systems als auch die einzelnen Verarbeitungsschritte analysiert und implementiert.

2 Technischer Hintergrund

In diesem Kapitel werden die technologischen Grundlagen erläutert, auf denen das System aufbaut. Dazu gehören die verwendeten KI-Modelle für Sprachsynthese und Videogenerierung, das ComfyUI Workflow-System sowie die eingesetzten Webtechnologien für das Backend.

2.1 Text-to-Speech und Voice Cloning

Text-to-Speech, kurz TTS, bezeichnet die automatische Umwandlung von geschriebenem Text in gesprochene Sprache. Wer frühere Navigationssysteme kennt, hat eine Vorstellung davon, wie solche Systeme klingen können, maschinell, monoton, und mit einer Aussprache, die man schnell als synthetisch erkennt. Neuere, auf Deep Learning basierende Ansätze haben das grundlegend verändert. (Xu Tan, 2021)

Systeme wie FastSpeech (Yi Ren, 2019) nutzen Transformer-Architekturen für eine schnelle, parallelisierbare Synthese, während autoregressive Ansätze wie Tacotron 2 (Jonathan Shen, 2017) durch ihre schrittweise Verarbeitung natürlichere Betonung erreichen, auf Kosten von mehr Rechenaufwand.

In einer Übersichtsarbeit zeigen Xu Tan et al. (2021), dass neuronale TTS-Systeme in den letzten Jahren erhebliche Fortschritte erzielt haben und die Qualität synthetischer Sprache deutlich verbessert wurde.

Für dieses Projekt war dabei vor allem die Funktion des Voice-Cloning der modernen neuronalen Systeme entscheidend. Damit bezeichnet man die Fähigkeit, aus einer kurzen Referenzaufnahme die charakteristischen Merkmale einer Stimme zu extrahieren und beliebige neue Texte in eben dieser Stimme zu synthetisieren (Hussam Azzuni, 2025). Der praktische Vorteil liegt darin, dass für einen neuen Sprecher keine langen Aufnahmen und kein separates Training nötig sind. Hierbei reichen wenige Sekunden bis maximal eine halbe Minute an Audiomaterial, um eine Stimme zu klonen, die dann ähnlich zum Original ist.

Im Rahmen der Arbeit wurden konkret zwei Open-Source TTS Systeme verglichen und in das System integriert: Chatterbox TTS (Resemble AI, 2025) und Qwen3-TTS (Qwen Team, Alibaba Cloud, 2026). Beide unterstützen Voice Cloning auf Basis kurzer Audiobeispiele und lassen sich über ComfyUI einbinden. Ein wesentlicher praktischer Unterschied ist, dass Chatterbox neben der Referenzaudiodatei auch ein schriftliches Transkript des gesprochenen Referenztextes erfordert. Qwen3-TTS führt diese Speech-to-Text-Erkennung mithilfe von ASR („Automatic Speech Recognition“) intern durch und kommt ohne explizites Transkript aus, was die Einrichtung eines neuen Sprechers vereinfacht. In subjektiven Hörtests klang Qwen3-TTS insgesamt natürlicher, bei sehr langen Texten neigt das Modell allerdings gelegentlich zu einem leicht erhöhten Sprechtempo mit sinkender Audiolautstärke. Chatterbox war in diesen Fällen konsistenter, klingt aber in der Grundqualität etwas synthetischer und kompri-

mierter. Für das Forschungsprojekt, in dem es auf gleichbleibende und reproduzierbare Ausgaben ankommt, spielt diese Abwägung eine Rolle, die bei der Auswahl des Workflows berücksichtigt werden sollte.

2.2 KI-basierte Videogenerierung

Neben der Sprachsynthese ist die Generierung eines passenden Videos ein weiterer wichtiger Schritt in der Pipeline. Es gibt dabei verschiedene Ansätze, wie KI-Modelle zur Videoerzeugung eingesetzt werden können.

Ein Ansatz sind sogenannte Text-to-Video-Modelle, die direkt aus einer Textbeschreibung ein Video generieren. Neuere Text-to-Video-Modelle kombinieren dabei häufig Diffusionsmodelle mit Transformer-Architekturen, um zeitlich kohärente Videosequenzen zu erzeugen. Allerdings sind diese häufig stark in der Länge der Videos begrenzt und lassen sich schlecht auf einem bestimmten Sprecher festlegen (Singh, 2023; Zhuoyi Yang, 2024). Diese Technik wird u.a. in Sora[3] von OpenAI eingesetzt.

Für diese Arbeit wurde deshalb ein anderer Ansatz gewählt. Die Verwendung von sogenannten Talking-Head-Modellen. Diese Modelle animieren ein vorhandenes Referenzbild oder Video anhand eines Audiosignals. Auf Basis des Referenzbildes und der Tonspur werden Lippenbewegungen, Gesichtsausdrücke sowie grobe Körperbewegungen erzeugt. Das Ergebnis ist ein Video, in dem ein virtueller Sprecher den erzeugten Text bzw. Ton in einer realistischen Weise wiedergibt.

Konkret wird das Modell Infinite Talk verwendet, das auf dem Wan 2.1-Basismodell aufbaut (Shaoshu Yang, 2025; Team Wan, 2025). Die Wahl fiel auf dieses Modell, weil es sich in ComfyUI integrieren lässt, mit der verfügbaren Hardware betrieben werden kann und in den durchgeführten Praxistests für den vorliegenden Anwendungsfall brauchbare Ergebnisse geliefert hat.

2.3 ComfyUI als Workflow-System

Um mehrere KI-Modelle in einer automatisierten Pipeline zu kombinieren, braucht man eine Möglichkeit, deren Verarbeitung zu steuern. ComfyUI hat sich in diesem Projekt als praktisches Werkzeug bewährt. Es handelt sich um ein quelloffenes, node-basiertes Workflow-System, das ursprünglich für Stable-Diffusion Anwendungen entwickelt wurde, sich aber inzwischen für eine Vielzahl von KI-Modellen nutzen lässt. Die Grundidee ist eine visuelle Programmieroberfläche, bei der einzelne Verarbeitungsschritte als sogenannte Nodes dargestellt werden, die miteinander verbunden werden. Jede Node übernimmt eine klar definierte Aufgabe, etwa ein Modell laden, eine Audiodatei einlesen, Text übergeben oder ein Ergebnis speichern. Abbildung 1 zeigt exemplarisch den Aufbau eines TTS-Workflows, der Teil eines Templates von TTS-Audio-Suite[4] in ComfyUI ist.

[3] https://openai.com/de-DE/sora/ (Letzter Zugriff: 19.03.2026)
[4] https://github.com/diodiogod/TTS-Audio-Suite (Letzter Zugriff: 19.03.2026)

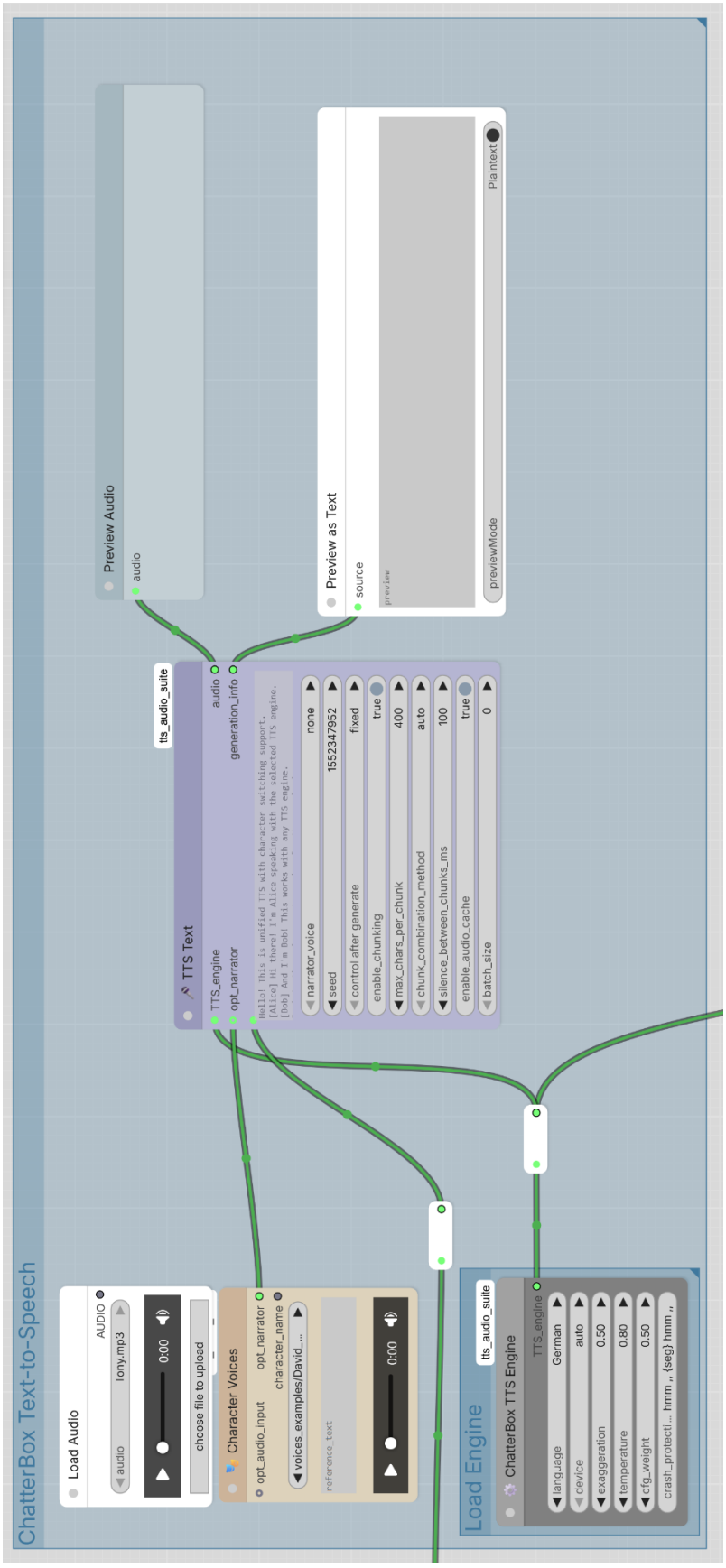

Abbildung 1: Screenshot eines ComfyUI Workflows für Chatterbox TTS auf Basis eines Templates von TTS-Audio-Suite. Quelle: Eigene Darstellung.

Was ComfyUI für dieses Projekt besonders geeignet macht, ist die HTTP-API. Workflows lassen sich als JSON exportieren und anschließend programmatisch über einen POST-Request einreichen. Das Backend dieser Arbeit nutzt genau diese Schnittstelle. Es lädt ein vorbereitetes Workflow-Template, befüllt die variablen Stellen wie Text, Stimme, Referenzbild, Auflösung, jobspezifischen Eingaben und übergibt das fertige JSON an ComfyUI. Über denselben Weg lässt sich anschließend der Verarbeitungsstand abfragen.

2.4 Webtechnologien für Backend

Das Backend des Systems ist als Webserver realisiert, der eine REST-API bereitstellt. REST ist ein weit verbreitetes Architekturmuster für Webschnittstellen, bei dem Ressourcen über HTTP-Endpunkte angesprochen werden (Fielding, 2000). Clients können über standardisierte HTTP-Methoden wie GET, POST oder DELETE auf Ressourcen zugreifen.

Als Laufzeitumgebung für das Backend wird Node.js[5] eingesetzt. Node.js ermöglicht die Erstellung serverseitiger Anwendungen in JavaScript und eignet sich gut für Anwendungen wie die hier entwickelte API. Als Framework wird Express.js[6] verwendet, da es eine einfache Definition von HTTP-Endpunkten erlaubt. Zur Speicherung der Generierungsjobs wird SQLite[7] eingesetzt. SQLite ist eine relationale Datenbank, die ohne separaten Datenbankserver auskommt und ihre Daten in einer einzigen lokalen Datei speichert. Das macht die Installation und den Betrieb des Systems deutlich einfacher als bei serverbasierten Datenbanken. Für die Nachbearbeitung der generierten Videos wird FFmpeg genutzt. FFmpeg ist ein weitverbreitetes Kommandozeilenprogramm zur Verarbeitung von Audio- und Videodaten. Im System wird es eingesetzt, um Intro-Videos und Wasserzeichen einzufügen.

[5] https://nodejs.org/ (Letzter Zugriff: 09.03.2026)
[6] https://expressjs.com/ (Letzter Zugriff: 09.03.2026)
[7] https://sqlite.org/ (Letzter Zugriff: 09.03.2026)

3 Verwandte Arbeiten

Bevor das eigene System konzipiert wurde, wurden bestehende Lösungen zur automatisierten Videogenerierung angeschaut. Dabei kann in zwei Kategorien unterschieden werden: kommerzielle Lösungen und quelloffene Projekte.

3.1 Kommerzielle Lösungen

Für die automatisierte Erstellung von Sprechervideos gibt es eine Reihe kommerzieller Dienste. Bekannte Anbieter sind Synthesia[8], HeyGen[9] und D-ID[10]. Diese Plattformen ermöglichen es, über eine Weboberfläche oder API einen Text einzugeben und daraus ein Video mit einem virtuellen Sprecher zu erzeugen.

Für den Einsatz in einem wissenschaftlichen Forschungsprojekt wurden diese Dienste jedoch nicht weiterverfolgt. Ausschlaggebend waren vor allem die externe Datenverarbeitung, laufende Nutzungskosten sowie die Abhängigkeit von anbieterspezifischen Nutzungsbedingungen. Für dieses Forschungsprojekt, in dem viele Varianten erzeugt werden sollen, ist daher ein lokal betreibbarer Open-Source Ansatz besser geeignet.

Auch Sora von OpenAI wurde betrachtet, scheidet jedoch aus, da es sich um ein allgemeines Text-to-Video-Modell handelt und sich eine konsistente Sprecher-Identität (z.B. ein Sprecher aus der Tagesschau) damit nicht zuverlässig umsetzen lässt.

3.2 Open Source Lösungen

Im Open Source Bereich existieren ebenfalls relevante Arbeiten. Wav2Lip (K R Prajwal, 2020) ist ein Projekt, welches die Lippen eines Sprechers in einem bestehenden Video mit einer neuen Audiodatei synchronisiert. Dies wird beispielsweise verwendet, um Inhalte in anderen Sprachen zu synchronisieren. SadTalker (Wenxuan Zhang, 2022) geht einen Schritt weiter und generiert zusätzlich zu Lippenbewegungen auch Kopfbewegungen, was realistischer wirkt.

Das in dieser Arbeit verwendete Modell Infinite Talk (Shaoshu Yang, 2025) baut auf dem Wan 2.1-Basismodell (Team Wan, 2025) auf und erreicht eine optisch bessere Qualität als ältere Ansätze.

[8] https://www.synthesia.io/de (Letzter Zugriff: 14.03.2026)
[9] https://www.heygen.com/de-de (Letzter Zugriff: 14.03.2026)
[10] https://www.d-id.com/ (Letzter Zugriff: 14.03.2026)

3.3 Vergleich

Tabelle 1 vergleicht die betrachteten Ansätze anhand der für das Projekt ausgewählten, wichtigsten Kriterien, darunter lokale Betreibbarkeit, Kostenstruktur und API-Anbindung. Die qualitativen Einschätzungen in der Tabelle, etwa zur Animationsqualität, beruhen dabei nicht auf einem standardisierten Benchmark, sondern auf dem subjektiven Eindruck der Demo-Videos der kommerziellen Anbieter und projektbezogene Praxistests.

Tabelle 1: Vergleich kommerzieller Dienste mit Open-Source-Modellen

Kriterium	Synthesia	HeyGen	D-ID	Sora (OpenAI)	Wav2Lip / SadTalker	Infinite Talk
Datenschutz	Extern, keine Kontrolle	Extern, keine Kontrolle	Extern, keine Kontrolle	Extern, keine Kontrolle	Extern oder Lokal	Lokal
Kosten	~79€ für 360 Minuten Video (Credit System)	~$29 / Monat oder Pay as you go (API)	~$18-300 / Monat für 16-300 Minuten Video	~$0.10 – $0.70 pro Sekunde	kostenlos lokal, Credit System online	kostenlos
Sprecher wählbar	Ja	Ja	Ja (limitiert 1-3)	Nein	Ja	Ja
Open Source	Nein	Nein	Nein	Nein	Ja	Ja
API-Integration	Ja	Ja	Ja	Ja	Ja (über ComfyUI), Nein online	Ja (über ComfyUI)
Animationsqualität	Sehr gut	Sehr gut	Sehr gut	Gut	Mittel	Gut
Nutzungsbedingungen	Einschränkend	Einschränkend	Einschränkend	Einschränkend	Frei lokal, Einschränkend online	Frei

Aus diesem Vergleich ergibt sich die Entscheidung für den eigenen Ansatz klar. Kommerzielle Dienste scheiden wegen Datenschutz, Kosten und einschränkenden Nutzungsbedingungen aus. Sora ist kein geeignetes Werkzeug, weil sich eine konsistente Sprecher-Identität über viele Videos hinweg nicht garantieren lässt. Wav2Lip und SadTalker wären prinzipiell denkbar. Wav2Lip ist jedoch auf die Neusynchronisierung bestehender Videos ausgelegt und erzeugt keine eigenständige Kopfbewegung. SadTalker ergänzt zwar Kopfbewegungen, basiert aber auf GAN-Architekturen, die inzwischen qualitativ hinter neueren Diffusionsmodell-basierten Ansätzen zurückliegen. Infinite Talk, das auf dem Wan 2.1 Modell aufbaut, ist das aktuell im Vergleich am ehesten geeignete Open-Source-Modell für den Anwendungsfall.

Hinweis: Die aufgeführten Preisangaben basieren auf den öffentlichen Angaben der jeweiligen Anbieter und wurden zuletzt im März 2026 abgerufen.

4 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an das zu entwickelnde System definiert. Grundlage dafür sind die im vorherigen Kapitel beschriebenen technischen Grundlagen und die konkreten Anforderungen des Forschungsprojekts des Doktoranden.

4.1 Zielgruppe und Anwendungskontext

Das System richtet sich in erster Linie an Forschende, die automatisiert KI-generierte audiovisuelle Inhalte erstellen möchten. Der unmittelbare Anwendungskontext ist das beschriebene Framing-Projekt des Doktoranden, bei dem unterschiedliche Varianten eines Nachrichtenvideos erzeugt werden sollen, zum Beispiel verschiedene Formulierungen desselben Themas.

Gleichzeitig soll das System so allgemein gehalten sein, dass es auch für andere ähnliche Anwendungsfälle genutzt werden kann. Der Doktorand wird das System dabei primär über die REST-API ansprechen, nicht über das Frontend.

4.2 Funktionale Anforderungen

Eine zentrale Funktion des Systems besteht darin, neue Jobs entgegenzunehmen und zu verwalten. Nutzende müssen die Möglichkeit haben, einen Job einzureichen. Ein Job enthält alle notwendigen Informationen zur Erstellung eines Videos. Dazu gehört mindestens der zu sprechende Text und die Auswahl eines Sprechers.

Nach dem Einreichen eines Jobs muss das System diesen in eine Warteschlange speichern und anschließend automatisch nacheinander verarbeiten, da die Ressourcen nur für einen Job gleichzeitig ausreichen.

Neben der Generierung selbst muss das System auch Funktionen zur Verwaltung von Jobs bereitstellen. Jobs müssen abgebrochen, bei Fehler wiederholt und nach Abschluss gelöscht werden können.

Eine optionale Nachbearbeitung soll möglich sein, bei der ein Intro in das Video eingefügt wird und/oder ein Wasserzeichen hinzugefügt wird.

Wenn ein Job erfolgreich abgeschlossen wurde, muss das erzeugte Video über die API heruntergeladen werden können.

4.3 Nicht-funktionale Anforderungen

Eine wichtige Anforderung des Systems ist die klare Trennung zwischen Benutzeroberfläche, Backend und KI-System. Im Gespräch mit dem Doktoranden wurde diese Architektur ausgewählt, da sie am besten zum Aufbau des Serversystems und Netzwerks passt, auf dem das System letztlich betrieben werden soll.

Die Sprecher, Workflows und Konfigurationsparameter sollen ohne Code-Änderungen austauschbar sein. Neue Sprecher sollen durch das Hinzufügen von Konfigurationsdateien und Assets eingebunden werden.

Ein weiterer wichtiger Aspekt ist die Automatisierung der Verarbeitungsschritte. Nutzende sollen möglichst keine manuellen Eingriffe durchführen müssen. Vom Einreichen eines Jobs bis zum fertigen Video soll der gesamte Prozess automatisiert ablaufen.

Das System soll möglichst einfach aufzusetzen sein und in vielen unterschiedlichen Linux-Systemumgebungen (Ubuntu/Debian, Arch) laufen können.

Schließlich sind die Wartbarkeit und Lesbarkeit des Backend-Codes wichtig. Nutzende sollen auch mit wenig Programmierkenntnissen Anpassungen vornehmen können.

5 Systemkonzept

Das Ziel des Systems ist, die KI-basierte Erzeugung sogenannter Sprechervideos so zu strukturieren, dass sie nicht mehr aus einzelnen manuellen Schritten besteht, sondern als automatisierter Prozess ausgeführt werden kann. Im Mittelpunkt steht dabei nicht die eigentliche Mediengenerierung, sondern vor allem die technische Umsetzung. Die verschiedenen Teilaufgaben, also Texteingabe, Auswahl eines Sprechers, Übergabe an ein KI-System, Nachbearbeitung und Bereitstellung des resultierenden Videos, sollen in einem System stattfinden.

Das Problem beim Einsatz von KI-Tools für Videogenerierung ist nicht, dass die einzelnen Tools fehlen, sondern dass man sie jedes Mal von Hand zusammenfügen muss. Das entwickelte System setzt an dieser Problematik an und bietet einen automatisierten Lösungsansatz. Über einen einzigen API-Aufruf soll eine vollständige Videogenerierung angestoßen werden können. Die Generierung muss ohne manuelle Eingriffe ablaufen. Die Ergebnisse müssen nachvollziehbar bleiben, weil es sich um ein Forschungsprojekt handelt, bei dem jeder Job dokumentiert und reproduzierbar sein muss. Und das System darf nicht so gebaut sein, dass jede Änderung, sei es ein neuer Sprecher oder ein neues TTS-Modell, einen Eingriff in den Code erfordert.

5.1 Anforderungen an das System

Aus den drei zentralen Zielen ergeben sich mehrere Anforderungen an das System. Die Anforderungen werden in funktionale und nicht funktionale Anforderungen aufgeteilt. Zu den funktionalen Anforderungen gehört, dass es möglich sein muss, neue Jobs zu erfassen, zu speichern und automatisiert abzuarbeiten. Außerdem muss der Bearbeitungsstand eines Auftrags nachvollziehbar sein, damit Nutzende sehen können, ob sich ein Job noch in der Warteschlange befindet, gerade verarbeitet wird oder bereits abgeschlossen ist. Nach erfolgreicher Verarbeitung muss das erzeugte Video abrufbar sein.

Zu den nicht funktionalen Anforderungen gehört, dass das System nachvollziehbar ist. Da das System im wissenschaftlichen Rahmen verwendet werden soll, müssen verwendete Parameter und Eingabedaten möglichst eindeutig einem Auftrag zugeordnet sein. Ein weiteres Kriterium ist die Erweiterbarkeit. Das System darf nicht auf einen einzigen konkreten Sprecher oder exakt einen ComfyUI-Workflow festgelegt sein, sondern soll unterschiedliche Konfigurationen unterstützen.

Darüber hinaus spielt die Trennung zwischen Benutzeroberfläche und eigentlicher Verarbeitungslogik eine wichtige Rolle. Das System soll nicht über ein konkretes Web-Frontend zu bedienen sein, sondern auch von anderen Anwendungen angesprochen werden können. Das ist sehr wichtig, weil das System als Grundlage für weitere Forschungsvorhaben dienen soll.

5.2 Aufbau des Systems

Das System ist in drei getrennte Komponenten aufgebaut. Ein Frontend für die Benutzerinteraktion, ein Node.js/Express.js-Backend für die Orchestrierung und ComfyUI als KI-Backend für die eigentliche Videogenerierung. Die Trennung von Frontend und Backend ist erforderlich, weil das System im Forschungskontext des Doktoranden nicht über eine Weboberfläche verwendet werden soll, sondern über die REST-API angesteuert wird.

Das Frontend kommuniziert ausschließlich über die Backend-API und enthält keine eigene Verarbeitungslogik. Wer einen neuen Job einreichen, den Status abfragen oder ein fertiggestelltes Video herunterladen möchte, kann das entweder über das Frontend oder direkt per HTTP-Request tun.

Das Backend ist die eigentliche Steuereinheit des Systems. Es nimmt Jobs entgegen, speichert sie in einer Datenbank, koordiniert die Abarbeitung und kommuniziert mit ComfyUI. Die Herausforderung lag dabei weniger in der Implementierung der einzelnen Endpunkte als in der asynchronen Koordination. Ein Video zu generieren dauert je nach Textlänge und Hardware mehrere Minuten oder Stunden. Die API kann also nicht einfach warten, bis ComfyUI fertig ist, stattdessen antwortet sie sofort mit HTTP „`202 Accepted`“ und lässt die Verarbeitung im Hintergrund weiterlaufen.

ComfyUI läuft als eigenständige Instanz und ist ausschließlich über seine HTTP-API an das Backend angebunden. Diese bewusste Trennung ermöglicht es, das KI-Backend in Zukunft auszutauschen oder zu erweitern, ohne das Backend ändern zu müssen.

Diese Trennung folgt dem Prinzip der Separation of Concerns (Dijkstra, 1982; Aseem Daga, 2006). Jede Komponente hat genau eine klar abgegrenzte Verantwortung. Das Frontend ist für die Benutzerinteraktion zuständig, dass Backend für die Orchestrierung und Persistenz, ComfyUI für die KI-Generierung. Änderungen in einer Schicht wirken sich nicht in andere Schichten aus, ein neues Frontend-Design erfordert keine Anpassung im Backend, und ein neues KI-Modell erfordert keine Änderungen an der REST-API.

Abbildung 2 veranschaulicht die Aufteilung des Systems in Frontend, Backend und KI-Backend. Für das Verständnis der Architektur ist dabei vor allem relevant, dass sämtliche Nutzerinteraktion ausschließlich über das Backend läuft und ComfyUI als getrennter Service eingebunden ist.

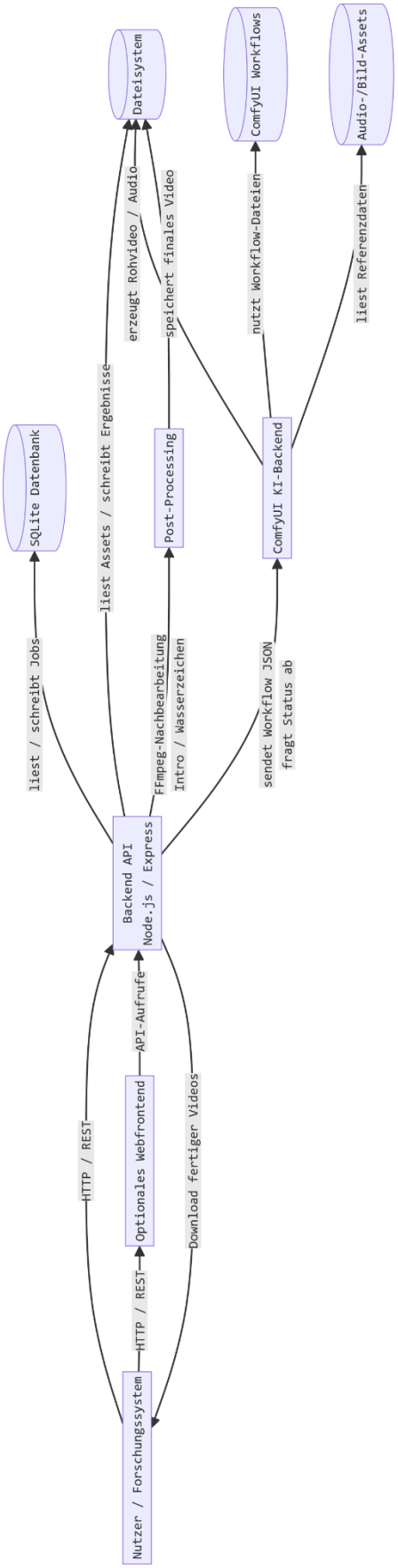

Abbildung 2: Architektur Diagramm des Systems. Quelle: Eigene Darstellung

5.3 Verarbeitungskonzept

Das zentrale Steuerungsprinzip ist der Job. Jeder Auftrag zur Videogenerierung wird als eigenständige Einheit in der Datenbank abgelegt und enthält alle Informationen, die für die Verarbeitung benötigt werden. Dazu zählt der zu sprechende Text, die gewählte Sprecher-ID, der ComfyUI-Workflow und optionale Einstellungen wie Wasserzeichen oder Intro-Video. Dieses Konzept macht den Verarbeitungsstatus jederzeit abfragbar und erlaubt es, Jobs bei Fehlern gezielt zu wiederholen.

Ob ein Job gerade verarbeitet wird, ist über das Feld „`running`“ in der Datenbank festgehalten. Das System verarbeitet immer nur einen Job gleichzeitig, nicht weil parallele Verarbeitung unmöglich wäre, sondern weil die zugrundeliegenden KI-Modelle die GPU vollständig auslasten. Eine parallele Abarbeitung würde bei einer einzelnen GPU keine Zeitersparnis bringen, sondern nur die Fehlerbehandlung und Ressourcenkoordination unnötig verkomplizieren.

Der eigentliche Ablauf folgt einem festen Muster. Nach dem Einreichen eines Jobs wird dieser zunächst mit dem Status „`queued`“ in der Datenbank gespeichert. Ein im Backend laufender Cronjob (ein zeitgesteuerter Hintergrundprozess, der in festgelegten Intervallen automatisch ausgeführt wird) prüft in regelmäßigen Intervallen, ob kein Job aktiv ist und ein neuer ausstehender Auftrag vorhanden ist. Ist das der Fall, wird der nächste Job ausgewählt, als „`running`“ markiert, der passende ComfyUI-Workflow befüllt und an ComfyUI übergeben. Nach Abschluss der KI-Generierung folgt optional die Nachbearbeitung des Videos, bevor der Job auf „`done`“ gesetzt wird. Der vollständige Ablauf wird im Rahmen der Implementierung in Kapitel 6.2.3 als Sequenzdiagramm detailliert dargestellt.

5.4 Modularität

Das System soll nicht nur technisch funktionieren, sondern auch anpassbar sein. Deshalb werden Bestandteile wie Sprecher, Workflows und Umgebungsparameter nicht ausschließlich in Code festgelegt, sondern als konfigurierbare Elemente behandelt. Dadurch wird erreicht, dass neue Sprecher oder alternative KI-Workflows mit vergleichsweise geringem Aufwand eingebunden werden können.

Diese Konfigurierbarkeit ist eng mit dem Ziel der Modularität verbunden. Es soll nicht an eine einzige konkrete Weboberfläche oder exakt einer KI-Modellkette gebunden sein. Vielmehr wurde es so entworfen, dass unterschiedliche Komponenten austauschbar bleiben. Das Frontend ist deshalb nicht als untrennbarer Teil der Verarbeitungslogik zu verstehen, sondern als eine mögliche Benutzerschnittstelle bzw. eine beispielhafte Implementierung. Die eigentliche Funktionalität liegt in der API und der dahinterliegenden Steuerung. Dadurch kann das System später auch von anderen Anwendungen genutzt werden.

6 Implementierung

In diesem Kapitel wird die konkrete Umsetzung des Systems beschrieben. Während im vorherigen Kapitel die konzeptionelle Architektur vorgestellt wurde, liegt hier der Fokus auf der praktischen Realisierung der einzelnen Systembestandteile. Ziel der Implementierung war es, ein lauffähiges und erweiterbares Websystem zu entwickeln, das die automatisierte Generierung von Videos mit einem lokal betriebenen KI-Systems verbindet.

Die Umsetzung orientiert sich dabei an den zuvor definierten Anforderungen. Vor allem sollten neue Jobs über eine Webschnittstelle eingereicht, in einer Warteschlage verwaltet, automatisiert verarbeitet und nach Abschluss zum Download zur Verfügung gestellt werden. Zusätzlich sollte das System so aufgebaut sein, dass neue Sprecher, weitere KI-Workflows und zusätzliche Verarbeitungsschritte mit möglichst wenig Aufwand ergänzt werden können.

6.1 Technologiestack und Projektstruktur

Für die Implementierung wurden bewusst Technologien gewählt, die leicht anwendbar, gut dokumentiert und für einen praxisnahen Aufbau geeignet sind. Das Backend wurde mit Node.js und Express.js umgesetzt. Diese Kombination eignet sich gut für REST-basierte Webanwendungen und ermöglicht eine übersichtliche Definition von API-Endpunkten. Die Jobs werden in einer SQLite-Datenbank gespeichert, hierfür wird das Paket „better-sqlite3“[11] verwendet, wodurch keine externe Datenbankinstanz betrieben werden muss. Das Abarbeiten der Warteschlange erfolgt über einen Cronjob, welcher mit dem Paket „node-cron“[12] umgesetzt wurde. Für die Nachbearbeitung der generierten Videos wird FFmpeg verwendet. Das KI-Backend wird durch ComfyUI bereitgestellt, das die Workflows zur Audio- und Videogenerierung ausführt.

Das Projekt ist in mehrere Teilbereiche untergliedert. ComfyUI wird als eigenständige Komponente betrieben. Daneben existieren ein Backend und ein Frontend als voneinander getrennte Node.js Anwendungen. Diese Trennung wurde bewusst gewählt, damit das eigentliche Forschungssystem des Doktoranden später direkt mit der Backend-API kommunizieren kann, ohne auf die Beispielweboberfläche angewiesen zu sein.

Zusätzlich existiert ein gemeinsamer Asset-Bereich, in dem Audio, Bild und Videodateien abgelegt werden. Diese Assets werden über ein Bash-Skript in die jeweils benötigten Verzeichnisse von ComfyUI und Backend kopiert. Dadurch wird vermieden, dass Dateien manuell eingepflegt werden müssen. Die genaue Ablagestruktur der Assets sowie die Verwendung der Kopierskripte sind in Anhang 2 beschrieben.

[11] https://www.npmjs.com/package/better-sqlite3 (Letzter Zugriff: 10.03.2026)
[12] https://www.npmjs.com/package/node-cron (Letzter Zugriff: 19.03.2026)

6.2 Backend

Das Backend bildet die zentrale Steuereinheit des Systems. Die folgenden Unterkapitel beschreiben die REST-API, die Datenbankstruktur, die Abarbeitungslogik der Jobwarteschlange sowie die optionale Nachbearbeitung.

6.2.1 REST-API und Endpunkte

Das Backend nimmt neue Jobs entgegen, speichert diese als Jobs in der Datenbank, überwacht deren Abarbeitung, kommuniziert mit ComfyUI und stellt die Ergebnisse über eine REST-API nach außen bereit. Die Endpunkte dieser API sind im Backend mithilfe von Express.js definiert.

Die nachfolgende Tabelle 2 gibt einen Überblick über alle verfügbaren HTTP-Endpunkte des Backends, inklusive einer kurzen Beschreibung und der Information, ob der jeweilige Endpunkt abgesichert ist.

Tabelle 2: HTTP-Endpunkte des Backends

Methode	Endpunkt	Authentifizierung	Beschreibung
POST	/api/job	Bearer Token	Neuer Generierungsauftrag einreichen
GET	/api/job/:jobId	Bearer Token	Statusabfrage eines einzelnen Jobs
GET	/api/jobs	Bearer Token	Alle Jobs zurückgeben
GET	/api/jobs/active	Bearer Token	Aktuell laufenden Job zurückgeben
POST	/api/job/:jobId/cancel	Bearer Token	Job abbrechen
POST	/api/job/:jobId/retry	Bearer Token	Fehlgeschlagenen oder abgebrochenen Job erneut einreihen
DELETE	/api/job/:jobId	Bearer Token	Job und Temp Daten löschen (nicht während aktiver Verarbeitung des Jobs)
GET	/api/job/:jobId/download	Capability URL (UUID)	Fertiges Video herunterladen
GET	/api/comfyui/health	Bearer Token	Online Status von ComfyUI abfragen
DELETE	/api/comfyui/outputs	Bearer Token	Inhalt des ComfyUI Output Ordners bereinigen
GET	/api/comfyui/speakers	Bearer Token	Alle verfügbaren Sprecher / Personas zurückgeben
GET	/api/comfyui/workflows	Bearer Token	Alle verfügbaren KI-Workflows zurückgeben

Eine vollständige Dokumentation aller Endpunkte inklusive Anfrage- und Antwortformaten, Authentifizierungshinweisen sowie Beispielobjekten findet sich in Anhang 1.

Da es sich bei dem entwickelten System um einen Forschungsprototypen handelt, der ausschließlich in einer kontrollierten Netzwerkumgebung betrieben wird, wurde das Sicherheitskonzept bewusst schlank gehalten. Der Zugriff auf die API wird über einen statischen Bearer-Token abgesichert, der per Umgebungsvariable konfiguriert wird und bei jeder Anfrage im Authorization-Header geprüft wird. Für den

Einsatz im Intranet ist dieser Ansatz ausreichend, vorausgesetzt es wird HTTPS zur Kommunikation verwendet. Für eine produktiv betriebene, öffentlich erreichbare Instanz wäre er jedoch nicht geeignet, in diesem Fall wären zeitlich begrenzte JSON Web Token (JWT) mit Ablaufdatum die empfehlenswertere Lösung.

Der Download Endpunkt ist absichtlich ohne Token Pflicht implementiert, da einige HTTP-Clients das Setzen von Authorization-Headern bei Datei Downloads nicht unterstützen. Als Ausgleich wird eine UUIDv4 als Teil der Download-URL genutzt. Da UUIDs 128 Bit groß sind, bietet diese Vorgehensweise ausreichend Entropie, um ein zufälliges Erraten der URL zuverlässig zu verhindern. Dieses Muster wird als „Capability URL“ (Jeni Tennison, 2014) bezeichnet und ist ein gängiger Kompromiss zwischen Sicherheit und Kompatibilität.

Der Endpunkt „`DELETE /api/comfyui/outputs`“ wurde nachträglich auf Wunsch des Doktoranden ergänzt, da andernfalls der Speicherplatz im ComfyUI-Ausgabeordner bei größeren Mengen generierter Videos regelmäßig manuell hätte freigegeben werden müssen. Der Endpunkt funktioniert nur, wenn Backend und ComfyUI auf demselben System betrieben werden und die Umgebungsvariable „`COMFYUI_PATH`“ konfiguriert ist. Außerdem ist die Bereinigung gesperrt, solange ein Job aktiv verarbeitet wird.

6.2.2 Datenbank und Jobverwaltung

Zur persistenten Speicherung der Jobs wird eine lokale SQLite-Datenbank eingesetzt. Die zugehörige Datenbankdatei wird beim ersten Start automatisch angelegt, sofern sie noch nicht existiert. Die Datenbank umfasst eine Tabelle namens „jobs", die alle relevanten Informationen eines Auftrags enthält. Gespeichert werden unter anderem die eindeutige „`jobId`“ (UUIDv4), die „`comfyPromptId`“ (welche nach der Übermittlung eines Workflows an ComfyUI befüllt wird), die beiden Statusfelder „`statusComfy`“ und „`statusPostProcessing`“, das Flag „`running`“, das serialisierte Feld „`workflowJson`“ sowie Zeitstempel für Erstellung und letzte Änderung. Eine Übersicht aller Spalten der „jobs"-Tabelle ist in Tabelle 3 dargestellt.

Tabelle 3: Datenbank – Spalten der Jobs Tabelle

Spalte	Typ	Constraint	Beschreibung
jobId	TEXT	PRIMARY KEY	UUID als eindeutiger Schlüssel
comfyPromptId	TEXT	NULLABLE	UUID, den ComfyUI zurückliefert beim Einreichen eines Jobs
statusComfy	TEXT	NOT NULL	Status der KI-Generierung: queued | processing | done | failed | cancelled | retry
statusPostProcessing	TEXT	NOT NULL	Status der Nachbearbeitung; analog zu statusComfy
running	INTEGER	NOT NULL	0 oder 1; gibt an, ob Job aktiv verarbeitet wird
workflowJson	TEXT	NULLABLE	Serialisiertes JSON-String mit den Job-Daten die übermittelt wurden

createdAt	TEXT	NOT NULL	Zeitstempel der Erstellung
updatedAt	TEXT	NOT NULL	Zeitstempel der letzten Änderung

Die Entscheidung, dass zwei getrennte Statusfelder existieren, wurde getroffen, weil ein Job aus mindestens zwei logischen Phasen besteht. Die erste Phase ist die Verarbeitung im KI-Backend und die zweite Phase die optionale Nachbearbeitung („Post-Processing“). Ein Video kann z.B. bereits erfolgreich durch ComfyUI generiert worden sein, während das anschließende Hinzufügen eines Intros oder Wasserzeichens noch aussteht oder fehlgeschlagen ist. Durch die Trennung in „`statusComfy`“ und „`statusPostProcessing`“ lässt sich dieser Zustand klarer abbilden als mit nur einem Gesamtstatus. Die möglichen Statuswerte für beide Felder sind „`queued`“, „`processing`“, „`retry`“, „`done`“, „`failed`“ und „`cancelled`“.

Ergänzend zu den Statuswerten wird über das Feld „`running`“ festgehalten, ob ein Job aktuell aktiv verarbeitet wird. Da die Videogenerierung mit ComfyUI und den verwendeten Modellen sehr ressourcenintensiv ist, wurde das System so umgesetzt, dass immer nur ein Job gleichzeitig aktiv ausgeführt wird. Die Datenbankschnittstelle stellt dafür u.a. Funktionen wie „`getRunningJob()`“, „`getNextPendingJob()`“ und „`setRunning()`“ zur Verfügung.

Ein weiterer Umsetzungsaspekt ist die sogenannte „Hydrierung“ der gespeicherten Workflow-Daten. Da das Workflow-Objekt als JSON-String in der Datenbank gespeichert wird, ein Prozess, der als Serialisierung bezeichnet wird und ein JavaScript-Objekt in eine speicherbare Zeichenkette (String) konvertiert, wird es beim Lesen automatisch wieder in ein JavaScript-Objekt umgewandelt (Deserialisierung). Dadurch kann der restliche Anwendungscode direkt damit arbeiten, ohne bei jedem Zugriff selbst eine Deserialisierung durchführen zu müssen.

Eine vollständige Übersicht der Tabellenspalten, Trigger sowie aller verfügbaren Datenbankfunktionen mit Parametern und Beispielcode ist in Anhang 3 dokumentiert.

6.2.3 Abarbeitung von Jobs

Eine Kernfunktion des Backends ist die automatische Verarbeitung eingereichter Jobs. Sobald ein neuer Auftrag gespeichert wurde, wird er nicht sofort in der HTTP-Anfrage verarbeitet, sondern zunächst in eine Warteschlange aufgenommen. Dieses Vorgehen ist notwendig, da die eigentliche Verarbeitung der KI-Modelle je nach Workflow und verfügbarer Hardware deutlich länger dauern kann. Die API antwortet deshalb bereits nach dem Anlegen eines Jobs mit einem passenden Statuscode, während die eigentliche Verarbeitung im Backend erfolgt.

Die Abarbeitung der Warteschlange erfolgt sequentiell, weshalb immer nur ein Job gleichzeitig aktiv verarbeitet wird. Dies hat den Hintergrund, dass es sonst einen zusätzlichen Aufwand bei der Koordination von Ressourcen und Fehlerfällen hätte, zumal der Engpass hier bei der Verarbeitung durch ComfyUI liegt und daher eine parallele Abarbeitung der Warteschlange kein Vorteil bringt. Für den

vorgesehenen Einsatz als Forschungsprototyp reicht diese Lösung aus. Der Zustand, ob ein Job ausgeführt wird, wird über das Feld „`running`“ definiert.

Wenn kein aktiver Job vorhanden ist, wählt das Backend den nächsten ausstehenden Eintrag aus der Datenbank aus, bei dem der Verarbeitungsstatus noch auf „`queued`“ oder „`retry`“ steht. Anschließend wird der „`running`“ Status des ausgewählten Jobs auf „1“ (True) gesetzt und durch das System weiterverarbeitet.

Vor dem Absenden an ComfyUI wird aus den gespeicherten Job-Daten ein konkreter ComfyUI-Workflow erzeugt. Dazu wird ein JSON-Template geladen und anschließend an den passenden Stellen mit den Job-Daten befüllt. Hierzu gehören der Text, die referenzierte Stimme, das Referenzbild, ggf. Videoprompts und weitere Parameter. Das Backend übernimmt dabei die Übersetzung zwischen relativ kompakter und einfacher Benutzeranfrage und dem deutlich ausführlicheren ComfyUI-Workflow. Dadurch wird die eigentliche Komplexität der KI-Pipeline von der Nutzerschnittstelle getrennt.

Nach dem erfolgreichen Absenden des Workflows an ComfyUI über eine API wird die von dort zurückgelieferte Prompt-ID (eine eindeutige UUID zur Nachverfolgung des Jobs in ComfyUI) im Job gespeichert. Diese Zuordnung ist wichtig, damit der laufende ComfyUI Prozess nachvollzogen oder bei Bedarf abgebrochen werden kann. Gleichzeitig wird der „`statusComfy`“ Status des Jobs auf „`processing`“ gesetzt. Tabelle 4 fasst die wesentlichen Zustandsübergänge eines Jobs zusammen. Die Tabelle dient dabei vor allem dazu, die Abarbeitungslogik des Systems nachvollziehbar zu machen.

Tabelle 4: Job-Status Übergangstabelle

Ausgangszustand	Zielzustand	Auslöser
-	queued	POST /api/job; Job erstellt
queued / retry	processing	Cronjob wählt Job aus der Warteschlange
processing	done	ComfyUI oder Post-Processing erfolgreich abgeschlossen
processing	failed	Fehler in ComfyUI oder Post-Processing
queued / processing	cancelled	POST /api/job/:jobId/cancel
failed / cancelled	retry	POST /api/job/:jobId/retry; wird wie queued behandelt
done / failed / cancelled	-	DELETE /api/job/:jobId; Job wird gelöscht

Die zugehörigen Status-Werte sowie deren genaue Bedeutung sind in der API-Dokumentation in Anhang 1 beschrieben.

Das Backend prüft in regelmäßigen Intervallen den Status des Fortschritts in ComfyUI ab. Tritt während der Verarbeitung in ComfyUI ein Fehler auf, wird der entsprechende Status („`statusComfy`“) im Job auf „`failed`“ gesetzt. Wenn ComfyUI die Ausführung erfolgreich abgeschlossen hat, wird der Status

auf „done“ gesetzt und das erzeugte Video vom Backend heruntergeladen. Zusätzlich bietet das System die Möglichkeit, fehlgeschlagene oder abgebrochene Jobs über einen API-Endpunkt erneut zur Verarbeitung zu markieren („retry“). Dieser Zustand wird durch die Datenbankschnittstelle „getNextPendingJob()“ berücksichtigt. Eine vollständige Übersicht aller Datenbankfunktionen inklusive Parametern und Beispielcode findet sich in Anhang 3.

Abbildung 3 zeigt den vollständigen Verarbeitungsablauf des Backends als Ablaufdiagramm, vom eingehenden API-Request über die KI-Generierung bis zum abschließenden Endstatus.

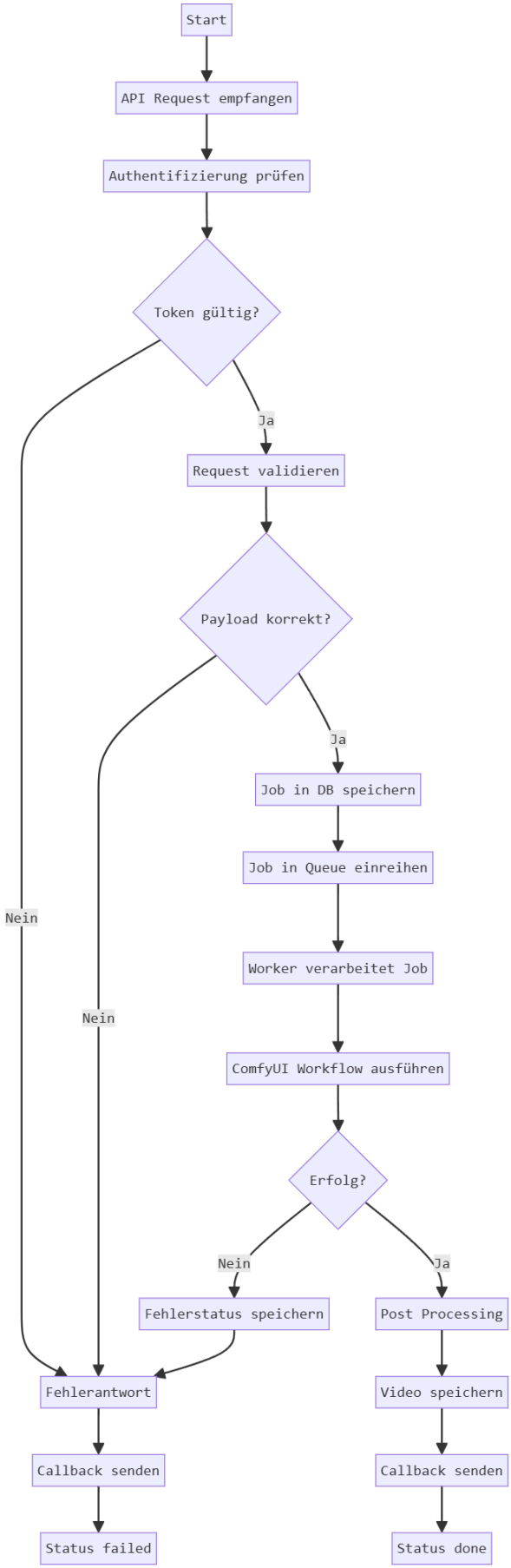

Abbildung 3: Ablaufdiagramm Verarbeitungsprozess im Backend. Quelle: Eigene Darstellung

Das Diagramm in Abbildung 3 beginnt mit der Authentifizierung und Payload-Validierung des eingehenden Requests. Danach übernimmt der asynchrone Worker-Prozess die Verarbeitung, bei der der ComfyUI Workflow ausgeführt wird und anschließend je nach Ergebnis in einen Erfolg oder Fehlerpfad verzweigt. Der im Diagramm dargestellte optionale Callback-Schritt wird im folgenden Absatz näher erläutert.

Im weiteren Verlauf des Projekts wurde das System um einen optionalen Callback Mechanismus (oder auch Webhook bekannt) erweitert. Dieser entstand auf Wunsch des Doktoranden, der das System über die REST-API ansteuert und daher aktiv abfragen müsste, um den Abschluss eines Jobs zu erkennen. Ist die Umgebungsvariable „`CALLBACK_URL`“ konfiguriert, sendet das Backend nach Abschluss oder Fehlschlag eines Jobs automatisch einen HTTP-POST-Request an diese Adresse. Über die Variable „`CALLBACK_TRIGGERS`“ lässt sich einschränken, bei welchen Statuswerten ein Callback ausgelöst wird, zum Beispiel ausschließlich bei „`done`“ oder „`failed`“. Bei erfolgreichen Jobs enthält der Callback-Payload neben der „`jobId`“ und dem „`status`“ auch eine „`downloadUrl`“, über die das fertige Video direkt heruntergeladen werden kann. Bei fehlgeschlagenen Jobs wird zusätzlich ein „`reason`“ Feld mitgesendet. Ist keine „`CALLBACK_URL`“ gesetzt, bleibt das Verhalten unverändert und der Client ist weiterhin für das Abfragen des Status verantwortlich.

Abbildung 4 veranschaulicht den vollständigen Verarbeitungsablauf als Sequenzdiagramm. Der obere Teil zeigt den synchronen Teil, die Anfrage des Nutzers und die sofortige Antwort mit HTTP „`202 Accepted`“. Der untere Teil zeigt den asynchronen Hintergrundprozess, der die eigentliche KI-Generierung und die Nachbearbeitung umfasst.

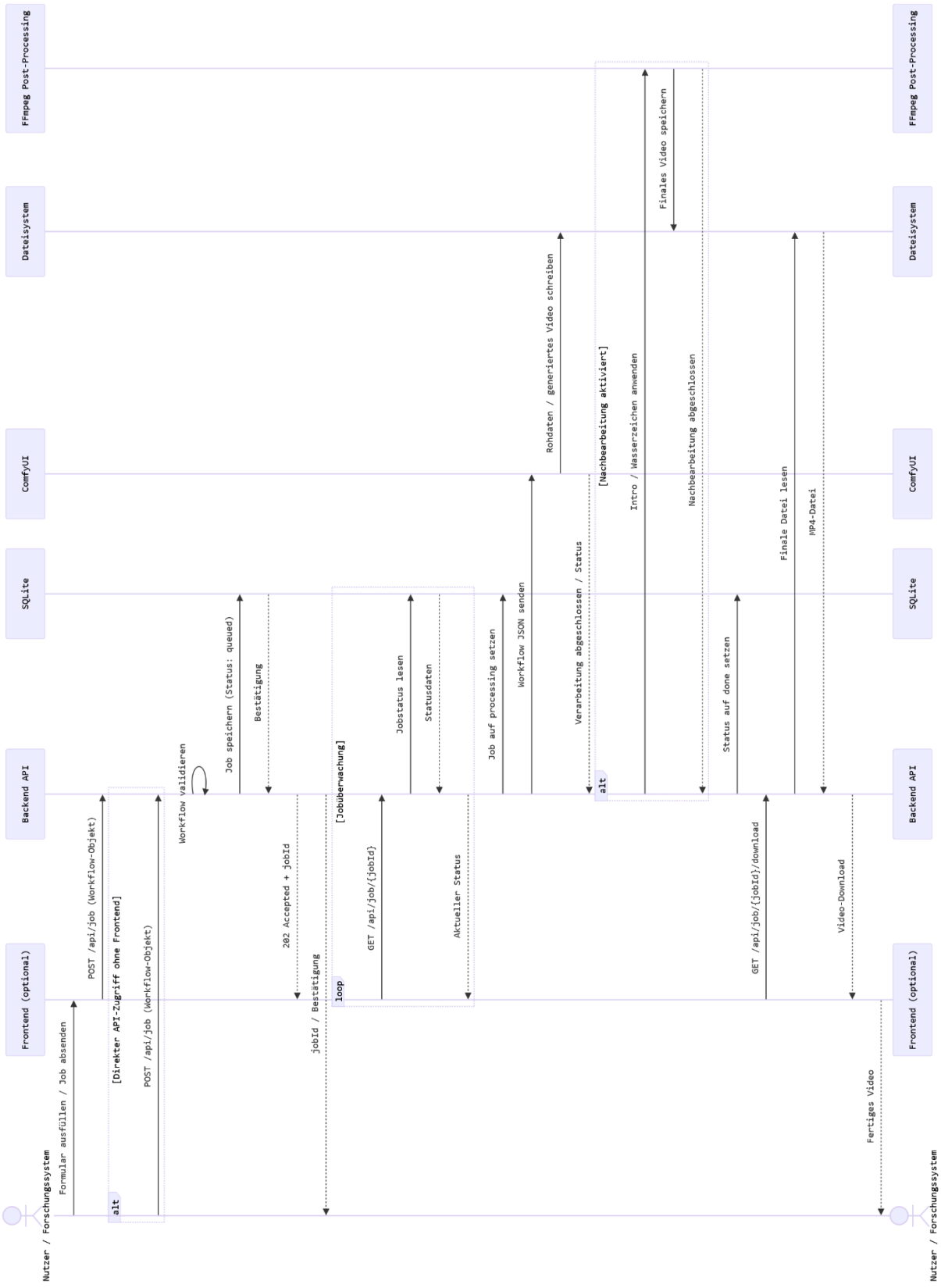

Abbildung 4: Sequenzdiagramm des Verarbeitungsablaufs. Quelle: Eigene Darstellung

6.2.4 Nachbearbeitung

Nach Abschluss der KI-Generierung in ComfyUI folgt eine optionale Nachbearbeitung („Post-Processing“). Je nach Konfiguration des Jobs können zusätzliche Bearbeitungsschritte ausgeführt werden, z.B. das Einfügen eines Intros oder das Hinzufügen eines Wasserzeichens. Für diese Aufgabe wird im Backend FFmpeg verwendet, das De-facto-Standardwerkzeug für Audio- und Videoverarbeitung, das sich zudem nahtlos in automatisierte Abläufe integrieren lässt.

Die Nachbearbeitung ist nicht im ComfyUI-Workflow integriert, sondern als eigenständiger Verarbeitungsschritt im Backend umgesetzt. Dadurch bleibt die Verantwortung klar getrennt. ComfyUI übernimmt die eigentliche KI basierte Generierung, während das Backend für organisatorische und technische Schritte zuständig ist. Änderungen an der Nachbearbeitung, etwa neue Overlays oder zusätzliche Videoelemente, können dadurch vorgenommen werden, ohne den KI-Workflow selbst neu aufbauen zu müssen, welcher wegen der hohen Komplexität weniger sinnvoll wäre.

Sobald die Nachbearbeitung erfolgreich abgeschlossen ist, wird der Status des Jobs entsprechend aktualisiert. Ein Auftrag gilt erst dann als vollständig beendet, wenn sowohl die KI-Verarbeitung als auch die Nachbearbeitung erfolgreich abgeschlossen wurden.

Die Konfiguration der Nachbearbeitungsoptionen, etwa das Hinterlegen von Intro-Videos und Wasserzeichendateien, ist in Anhang 2 beschrieben.

6.2.5 Konfigurierbarkeit und Erweiterbarkeit

Ein wesentliches Gestaltungsmerkmal des Systems ist, dass es ohne Code-Änderungen an neue Anforderungen angepasst werden kann.

Sprecher werden in der JSON-Datei „speakers.json“ im Backend definiert. Ein Eintrag enthält unter anderem die Referenz-Audiodatei für die TTS-Engine, das Referenzbild für die Videoanimation, ein optionales Intro-Video sowie die Auflösung des erzeugten Videos. Um einen neuen Sprecher hinzuzufügen, genügt es, die entsprechenden Asset-Dateien bereitzustellen und einen neuen Eintrag in der JSON-Datei anzulegen.

KI-Workflows werden in der JavaScript-Datei „workflows.js“ konfiguriert. Dort ist für jeden Workflow definiert, welche ComfyUI Workflow-JSON verwendet wird, welche Features es unterstützt (z.B. Videoprompt) und welche Node IDs in der Workflow-JSON vorhanden sind. Auf diese Weise kann das Backend bei jedem Job automatisch die richtigen Stellen im Workflow-JSON befüllen, ohne dass dafür Anpassungen am Code nötig sind.

Eine Anleitung zur Vorbereitung und Konfiguration von Assets, Sprechern, Workflows sowie der Backend- und Frontend-Umgebungsvariablen findet sich in Anhang 2.

6.3 Frontend

Das Frontend ist als eigenständige Node.js/Express.js-Anwendung implementiert und kommuniziert ausschließlich über die REST-API mit dem Backend. Als Template-Engine wird EJS[13] verwendet, die es ermöglicht, dynamische HTML-Seiten serverseitig zu rendern.

Das Frontend bietet folgende Funktionen: Einreichen neuer Jobs mit Texteingabe, Sprecher- und Workflow-Auswahl sowie optionalen Einstellungen, eine Übersicht aller vorhandenen Jobs mit Statusanzeige sowie Aktionen zum Abbrechen, Wiederholen und Löschen von Jobs. Abgeschlossene Videos können direkt im Browser heruntergeladen werden.

Abbildung 5 zeigt einen Screenshot des Frontend-Prototyps. Im oberen Bereich befindet sich das Formular zum Einreichen neuer Jobs, im mittleren Bereich der Pipeline-Status und im unteren Bereich die Jobliste mit Aktionsbuttons.

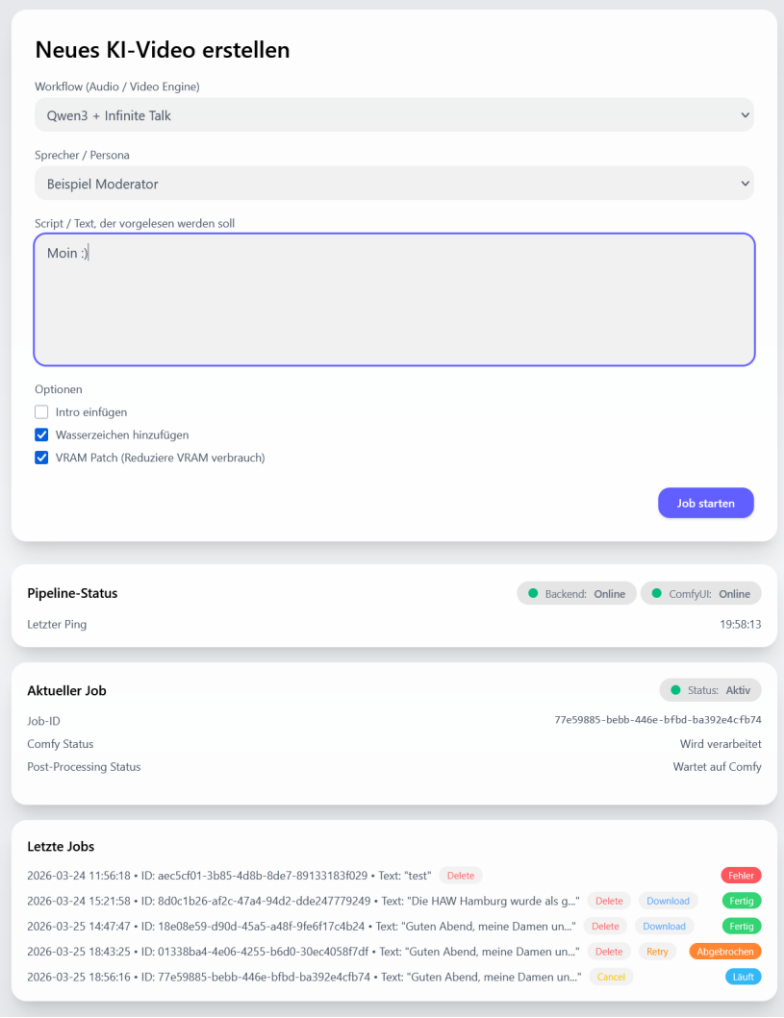

Abbildung 5: Screenshot vom Frontend Prototyp. Quelle: Eigene Darstellung

Als Erstes wählt man den gewünschten Workflow (z.B. „Qwen3 + Infinite Talk“), dann den Sprecher aus dem Dropdown Menü und gibt den zu sprechenden Text in ein Textfeld ein. Darunter befinden sich

[13] https://ejs.co/ (Letzter Zugriff: 16.03.2026)

optionale Einstellungen wie das Aktivieren des Wasserzeichens oder des VRAM-Patches. Im mittleren Bereich zeigt die Seite den aktuellen Pipeline-Status, ob Backend und ComfyUI erreichbar sind. Im unteren Bereich ist eine Liste der bisherigen Jobs mit ihrem Status und Aktionsbuttons zu sehen, wie z.B. das Löschen des Jobs oder das Herunterladen des fertigen Videos.

Die zugrunde liegenden API-Endpunkte, die das Frontend verwendet, sind in Anhang 1 dokumentiert.

6.4 Einrichtung und Deployment

Ein funktionierendes System setzt nicht nur eine korrekte Implementierung voraus, sondern auch eine reproduzierbare und möglichst einfache Einrichtung. Da das System auf unterschiedlichen Linux-Distributionen lauffähig sein soll und mehrere Komponenten (ComfyUI, Backend und Frontend) zusammenspielen müssen, wurde sowohl die Installation als auch der Betrieb so weit wie möglich automatisiert. Die folgenden Unterkapitel beschreiben zunächst das Installationsskript für ComfyUI und dessen Abhängigkeiten und anschließend die Containerisierung von Backend und Frontend mit Docker.

6.4.1 Installationsskript und ComfyUI-Abhängigkeiten

Ein Teil des praktischen Projektaufwands lag in der Einrichtung von ComfyUI selbst. Das Herunterladen der KI-Modelle, die Installation der benötigten ComfyUI-Nodes sowie das Einrichten der Python-Umgebung sind Schritte, die beim erstmaligen Aufsetzen manuell fehleranfällig und zeitintensiv sind. Um diesen Prozess zu vereinfachen und auf verschiedenen Linux-Systemen reproduzierbar zu gestalten, wurde ein Bash-Installationsskript entwickelt.

Das Skript erkennt automatisch das verwendete Betriebssystem (Arch Linux oder Ubuntu/Debian) und installiert die systemseitigen Abhängigkeiten über den jeweiligen Paketmanager. Anschließend wird eine virtuelle Python-Umgebung angelegt, in der das CLI-Tool „comfy-cli"[14] als Paketverwaltungs- und Steuerungswerkzeug für ComfyUI installiert wird. Über „comfy-cli" wird dann ComfyUI selbst eingerichtet, wobei der Nutzer während der Installation den GPU-Typ auswählen muss.

Im weiteren Verlauf installiert das Skript automatisch alle benötigten ComfyUI Custom Nodes. Tabelle 5 gibt einen Überblick über die installierten Nodes und ihre jeweilige Funktion:

Tabelle 5: ComfyUI Custom Nodes

ComfyUI Node	Funktion
ComfyUI-WanVideoWrapper[15]	Integration des Wan 2.1 Videogenerierungsmodells
ComfyUI-VideoHelperSuite[16]	Hilfsfunktionen für Videoein- und -ausgabe
TTS-Audio-Suite[17]	Integration von Chatterbox TTS

[14] https://github.com/Comfy-Org/comfy-cli (Letzter Zugriff 25.03.2026)
[15] https://github.com/kijai/ComfyUI-WanVideoWrapper (Letzter Zugriff: 20.03.2026)
[16] https://github.com/Kosinkadink/ComfyUI-VideoHelperSuite (Letzter Zugriff: 20.03.2026)
[17] https://github.com/diodiogod/TTS-Audio-Suite (Letzter Zugriff: 20.03.2026)

ComfyUI-Qwen3-TTS[18]	Integration von Qwen3-TTS
ComfyUI-Qwen3-ASR[19]	Automatische Spracherkennung für Qwen3-TTS
rgthree-comfy[20]	Hilfsnodes für ComfyUI-Workflow-Organisation

Eine vollständige Schritt-für-Schritt-Anleitung zur Installation, ist in Anhang 4 beschrieben.

Neben den Custom Nodes lädt das Skript automatisch alle benötigten KI-Modelle für das Infinite Talk Workflow von HuggingFace[21] herunter und platziert sie in den jeweils vorgesehenen Unterverzeichnissen von ComfyUI. Die KI-Modelle für „TTS-Audio-Suite“ und „Qwen3-TTS“ werden automatisch von den Workflows heruntergeladen, wenn diese einmalig bei einem Generierungsjob aufgerufen werden.

6.4.2 Containerisierung mit Docker

Um den Betrieb des Systems auf unterschiedlichen Linux-Systemen weiter zu vereinfachen, werden Backend und Frontend als Docker-Container bereitgestellt. Docker[22] ist ein weit verbreitetes System zur Containerisierung von Anwendungen, dabei bündelt ein Container die Anwendung zusammen mit allen ihren Abhängigkeiten in einer isolierten Umgebung, sodass sie unabhängig vom Host-System lauffähig sind. Für jede der beiden Anwendungen existiert jeweils eine eigene „Dockerfile“ (zum Bauen des Containers), und für das Starten beider Dienste wird jeweils eine „docker-compose.yml“ bereitgestellt.

Das „Dockerfile“ des Backends basiert auf einem offiziellen Node.js Alpine Image[23]. Es kopiert den Quellcode in den Container, installiert die benötigten Abhängigkeiten und startet den Express.js-Server. Die SQLite-Datenbankdatei sowie generierte Videos werden über ein Docker-Volume in das Host-Dateisystem eingehängt, damit die Daten nach einem Neustart des Containers erhalten bleiben. Das Frontend ist ähnlich aufgebaut, benötigt jedoch kein persistentes Volume.

Eine Containerisierung von ComfyUI selbst wäre technisch möglich, erfordert jedoch zusätzliche Konfiguration für den GPU-Zugriff (z.B. NVIDIA Container Toolkit[24]) und würde trotzdem dieselben manuellen Installationsschritte für Modelle und Custom Nodes erfordern. Der Nutzen wäre daher im Verhältnis zum Aufwand gering, weshalb ComfyUI weiterhin direkt auf dem Host-System betrieben wird.

Die Befehle zum Starten von Backend und Frontend, sowohl als auch per Docker Compose, sind in Anhang 4 aufgeführt.

[18] https://github.com/DarioFT/ComfyUI-Qwen3-TTS (Letzter Zugriff: 20.03.2026)
[19] https://github.com/DarioFT/ComfyUI-Qwen3-ASR (Letzter Zugriff: 20.03.2026)
[20] https://github.com/rgthree/rgthree-comfy (Letzter Zugriff: 21.03.2026)
[21] https://huggingface.co/ (Letzter Zugriff: 21.03.2026)
[22] https://www.docker.com/ (Letzter Zugriff: 14.03.2026)
[23] https://hub.docker.com/_/node (Letzter Zugriff: 14.03.2026)
[24] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html (Letzter Zugriff: 19.03.2026)

6.5 Teststrategie

Um die Kernfunktionen des Backends zu prüfen, wurden automatisierte Tests mit zwei etablierten JavaScript-Bibliotheken umgesetzt. Jest[25] als Test-Framework und Supertest[26] für HTTP-nahe Routentests. Jest ermöglicht das Schreiben und Ausführen von Unit-Tests direkt in Node.js-Projekten, während Supertest HTTP-Anfragen gegen Express.js-Routen simulieren kann, ohne dass dafür ein laufender Server notwendig ist.

Der Schwerpunkt der Tests lag auf drei Bereichen: den API-Routen, der Authentifizierungs-Middleware und der Eingabevalidierung.

Für die API-Routen wurden Tests für alle wesentlichen Endpunkte erstellt. Da die Tests isoliert und reproduzierbar sein sollten, wurden alle externen Abhängigkeiten des Routers durch sogenannte „Mocks" ersetzt. Datenbankzugriffe, die Validierungslogik, Cleanup-Funktionen und der ComfyUI-Client wurden durch simulierte Platzhalter ersetzt, die definierte Rückgabewerte liefern. Dadurch können die Tests ohne eine laufende Datenbankverbindung oder eine aktive ComfyUI-Instanz ausgeführt werden. Getestet wurden sowohl Erfolgsfälle als auch relevante Fehlerfälle, zum Beispiel das Ablehnen ungültiger Workflow-Daten beim Erstellen eines Jobs, die korrekte Fehlerantwort bei nichtexistierenden Jobs, der Schutz vor dem Löschen aktiver Jobs sowie das erwartete Verhalten beim Abbrechen, Wiederholen und Abfragen von Jobs.

Die Authentifizierungs-Middleware wurde in separaten Tests geprüft. Diese decken ab, ob fehlende oder falsch formatierte Authorization-Header korrekt abgelehnt werden, ob eine fehlende Umgebungsvariable für das Token zu einem Fehler führt, ob ungültige Tokens abgelehnt werden und ob ein gültiges Token die Anfrage wie erwartet weiterleitet.

Für die Eingabevalidierung der Job-Daten wurden Tests geschrieben, die prüfen, ob das Fehlen von Pflichtfeldern wie Text, Sprecher-ID und Workflow-Name korrekt erkannt wird und die API im Fehlerfall eine aussagekräftige Fehlermeldung zurückgibt.

Was nicht Teil der automatisierten Teststrategie war, ist ein Integrations- oder Systemtest gegen eine echte, laufende ComfyUI-Instanz. Das bedeutet, die tatsächliche Kommunikation zwischen Backend und ComfyUI, also das Einreichen eines Workflows, das Abfragen des Fortschritts und das Herunterladen des generierten Videos, wurde ausschließlich durch manuelle Tests im realen Betrieb geprüft. Automatisierte Integrationstests für diesen Teil wären zwar sinnvoll, hätten jedoch eine dauerhaft laufende ComfyUI-Instanz mit entsprechender GPU-Hardware vorausgesetzt und hätten den zeitlichen Rahmen dieses Projekts gesprengt.

[25] https://jestjs.io/ (Letzter Zugriff: 19.03.2026)
[26] https://github.com/forwardemail/supertest (Letzter Zugriff: 19.03.2026)

7 Evaluation

In diesem Kapitel wird bewertet, inwieweit das entwickelte System die definierten Anforderungen erfüllt. Dabei geht es sowohl um die funktionalen als auch um die nicht-funktionalen Anforderungen.

7.1 Testumgebung

Das System wurde auf einem Linux-System mit einer NVIDIA RTX 3090 (24 GB VRAM) Grafikkarte getestet. Als Betriebssystem wurde primär Arch Linux verwendet, sekundär Ubuntu 24.04.4 LTS in einer WSL-Umgebung unter Windows. Die Tests umfassten sowohl die Installation des Systems mithilfe des bereitgestellten Installationsskripts als auch die Verarbeitung mehrerer Jobs mit unterschiedlichen Sprechern und Workflows. Tabelle 6 gibt einen Überblick über die Testumgebung:

Tabelle 6: Testumgebung

Komponente	Version / Spezifikation
GPU	NVIDIA GeForce RTX 3090 (24GB VRAM)
RAM	32 GB
Betriebssystem	Arch Linux / Ubuntu 24.04 LTS
Node.js	24.x
Python	3.12
ComfyUI	0.16.3
TTS-Engines	Chatterbox TTS, Qwen3-TTS
Videomodell	Wan 2.1, Infinite Talk (Quantisiert auf Q4_K_M)

7.2 Funktionstests

Die wichtigsten Funktionen des Systems wurden im Rahmen manueller Tests überprüft. Dabei wurden verschiedene Szenarien mit unterschiedlichen Sprechern, Workflows und Konfigurationen getestet.

Das grundlegende Szenario, einen Job einreichen und automatisch verarbeiten lassen, funktionierte in allen Testfällen wie erwartet. Jobs wurden sowohl über die API als auch über das Frontend eingereicht und in allen Fällen korrekt in die Warteschlange aufgenommen und automatisch verarbeitet. Die Statusabfragen während der Verarbeitung lieferten stets den aktuellen Zustand des Jobs.

Der Wechsel zwischen verschiedenen Sprechern funktionierte ohne Anpassungen am Code. Es reicht, die Sprecher-ID im Request zu ändern, das Backend wählt dann automatisch die passenden Referenzdateien aus der Konfiguration. Mehrere Sprecher wurden konfiguriert und erfolgreich getestet.

Beide TTS-Engines (Chatterbox und Qwen3-TTS) wurden mit jeweils mehreren Sprechern getestet. In den durchgeführten Praxistests erzeugten beide realistische synthetische Sprache auf Basis der jeweili-

gen Referenzstimme. Qwen3-TTS wirkte dabei im subjektiven Eindruck meist natürlicher, wurde jedoch bei längeren Ausgaben leicht leiser zum Ende hin. Es wurde keine Nutzerstudie durchgeführt, daher ist diese Einschätzung als projektbezogene Beobachtung zu verstehen.

Die optionalen Nachbearbeitungsschritte wurden ebenfalls überprüft. Jobs mit aktiviertem Wasserzeichen und Jobs mit Intro-Video wurden jeweils erfolgreich verarbeitet. Die erzeugten Videos enthielten die gewünschten Elemente in korrekter Form.

Auch die Fehlerbehandlung wurde getestet. Fehlgeschlagene Jobs konnten über den „`POST /api/job/:jobId/retry`“-Endpunkt erneut zur Verarbeitung angestoßen werden. Laufende Jobs ließen sich über den „`POST /api/job/:jobId/cancel`“-Endpunkt abbrechen, woraufhin der Job den Status „`cancelled`“ erhielt und das System zum nächsten wartenden Job wechselte. Die genauen Anfrage- und Antwortformate dieser Endpunkte sind in Anhang 1 dokumentiert.

Die Verarbeitungsdauer eines Jobs variiert je nach Länge des Textes und der gewählten Konfiguration. In den Testläufen lagen die Zeiten für die KI-Generierung je nach Textlänge bei etwa fünf bis fünfzehn Minuten. Die anschließende Nachbearbeitung mit FFmpeg dauerte in der Regel nur wenige Sekunden.

7.3 Bewertung der Anforderungen

Die folgende Tabelle stellt die in Kapitel 4 und 5 definierten Anforderungen den Testergebnissen gegenüber. Jede Anforderung wird mit „Erfüllt“, „Teilweise erfüllt“ oder „Nicht erfüllt“ bewertet und kurz begründet.

Tabelle 7: Erfüllung der Anforderungen

Anforderung	Status	Bemerkung
Jobs über API einreichen, verwalten, abfragen	Erfüllt	Alle Endpunkte implementiert und getestet
Sequenzielle automatische Abarbeitung	Erfüllt	Cronjob im Backend, ein Job gleichzeitig
Abbrechen, Wiederholen, Löschen von Jobs	Erfüllt	Cancel, retry, delete Endpunkte vorhanden
Optionale Nachbearbeitung	Erfüllt	FFmpeg integriert, pro Job konfigurierbar
Video Download nach Abschluss	Erfüllt	Download mit Job UUID
Sprechern ohne Code Änderung hinzufügbar	Erfüllt	speakers.json, Asset Kopier BASH Script
Workflow ohne Code Änderung konfigurierbar	Erfüllt	workflows.js, ComfyUI Template
Trennung Frontend / Backend / KI	Erfüllt	Drei eigenständige Anwendungen, Docker
Betrieb auf verschiedenen Linux Systemen	Erfüllt	Arch und Ubuntu getestet
Einfaches Deployment	Teilweise	Docker für Backend / Frontend; ComfyUI über Script oder manuell

Sequenzielle Verarbeitung mehrerer Jobs	Erfüllt	Bewusste Designentscheidung; parallele Verarbeitung war aufgrund der GPU-Auslastung explizit nicht vorgesehen
Integrationstest gegen echte ComfyUI Instanz	Nicht erfüllt	ComfyUI wird nicht getestet, benötigt viel Zeit und Ressourcen

Die nicht-funktionalen Anforderungen werden weitgehend erfüllt. Sprecher und Workflows sind ohne Code-Änderungen konfigurierbar. Die Trennung von Frontend und Backend ist konsequent umgesetzt, das System kann vollständig ohne Frontend über die API genutzt werden.

Das bereitgestellte Installationsskript vereinfacht das Aufsetzen erheblich, wenngleich der manuelle Aufwand für die ComfyUI-Konfiguration nicht vollständig entfällt.

Eine bekannte Einschränkung ist die sequenzielle Verarbeitung. Das System verarbeitet immer nur einen Job gleichzeitig. Für den vorgesehenen Forschungskontext ist das vertretbar, um größere Mengen gleichzeitig verarbeiten zu können, müsste die Architektur erweitert werden. Außerdem setzt das System eine leistungsstarke NVIDIA GPU mit mindestens 24 GB VRAM voraus, was die Nutzbarkeit auf entsprechend ausgestattete Systeme beschränkt.

8 Ethische Aspekte

Der Einsatz von KI-Technologien zur Generierung audiovisueller Inhalte wirft grundlegende ethische und rechtliche Fragen auf. In diesem Kapitel werden die wesentlichen Aspekte beleuchtet, die beim Einsatz des entwickelten Systems zu berücksichtigen sind.

8.1 Deepfakes und Missbrauchspotenzial

Talking-Head-Modelle wie das verwendete Infinite-Talk-Modell ermöglichen es, ein beliebiges Referenzbild einer Person mit einer synthetischen oder aufgezeichneten Tonspur zu animieren und so einen realistisch wirkenden virtuellen Sprecher zu erzeugen. Solche Verfahren werden allgemein als „Deepfake" bezeichnet (Claudiu Popa, 2025).

Das Missbrauchspotenzial ist erheblich. Es können falsche Aussagen erzeugt werden, die eine reale Person nie getätigt hat. Gefälschte Nachrichtenvideos könnten Desinformationen verbreiten, und das Ansehen einer Person könnte ohne deren Zustimmung durch gefälschte Inhalte beschädigt werden (Mohamed R. Shoaib, 2023). Die verwendeten TTS-Modelle verstärken dieses Risiko, da sie in der Lage sind, die Stimme einer Person anhand einer kurzen Referenzaufnahme zu imitieren (Hussam Azzuni, 2025).

Besonders schwerwiegend wäre der Einsatz dieser Technologien zur Erzeugung von sexuellem Missbrauchsmaterial (CSAM, engl. Child Sexual Abuse Material). Entsprechende Inhalte sind unabhängig davon, ob sie mit realen Personen oder vollständig synthetisch erzeugt werden, in Deutschland nach §184b StGB strafbar und fallen in vielen Jurisdiktionen weltweit unter explizite Verbote.

8.2 Rechtliche Rahmenbedingungen

Für den Einsatz von KI-generierten Deepfake-Inhalten gelten in der Europäischen Union und in Deutschland konkrete gesetzliche Vorgaben, die beim Betrieb eines solchen Systems beachtet werden müssen.

Der EU AI Act (Europäisches Parlament, 2024), der seit 2025 schrittweise in Kraft tritt, stuft Systeme zur Generierung synthetischer Medieninhalte als regulierungspflichtig ein. Er verpflichtet Betreiber solcher Systeme dazu, KI-generierte Audio-, Bild- und Videoinhalte als solche zu kennzeichnen, sofern sie für die Öffentlichkeit bestimmt sind.

8.3 Einwilligung und Datenschutz

Ein zentrales ethisches Kriterium ist die informierte Einwilligung der beteiligten Personen. Wer als Sprecher in einem solchen System verwendet wird, muss dem ausdrücklich zugestimmt haben. Das gilt sowohl für die Referenzaudiodatei, die zur Stimmsynthese dient, als auch für das Referenzbild, das für die Gesichtsanimation verwendet wird.

Da das System lokal betrieben wird, verbleiben alle verarbeiteten Daten auf dem eigenen System und werden nicht an externe Dienste übermittelt. Dies ist ein wesentlicher Vorteil gegenüber kommerziellen Anbietern, bei denen Bild- und Audiodaten auf externen Servern verarbeitet werden. Der lokale Betrieb schließt jedoch ethische und rechtliche Fragen nicht aus, insbesondere wenn Aufnahmen realer Personen ohne deren Wissen verwendet werden.

8.4 Einsatz im Forschungskontext

Das System wird im Rahmen eines kontrollierten Forschungsprojekts an der HAW Hamburg eingesetzt. Die verwendeten Referenzmaterialien und erzeugten Videos dienen ausschließlich wissenschaftlichen Zwecken zur Untersuchung von politischem Framing. Dennoch sollte auch im Forschungskontext sichergestellt sein, dass alle beteiligten Personen ihrer Verwendung als Sprecher ausdrücklich zugestimmt haben und dass erzeugte Inhalte nicht außerhalb des kontrollierten Forschungsrahmens verbreitet werden.

9 Fazit und Ausblick

In diesem abschließenden Kapitel werden die Ergebnisse der Arbeit zusammengefasst und bewertet. Zunächst wird die eingangs gestellte Leitfrage direkt beantwortet und eingeordnet, inwieweit der entwickelte Prototyp diese Frage in der Praxis beantworten konnte. Anschließend folgt eine Reflexion über Entscheidungen, die im Rückblick anders getroffen hätten werden können.

9.1 Beantwortung der Leitfrage

Die Leitfrage dieser Arbeit lautete: „Wie kann die KI-basierte Generierung audiovisueller Medieninhalte durch ein modulares Websystem automatisiert und reproduzierbar gemacht werden?“

Der entwickelte Prototyp funktioniert, jedoch mit Einschränkungen. Es zeigt, dass sich bestehende Open-Source-Modelle für TTS und Talking-Head Generierung zu einem automatisierten System zusammenfügen lassen. Die wichtigste Erkenntnis dabei ist, dass das Problem nicht in der Existenz der einzelnen KI-Modelle liegt, sondern in deren Orchestrierung. Das adressiert dieses Projekt durch eine REST-API als einheitliche Schnittstelle, eine Jobqueue für die asynchrone Verarbeitung und ComfyUI als flexibles KI-Backend. Es entsteht ein System, bei dem ein einziger API-Aufruf ausreicht, um eine vollständige Videogenerierung anzustoßen.

Die Reproduzierbarkeit ist das stärkste Argument für die gewählte Architektur. Alle Eingaben, Parameter und Statusinformationen eines Jobs bleiben in der Datenbank erhalten. Das bedeutet, wer in sechs Monaten wissen möchte, mit welchem Workflow, welchem Sprecher und welchem Text ein bestimmtes Video erzeugt wurde, kann das nachvollziehen.

Die Modularität ist gut umgesetzt. Sprecher lassen sich ohne Code-Änderungen einbinden, neue ComfyUI-Workflows erfordern lediglich einen Konfigurationseintrag. Die Grenzen liegen tiefer im System. Das aktuelle System steuert genau eine ComfyUI-Instanz an und verarbeitet Jobs sequenziell. Wer das erweitern möchte, sodass mehrere ComfyUI-Instanzen, zum Beispiel auf verschiedenen Rechnern mit eigener GPU, gleichzeitig Jobs verarbeiten können, müsste erhebliche Teile des Backends überarbeiten. Das ist vertretbar für einen Prototyp, zeigt aber, dass mit „modular" eher die Konfigurationserweiterung gemeint ist als die vollständige Austauschbarkeit der Kernkomponenten.

9.2 Reflexion

Rückblickend gibt es einige Punkte, die bei einer Weiterentwicklung anders angegangen werden sollten.

Die Cronjob-basierte Warteschlange erfüllt ihren Zweck, ist aber für eine spätere Skalierung wenig geeignet. Würde das System auf mehrere Maschinen mit jeweils eigener GPU ausgeweitet, wäre eine dedizierte Task-Queue-Bibliothek wie BullMQ[27] von Beginn an die robustere Grundlage gewesen.

[27] https://bullmq.io/ (Letzter Zugriff: 19.03.2026)

Die automatisierten Tests decken die API-Routen, die Authentifizierung und die Eingabevalidierung ab, nicht aber die tatsächliche Kommunikation mit ComfyUI. Integrationstests gegen eine echte ComfyUI-Instanz wären eine sinnvolle Ergänzung, hätten jedoch dauerhaft laufende GPU-Hardware vorausgesetzt.

Das Frontend wurde mit dem geringsten Zeitaufwand aller Systemkomponenten entwickelt, was sich in der Codequalität widerspiegelt. Die Entscheidung, auf ein JavaScript-Framework wie Vue.js oder React zu verzichten, war pragmatisch begründet, das Frontend ist kein zentraler Bestandteil des Systems und dient ausschließlich als Demonstrationswerkzeug. Rückblickend wäre jedoch ein komponentenbasierter Ansatz mit Vue.js oder React besser gewesen, da dieser die Wartbarkeit und Erweiterbarkeit des Codes durch klare Strukturierung und Wiederverwendbarkeit von Komponenten verbessert hätte.

Eine weitere Einschränkung betrifft längere Videos, da die Qualität der Animation mit Infinite Talk mit zunehmender Länge abnehmen kann. Eine mögliche Weiterentwicklung wäre daher ein Wechsel zwischen verschiedenen Szenen desselben Sprechers, um diese Schwankungen weniger auffällig zu machen. Dies wäre jedoch technisch aufwendiger.

9.3 Ausblick

Das System ist als Prototyp konzipiert und für den geplanten Einsatz im Forschungsprojekt gut geeignet. Für eine Weiterentwicklung gibt es mehrere naheliegende Richtungen mit unterschiedlichem Aufwand.

Am einfachsten wäre die Integration weiterer TTS-Engines. Das Workflow-System in ComfyUI ist flexibel genug, um neue Modelle mit überschaubarem Aufwand einzubinden.

Komplexer, aber erheblich leistungsfähiger wäre die Unterstützung paralleler Jobverarbeitung über mehrere ComfyUI-Instanzen hinweg. Wenn mehrere Maschinen oder Systeme mit jeweils eigener GPU verfügbar sind, auf denen jeweils eine eigene ComfyUI-Instanz läuft, könnte das Backend so erweitert werden, dass Jobs gleichzeitig an verschiedene dieser Instanzen verteilt werden. Statt eines einzigen aktiven Jobs würden dann mehrere Jobs parallel laufen, jeder auf einer eigenen GPU. Das erfordert eine Verwaltungsschicht, die weiß, welche ComfyUI-Instanz gerade verfügbar ist. Der Aufwand für Job-Scheduling, Fehlerbehandlung und Zustandsverwaltung ist dabei erheblich, wäre aber für größere Produktionsmengen ein sinnvoller nächster Schritt.

Eine weniger offensichtliche, aber für das Forschungsprojekt nützliche Erweiterung wäre eine Batch-Submission-API. Statt Jobs einzeln einzureichen, könnte ein Forschungssystem eine Liste von Text-Varianten auf einmal übergeben, die dann automatisch nacheinander verarbeitet werden. Das würde den Integrationsaufwand auf Seiten des Forschungssystems reduzieren.

Schließlich wäre ein verbessertes Echtzeit-Monitoring sinnvoll. Das implementierte Callback-System benachrichtigt bei Abschluss oder Fehler eines Jobs automatisch eine konfigurierte URL, wodurch aktives Polling für den Standardfall entfällt. Während der eigentlichen Verarbeitung gibt es jedoch keine

Zwischenstandmeldungen. WebSockets oder Server-Sent Events wären hier eine elegantere Ergänzung, die den Fortschritt eines laufenden Jobs in Echtzeit an den Client übertragen könnten, ohne dass dieser regelmäßig den Statusendpunkt abfragen müsste.

Eine weitere mögliche Erweiterung wäre die Unterstützung mehrerer Sprecher innerhalb eines Videos. Durch die vorhandene ComfyUI-Integration wäre dies grundsätzlich umsetzbar, würde aber zusätzlichen Aufwand bei Konfiguration und Verwaltung mit sich bringen.

Literaturverzeichnis

Aseem Daga, S. d. (Dezember 2006). *Separation of Concerns: Techniques, Issues and Implications.* doi:10.1515/JISYS.2006.15.1-4.153

Claudiu Popa, R. P. (15. Juni 2025). *Deepfake Technology Unveiled: The Commoditization of AI and Its Impact on Digital Trust.* Abgerufen am 12. März 2026 von https://arxiv.org/abs/2506.07363

Dijkstra, E. W. (1982). *On the role of scientific thought.* New York, NY, US: Springer-Verlag.

Edresson Casanova, K. D. (7. Juni 2024). *XTTS: a Massively Multilingual Zero-Shot Text-to-Speech Model.* Abgerufen am 19. März 2026 von https://arxiv.org/abs/2406.04904

Entman, R. M. (1993). *Framing: Toward Clarification of a Fractured Paradigm.* Abgerufen am 14. März 2026 von https://fbaum.unc.edu/teaching/articles/J-Communication-1993-Entman.pdf

Europäisches Parlament. (12. Juli 2024). *Festlegung harmonisierter Vorschriften für künstliche Intelligenz*. Abgerufen am 21. März 2026 von https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:02024R1689-20240712

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* Abgerufen am 12. März 2026 von https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf

Hussam Azzuni, A. E. (1. Mai 2025). *Voice Cloning: Comprehensive Survey.* Abgerufen am 12. März 2026 von https://arxiv.org/abs/2505.00579

Javier Selva, A. S. (16. Januar 2022). *Video Transformers: A Survey.* Abgerufen am 11. März 2026 von https://arxiv.org/abs/2201.05991

Jeni Tennison, O. (18. Februar 2014). *Good Practices for Capability URLs*. Abgerufen am 15. März 2026 von https://www.w3.org/TR/2014/WD-capability-urls-20140218/

Jonathan Shen, R. P.-R. (16. Dezember 2017). *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions.* Abgerufen am 13. März 2026 von https://arxiv.org/abs/1712.05884

K R Prajwal, R. M. (23. August 2020). *A Lip Sync Expert Is All You Need for Speech to Lip Generation In The Wild.* Abgerufen am 14. März 2026 von https://arxiv.org/abs/2008.10010

Mohamed R. Shoaib, Z. W. (29. November 2023). *Deepfakes, Misinformation, and Disinformation in the Era of Frontier AI, Generative AI, and Large AI Models.* Abgerufen am 2026 von https://arxiv.org/abs/2311.17394v1

Qwen Team, Alibaba Cloud. (22. Januar 2026). *Qwen3-TTS Technical Report*. Abgerufen am 11. März 2026 von https://arxiv.org/abs/2601.15621

Resemble AI. (2025). *Chatterbox-TTS - GitHub*. Abgerufen am 8. März 2026 von https://github.com/resemble-ai/chatterbox

Shaoshu Yang, Z. K. (19. August 2025). *InfiniteTalk: Audio-driven Video Generation for Sparse-Frame Video Dubbing.* Abgerufen am 9. März 2026 von https://arxiv.org/abs/2508.14033

Singh, A. -K. (10. November 2023). *A Survey of AI Text-to-Image and AI Text-to-Video Generators.* doi:10.1109/AIRC57904.2023.10303174

Team Wan, A. W.-W. (26. März 2025). *Wan: Open and Advanced Large-Scale Video Generative Models.* Abgerufen am 9. März 2026 von https://arxiv.org/abs/2503.20314

Vineet Kumar Rakesh, S. M. (23. Juni 2025). *Advancing Talking Head Generation: A Comprehensive Survey of Multi-Modal Methodologies, Datasets, Evaluation Metrics, and Loss Functions.* Abgerufen am 12. März 2026 von https://arxiv.org/abs/2507.02900

Wenxuan Zhang, X. C. (22. November 2022). *SadTalker: Learning Realistic 3D Motion Coefficients for Stylized Audio-Driven Single Image Talking Face Animation.* Abgerufen am 14. März 2026 von https://arxiv.org/abs/2211.12194

Xu Tan, T. Q.-Y. (29. Juni 2021). *A Survey on Neural Speech Synthesis.* Abgerufen am 12. März 2026 von https://arxiv.org/abs/2106.15561

Yi Ren, Y. R.-Y. (22. Mai 2019). *FastSpeech: Fast, Robust and Controllable Text to Speech.* Abgerufen am 13. März 2026 von https://arxiv.org/abs/1905.09263

Zhuoyi Yang, J. T. (12. August 2024). *CogVideoX: Text-to-Video Diffusion Models with An Expert Transformer.* Abgerufen am 13. März 2026 von https://arxiv.org/abs/2408.06072

Anhang 1: API-Dokumentation

Anhang 1: Übersicht

Diese API ermöglicht es, automatisiert KI-generierte Videos zu erstellen. Dabei wird ein Text mit einer gewählten Stimme in ein Video umgewandelt (Infinite Talk + TTS-Audio-Suite/Qwen3-TTS via ComfyUI). Optional können dem fertigen Video ein Wasserzeichen oder ein Intro hinzugefügt werden.

Jobs werden in einer Warteschlange verwaltet und sequenziell verarbeitet. Der Verarbeitungsstatus lässt sich jederzeit über die API abfragen.

Anhang 1: Authentifizierung

Alle Endpunkte unter `/api`, mit Ausnahme des Download-Endpunkts, erfordern eine Authentifizierung über einen Bearer Token.

Der Token muss im Authorization-Header jeder Anfrage mitgesendet werden:

```
Authorization: Bearer <TOKEN>
```

Bei fehlendem oder ungültigem Token antwortet die API mit HTTP 401 bzw. 403.

> **Hinweis:** Der Download-Endpunkt (`GET /api/job/{jobId}/download`) benötigt keine Authentifizierung. Dies kann in der `server.js` Datei bei Bedarf abgeändert werden.

Code Änderung zum hinzufügen einer Authentifizierung beim Download-Endpunkt

```
// Routes
app.use('/api', [downloadRouter]);
// ändern zu:
app.use('/api', [authMiddleware, downloadRouter]);
```

Anhang 1: Basis-URL

`http://<HOST>:<PORT>`

Standard-Konfiguration: `http://127.0.0.1:4000`

In der `.env`-Datei anpassbar:

```
// bei Docker default 0.0.0.0
HOST="127.0.0.1"
PORT=4000
```

Anhang 1: Endpunkte

Anhang 1: Status

Prüft, ob der Server erreichbar ist. Keine Authentifizierung erforderlich.

```
> GET /status
```

Antwort (Status 200):

```
{
  "status": "online"
}
```

Anhang 1: ComfyUI Health-Check

Prüft, ob das ComfyUI-Backend erreichbar und funktionsfähig ist.

```
> GET /api/comfyui/health
```

Erfolgreiche Antwort (200):


```
{
  "status": "online",
  "comfyui": "up",
  "systemStats": {
    "system": {
      "os": "linux",
      "ram_total": 33232343040,
      "ram_free": 22835675136,
      "comfyui_version": "0.16.3",
      "required_frontend_version": "1.39.19",
      "installed_templates_version": "0.9.10",
      "required_templates_version": "0.9.10",
      "python_version": "3.12.13 (main, Mar  5 2026, 22:34:15) [GCC 15.2.1 20260
209]",
      "pytorch_version": "2.10.0+cu128",
      "embedded_python": false,
      "argv": [
        "main.py"
      ]
    },
    "devices": [
      {
        "name": "cuda:0 NVIDIA GeForce RTX 3090 : cudaMallocAsync",
        "type": "cuda",
        "index": 0,
        "vram_total": 25262817280,
        "vram_free": 20833304576,
        "torch_vram_total": 0,
        "torch_vram_free": 0
      }
    ]
  }
}
```


Fehler Antwort (200):

```
{
  "status": "offline",
  "comfyui": "down",
  "error": "Fehlermeldung" // z.B. "fetch failed"
}
```

Anhang 1: ComfyUI Output löschen

Löscht alle Medien im output Ordner von ComfyUI.

`> DELETE /api/comfyui/outputs`

Erfolgreiche Antwort (200):

```
{
  "success": true,
  "outputDir": "/home/user/haw-bachelor/comfy/output",
  "deletedCount": 6
}
```

Fehler Antwort (500):

Bei nicht konfigurierter `COMFYUI_PATH` environment Variable:

```
{
  "success": false,
  "error": "Missing COMFYUI_PATH environment variable."
}
```

Bei falsch konfigurierter COMFYUI_PATH environment Variable:

```
{
  "success": false,
  "error": "COMFYUI_PATH does not appear to be a valid ComfyUI directory."
}
```

Wenn aktuell ein Job verarbeitet wird:

```
{
  "success": false,
  "error": "Cleanup is not possible. Job xxx-uuid-xxx-xxx is running."
}
```

Anhang 1: Verfügbare Sprecher

Gibt eine Liste aller verfügbaren TTS-Sprecher zurück, die im Workflow verwendet werden können.

`> GET /api/comfyui/speakers`

Erfolgreiche Antwort (200):

```
[
  {
    "id": "eindeutige_sprecher_id",
    "label": "Kennzeichnung für das Frontend",
    "audioReferenceChatterbox": "voices_examples/bachelor/referenz.wav",
    "audioReferenceQwen3": "referenz.wav",
    "imageReference": "start_frame.png",
    "videoPrompt": "a news reporter is talking about news",
    "videoIntro": "intro.mp4",
    "width": 768,
    "height": 432
```

```
  },
  ...
]
```

Fehler Antwort (500):

```
{
  "error": "Can't read speaker json file at path/to/speaker.json",
}
```

Anhang 1: Job einreichen

Reicht einen neuen Video-Generierungsjob ein. Der Job wird in die Warteschlange aufgenommen und asynchron verarbeitet.

`> POST /api/job`

Request-Body: Workflow-Objekt als JSON

Beispiel:

```
{
  "text": "Hallo Welt, dies ist ein Test Text!",
  "speaker": "eindeutige_sprecher_id", // entspricht der "id" aus /api/comfyui/speakers
  "comfyWorkflow": "qwen3", // oder "chatterbox"
  "vramPatch": false,
  "addWatermark": true,
  "addIntro": false
}
```

Erfolgreiche Antwort (202 Accepted):

```
{
  "message": "Job submitted",
  "jobId": "5a8a8b2c-4a90-4d49-adbb-e18dc209398a" // UUIDv4
}
```

Fehler Antwort (400 Bad Request):

```
{
  "error": "Invalid workflow data",
  "details": "Fehlerbeschreibung"
}
```

Fehler Antwort (500 Server Error):

```
{
  "error": "Failed to save job",
  "details": "Fehlerbeschreibung"
}
```

Anhang 1: Job-Status abfragen

Gibt den aktuellen Status und alle Informationen zu einem bestimmten Job zurück.

`> GET /api/job/{jobId}`

Parameter:

Parameter	Typ	Beschreibung
`jobId`	String	ID des Jobs (UUIDv4)

Erfolgreiche Antwort (200):

```
{
  "jobId": "5a8a8b2c-4a90-4d49-adbb-e18dc209398a",
  "job": { ... } // job Objekt
}
```

Das job-Objekt ist unter Anhang 1: Job-Objekt beschrieben.

Fehler Antwort (404):

```
{
  "message": "Job not found!"
}
```

Anhang 1: Alle Jobs abfragen

Gibt eine Liste aller Jobs in der Datenbank zurück, sortiert nach Erstellungszeitpunkt.

```
> GET /api/jobs
```

Erfolgreiche Antwort (200):

```
[
  {
    "jobId": "5a8a8b2c-4a90-4d49-adbb-e18dc209398a",
    "comfyPromptId": "43e67089-add2-49c8-b65f-b181957b2fb4",
    "statusComfy": "done",
    "statusPostProcessing": "done",
    "running": 0,
    "workflowJson": "{\"speaker\":\"eindeutige_sprecher_id\",\"text\":\"Mein Nam
e ist Max Mustermann und diese Stimme wurde mit Qwen3-TTS generiert.\",\"comfyWo
rkflow\":\"qwen3\",\"vramPatch\":true,\"addIntro\":true,\"addWatermark\":true}",
    "lastErrorStage": null,
    "lastErrorMessage": null,
    "createdAt": "2026-03-06 13:19:09",
    "updatedAt": "2026-03-06 13:26:53",
    "finishedAt": "2026-03-06 13:26:53",
    "workflow": {
      "speaker": "eindeutige_sprecher_id",
      "text": "Mein Name ist Max Mustermann und diese Stimme wurde mit Qwen3-TTS
generiert.",
      "comfyWorkflow": "qwen3",
      "vramPatch": true,
      "addIntro": true,
      "addWatermark": true
    }
  },
  ... // weitere Job-Objekte
]
```

Anhang 1: Aktiven Job abfragen

Gibt den aktuell in Verarbeitung befindlichen Job zurück.

```
> GET /api/jobs/active
```

Erfolgreiche Antwort (200):

Es wird ein Job-Objekt zurückgegeben (siehe Anhang 1: Job-Objekt).

```
{ ... }
```

Keinen aktiven Jobs (404):

```
{
  "message": "No active job"
}
```

Anhang 1: Job abbrechen

Bricht einen laufenden oder wartenden Job ab.

```
> POST /api/job/{jobId}/cancel
```

Parameter:

Parameter	Typ	Beschreibung
jobId	String	ID des Jobs (UUIDv4)

Erfolgreiche Antwort (200):

```
{
  "message": "Job interrupted.",
  "comfycancelled": true,
  "job": { ... } // job-Objekt
}
```

Wenn der Job bereits abgeschlossen, fehlgeschlagen oder abgebrochen war:

```
{
  "message": "Job is already finished, failed or cancelled.",
  "job": { ... } // job-Objekt
}
```

Fehler Antwort (500 Server Error):

```
{
  "error": "Failed to interrupt job."
}
```

Anhang 1: Job wiederholen

Markiert einen fehlgeschlagenen oder abgebrochenen Job zur erneuten Verarbeitung.

```
> POST /api/job/{jobId}/retry
```

Parameter:

Parameter	Typ	Beschreibung
`jobId`	String	ID des Jobs (UUIDv4)

Erfolgreiche Antwort (200):

```
{
  "message": "Job marked for retry.",
  "job": { ... } // job-Objekt
}
```

Bereits erfolgreich abgeschlossene Schritte (done) werden nicht erneut ausgeführt. Nur fehlgeschlagene oder abgebrochene Schritte werden zurückgesetzt.

Fehler Antwort (404):

```
{
  "message": "Job not found!"
}
```

Anhang 1: Job löschen

Löscht einen Job dauerhaft aus der Datenbank. Laufende Jobs können nicht gelöscht werden.

```
> DELETE /api/job/{jobId}
```

Parameter:

Parameter	Typ	Beschreibung
`jobId`	String	ID des Jobs (UUIDv4)

Erfolgreiche Antwort (200):

```
{
  "success": true,
  "jobId": "5a8a8b2c-4a90-4d49-adbb-e18dc209398a"
}
```

Fehler – Job läuft gerade (500):

```
{
  "success": false,
  "error": "Can't delete running job."
}
```

Fehler – Job nicht gefunden (404):

```
{
  "success": false,
  "message": "Job not found!"
}
```

Anhang 1: Video herunterladen

Lädt das fertiggestellte Video eines abgeschlossenen Jobs herunter.

```
> GET /api/job/{jobId}/download
```

Dieser Endpunkt erfordert keine Authentifizierung.

Parameter:

Parameter	Typ	Beschreibung
jobId	String	ID des Jobs (UUIDv4)

Erfolgreiche Antwort (200):

Die MP4-Datei wird als Download zurückgegeben.

Die zurückgegebene Datei hängt von den Workflow-Einstellungen ab:

Workflow-Einstellung	Zurückgegebene Datei
addIntro: true	Video mit Intro
addWatermark: true	Video mit Wasserzeichen
Keines von beiden	Rohvideo aus ComfyUI

Fehler – Datei wurde nicht gefunden (404):

```
{
  "message": "File not found!"
}
```

Fehler – Job nicht gefunden (404):

```
{
  "message": "Job not found!"
}
```

Anhang 1: Workflow-Objekt

Das Workflow-Objekt wird beim Einreichen eines Jobs im Request-Body übergeben.

Feld	Typ	Pflicht	Beschreibung
text	String	Ja	Der Text, der gesprochen und in das Video umgewandelt werden soll.
speaker	String	Ja	ID des gewünschten Sprechers. Verfügbare IDs: GET /api/comfyui/speakers
comfyWorkflow	String	Ja	Name der zu verwendenden ComfyUI Workflows. (chatterbox oder qwen3)
vramPatch	Boolean	Nein	Aktiviert einen VRAM-Optimierungs-Patch für Systeme mit begrenztem Videospeicher. Standard: false
addWatermark	Boolean	Nein	Fügt dem fertigen Video ein Wasserzeichen hinzu. Standard: false

Feld	Typ	Pflicht	Beschreibung
addIntro	Boolean	Nein	Fügt dem fertigen Video ein Intro voran. Standard: false

Beispiel (JSON-Objekt):

```
{
  "text": "Willkommen zu diesem KI-generierten Video.",
  "speaker": "de_speaker_1",
  "comfyWorkflow": "chatterbox",
  "vramPatch": false,
  "addWatermark": true,
  "addIntro": false
}
```

Anhang 1: Job-Objekt

Ein Job-Objekt hat folgende Felder:

Feld	Typ	Beschreibung
jobId	String	Eindeutige Job-ID (UUID)
comfyPromptId	String	Interne Prompt-ID von ComfyUI (wird nach Einreichung gesetzt)
statusComfy	String	Verarbeitungsstatus im ComfyUI-System (siehe Status-Werte)
statusPostProcessing	String	Verarbeitungsstatus der Nachbearbeitung (Wasserzeichen / Intro)
running	Integer	1 = Job wird gerade verarbeitet, 0 = Job ist nicht aktiv
workflow	Object	Das ursprünglich eingeschickte Workflow-Objekt
lastErrorStage	String	Stufe des letzten Fehlers: comfy oder postprocessing. null wenn kein Fehler aufgetreten ist.
lastErrorMessage	String	Fehlermeldung des letzten fehlgeschlagenen Verarbeitungsschritts. null wenn kein Fehler aufgetreten ist.
createdAt	String	Erstellungszeitpunkt
updatedAt	String	Letzter Aktualisierungszeitpunkt
finishedAt	String	Zeitpunkt der erfolgreichen Fertigstellung. null solange der Job noch nicht abgeschlossen ist.

Anhang 1: Status-Werte

Sowohl `statusComfy` als auch `statusPostProcessing` können folgende Werte annehmen:

Status	Beschreibung
queued	Job wartet auf Verarbeitung.
processing	Job wird aktuell verarbeitet.
done	Dieser Schritt wurde erfolgreich abgeschlossen.
failed	Dieser Schritt ist fehlgeschlagen.

Status	Beschreibung
cancelled	Job wurde manuell abgebrochen.
retry	Job ist für einen erneuten Versuch markiert.

Ein Job gilt als vollständig abgeschlossen, wenn sowohl `statusComfy` als auch `statusPostProcessing` den Wert done haben. Das Video kann dann heruntergeladen werden.

Anhang 2: Konfiguration des Projekts

Anhang 2: Assets vorbereiten

Alle Assets liegen im Verzeichnis `assets/` und werden von dort durch die Skripte `copy-assets-backend.sh` und `copy-assets-comfy.sh` in die ComfyUI-Verzeichnisstruktur und der Backend-Verzeichnisstruktur kopiert.

Anhang 2: Audio Assets

Audio-Dateien dienen als Referenz-Stimme für die TTS-Generierung. Diese müssen folgende technische Anforderungen erfüllen:

Eigenschaft	Anforderung
Format	.wav
Kanäle	Mono
Sample Rate	22.050 Hz
Länge	ca. 5–30 Sekunden

Konvertierung mit FFmpeg:

```
ffmpeg -i quelle.wav -ac 1 -ar 22050 sprecher.wav
```

Besonderheit: Engine chatterbox

Wird die Audio-Engine chatterbox verwendet, muss zusätzlich zur .wav-Datei eine gleichnamige .reference.txt-Datei existieren, die den gesprochenen Inhalt der Referenz-Audiodatei als Klartext enthält.

Beispiel:

```
assets/audio/sprecher.wav
assets/audio/sprecher.reference.txt
```

Inhalt von `sprecher.reference.txt`: “Dies ist der gesprochene Text aus der Sprecher Audio Datei.”

Hinweis: Für die Engine qwen3 wird keine `.reference.txt`-Datei benötigt.

Anhang 2: Bild Assets

Referenzbilder für Infinite Talk müssen im Verzeichnis `assets/images/` abgelegt werden. Das Bild wird als Startframe für die Videogenerierung verwendet.

- Format: .png oder .jpg
- Auflösung: Sollte der konfigurierten width × height des Sprechers entsprechen (siehe Sprecher konfigurieren)

Anhang 2: Video Assets

Intro-Videos für die Nachbearbeitung müssen im Verzeichnis assets/videos/ abgelegt werden.

> **Achtung:** Das Intro-Video muss **dieselbe Aspect Ratio** besitzen wie die konfigurierten width/height-Werte des Sprechers (z. B. 16:9 -> 768×432). Stimmt die Aspect Ratio nicht überein, schlägt das Zusammenfügen von Intro und generiertem Video mit ffmpeg fehl.

Anhang 2: Assets kopieren

Das Kopieren der Assets ist auf zwei separate Skripte aufgeteilt, eines für ComfyUI und eines für das Backend.

`copy-assets-comfy.sh:` Kopiert Audio- und Bild-Assets in die ComfyUI-Verzeichnisstruktur. Das Skript erwartet, dass ComfyUI im Unterordner `comfy/` des aktuellen Verzeichnisses vorhanden ist.

Verwendung:

```
chmod +x copy-assets-comfy.sh
./copy-assets-comfy.sh
```

Quelle	Ziel	Beschreibung
assets/audio/	comfy/custom_nodes/tts_audio_suite/voices_examples/bachelor/	Referenz-Stimmen für chatterbox
assets/audio/	comfy/input/	Referenz-Stimmen für qwen3
assets/images/	comfy/input/	Referenzbilder für Infinite Talk

`copy-assets-backend.sh:` Kopiert Video-Assets in die Backend-Verzeichnisstruktur. Das Skript erwartet, dass das Backend im Unterordner backend/ des aktuellen Verzeichnisses vorhanden ist.

Verwendung:

```
chmod +x copy-assets-backend.sh
./copy-assets-backend.sh
```

Quelle	Ziel	Beschreibung
assets/videos/	backend/data/assets/	Intro-Videos

Anhang 2: Sprecher konfigurieren

Die verfügbaren Sprecher werden in der Datei `./data/speakers.json` im Backend-Verzeichnis definiert.

Beispielstruktur:

```
{
    "speakers": [
        {
            "id": "eindeutige_sprecher_id",
            "label": "Moderator Tagesschau (männlich - Jens Riewa)",
            "audioReferenceChatterbox": "voices_examples/bachelor/referenz.wav",
            "audioReferenceQwen3": "referenz.wav",
            "imageReference": "start_frame.png",
            "videoPrompt": "a news reporter is talking about news",
            "videoIntro": "intro.mp4",
            "width": 768,
            "height": 432
        }
    ]
}
```

Feldbeschreibungen:

Feld	Pflicht	Beschreibung
id	Ja	Eindeutiger Bezeichner des Sprechers. Darf **nicht** doppelt vorkommen.
label	Ja	Anzeigename im Frontend. Frei wählbar.
audioReferenceChatterbox	Ja	Pfad zur Referenz-Audiodatei für die Engine chatterbox. **Nur den Dateinamen anpassen**, der Pfadpräfix voices_examples/bachelor/ muss gleich bleiben.
audioReferenceQwen3	Ja	Dateiname der Referenz-Audiodatei für die Engine qwen3 (nur Dateiname, kein Pfad).
imageReference	Ja	Dateiname des Referenzbilds für Infinite Talk.
videoPrompt	Ja	Englischsprachiger Prompt zur Steuerung der Videogenerierung.
videoIntro	Nein	Dateiname des Intro-Videos. Kann auf "" gesetzt werden, falls kein Intro existiert.
width	Ja	Breite des generierten Videos in Pixel. Muss zur Aspect Ratio des Intro-Videos passen.
height	Ja	Höhe des generierten Videos in Pixel. Muss zur Aspect Ratio des Intro-Videos passen.

Hinweis zu `audioReferenceChatterbox`: Der Pfad muss immer mit `voices_examples/bachelor/` beginnen. Nur der eigentliche Dateiname (z.B. `mein_sprecher.wav`) wird angepasst.

Anhang 2: Workflows konfigurieren

Die verfügbaren ComfyUI Workflows werden im Backend in der Datei `backend/data/workflows.js` definiert.

Ein Workflow beschreibt:

- welche ComfyUI Workflow Datei verwendet wird
- welche Nodes im Workflow verändert werden können
- welche Features der Workflow unterstützt (z.B. VRAM-Patch oder Video-Prompt)

Das Backend nutzt diese Konfiguration, um automatisch Werte wie Text, Sprachreferenz oder Bild-Inputs in den ComfyUI Workflow einzusetzen.

Beispielstruktur:

```
  qwen3: {
    id: 'qwen3',
    label: 'Qwen3 + Infinite Talk',
    workflowPath: path.join(__dirname, '..', 'data', 'workflows', 'qwen3-video-a
pi-v2.json'),
    supports: {
      audioText: true,
      audioVoiceReference: true,
      imageReference: true,
      videoPrompt: true,
      videoResolution: true,
      vramPatch: true,
    },
    nodes: {
      audioText: {
        nodeId: '31',
        inputKey: 'text'
      },
      audioVoiceReference: {
        nodeId: '29',
        inputKey: 'audio',
        speakerField: 'audioReferenceQwen3',
      },
      imageReference: {
        nodeId: '23',
        inputKey: 'image'
      },
      videoPrompt: {
        nodeId: '20',
        inputKey: 'positive_prompt'
      },
      videoResolution: {
        nodeId: '11',
        widthKey: 'width',
        heightKey: 'height',
      },
    },
  }
```

Feldbeschreibungen:

Feld	Pflicht	Beschreibung
id	Ja	Eindeutige Kennung des Workflows. Wird vom Backend und Frontend verwendet.
label	Ja	Anzeigename des Workflows im Frontend.
workflowPath	Ja	Pfad zur ComfyUI Workflow-JSON-Datei.
supports	Ja	Gibt an, welche Features der Workflow unterstützt.
nodes	Nein	Konfiguration der Nodes, deren Inputs durch das Backend gesetzt werden können.

Feld	Beschreibung
audioText	Der Workflow unterstützt einen Textinput für die Sprachgenerierung.
audioVoiceReference	Der Workflow unterstützt eine Referenzstimme für Voice-Cloning.
imageReference	Der Workflow verwendet ein Referenzbild für die Videogenerierung.
videoPrompt	Der Workflow unterstützt einen Textprompt zur Steuerung der Videogenerierung.
videoResolution	Der Workflow erlaubt das Setzen von width und height.
vramPatch	Der Workflow kann mit einem VRAM-Patch modifiziert werden.

Hinweis: Der VRAM-Patch funktioniert nur bei Workflows, die das Infinite Talk Video Modell verwenden.

Feld	Beschreibung
nodeId	ID der Node im ComfyUI Workflow
inputKey	Name des Inputs innerhalb der Node
speakerField	Feld aus speakers.json, dessen Wert verwendet wird

Anhang 2: Backend konfigurieren

Das Backend befindet sich im Unterordner `backend/` des Projekts.

Kopiere die `.env.example` Beispieldatei und passe die Werte an:

```
cp .env.example .env
```

.env-Parameter:

Variable	Standard	Beschreibung
COMFY_UI_BACKEND	http://127.0.0.1:8188	Host / URL der ComfyUI-Instanz.
COMFYUI_PATH	-	Absoluter Pfad zur ComfyUI installation, z.B. /home/user/haw-bachelor/comfy
AUTH_TOKEN	secure-token	Token zur Absicherung der API.

Variable	Standard	Beschreibung
HOST	127.0.0.1	Netzwerkschnittstelle, auf der das Backend hört.
PORT	4000	Port des Backends.
CALLBACK_URL	-	Optionale URL, an die der Backend-Worker Status Callbacks per POST sendet.
CALLBACK_TRIGGERS	-	Kommagetrennte Liste der Statuswerte, bei denen ein Callback gesendet wird, z.B. done.

Beispiel .env:

```
COMFY_UI_BACKEND="http://127.0.0.1:8188"
COMFYUI_PATH="/home/user/haw-bachelor/comfy"
AUTH_TOKEN="mein-sicheres-token-123"
HOST="127.0.0.1"
PORT=4000
CALLBACK_URL="https://example.com/job-callback"
CALLBACK_TRIGGERS="done,failed"
```

Hinweis: Die Umgebungsvariablen `COMFY_UI_BACKEND` und `AUTH_TOKEN` können direkt in der docker-compose.yml angepasst werden. HOST und PORT sollten in der Docker-Umgebung nicht angepasst werden.
Hinweis: Die Umgebungsvariable `COMFYUI_PATH` wird nur durch die Cleanup Funktion verwendet, um den Inhalt aus dem outputs Ordner über die API löschen zu können.

Anhang 2: Callback-Konfiguration

Das Backend kann nach Abschluss oder Fehler eines Jobs optional einen HTTP-Callback auslösen. Die Implementierung verwendet `CALLBACK_URL` als Zieladresse und filtert die auszulösenden Status über `CALLBACK_TRIGGERS`.

- Ohne `CALLBACK_URL` wird kein Callback gesendet.
- `CALLBACK_TRIGGERS` ist optional. Wenn die Variable leer ist, wird bei allen Status ein Callback gesendet, die im Code aktiv ausgelöst werden.
- Für `done` ergänzt das Backend automatisch `downloadUrl` mit dem relativen Pfad `/api/job/<jobId>/download`.
- Für failed wird zusätzlich ein `reason` mitgesendet.

Beispiel: Payload bei Erfolg

```
{
  "jobId": "abc123",
  "status": "done",
  "downloadUrl": "/api/job/abc123/download"
}
```

Beispiel: Payload bei Fehler

```
{
  "jobId": "abc123",
  "status": "failed",
  "reason": "Tried to download video of completed job abc123, however something went wrong!"
}
```

Anhang 2: Frontend konfigurieren

Das Frontend befindet sich im Unterordner `frontend/` des Projekts.

Kopiere die `.env.example` Beispieldatei und passe die Werte an:

```
cp .env.example .env
```

.env-Parameter:

Variable	Standard	Beschreibung
BACKEND_ENDPOINT	http://127.0.0.1:4000	URL des laufenden Backends.
AUTH_TOKEN	secure-token	Token zur Authentifizierung mit dem Backend.
HOST	127.0.0.1	Netzwerkschnittstelle, auf der das Frontend hört.
PORT	3000	Port des Frontends.

Beispiel .env:

```
BACKEND_ENDPOINT="http://localhost:4000"
AUTH_TOKEN="mein-sicheres-token-123"
HOST="127.0.0.1"
PORT=3000
```

> **Hinweis:** Die Umgebungsvariablen `BACKEND_ENDPOINT` und `AUTH_TOKEN` können direkt in der `docker-compose.yml` angepasst werden. `HOST` und `PORT` sollten in der Docker-Umgebung nicht angepasst werden.

Anhang 3: Datenbank Dokumentation

Das Backend verwendet das „better-sqlite3“ Package als SQLite-Datenbank. Die Datenbankdatei liegt unter: `/data/queue.db`

Liegt die Datenbank noch nicht vor, wird diese automatisch erstellt.

Anhang 3: Tabelle „jobs“

Spalte	Typ	Pflicht	Standard	Beschreibung
jobId	TEXT	Ja	–	Primärschlüssel, eindeutige Job-ID (z.B. UUID)
comfyPromptId	TEXT	Nein	NULL	Prompt-ID aus ComfyUI, nach dem Absenden befüllt
statusComfy	TEXT	Ja	'queued'	Status der ComfyUI-Verarbeitung (siehe Status-Werte)
statusPostProcessing	TEXT	Ja	'queued'	Status der Nachbearbeitung (siehe Status-Werte)
running	INTEGER	Ja	0	1 = Job wird gerade ausgeführt, 0 = inaktiv
workflowJson	TEXT	Nein	NULL	Workflow als serialisiertes JSON (z.B. vom Frontend)
lastErrorStage	TEXT	Nein	NULL	Stufe, in der der letzte Fehler aufgetreten ist (comfy oder postprocessing)
lastErrorMessage	TEXT	Nein	NULL	Fehlermeldung des letzten fehlgeschlagenen Verarbeitungsschritts
createdAt	TEXT	Ja	datetime('now')	Zeitstempel der Erstellung
updatedAt	TEXT	Ja	datetime('now')	Zeitstempel der letzten Änderung (automatisch)
finishedAt	TEXT	Nein	NULL	Zeitstempel der erfolgreichen Fertigstellung (gesetzt nach abgeschlossener Nachbearbeitung)

Anhang 3: Status-Werte

Sowohl `statusComfy` als auch `statusPostProcessing` verwenden dieselben möglichen Status-Werte:

Status	Bedeutung
queued	Job wartet auf Verarbeitung
processing	Job wird aktuell verarbeitet
retry	Job soll erneut versucht werden
done	Verarbeitung erfolgreich abgeschlossen
failed	Verarbeitung fehlgeschlagen
cancelled	Verarbeitung abgebrochen

Anhang 3: Trigger

```
CREATE TRIGGER IF NOT EXISTS jobs_updatedAt
AFTER UPDATE ON jobs
BEGIN
  UPDATE jobs SET updatedAt = datetime('now') WHERE jobId = NEW.jobId;
END;
```

Dieser Trigger setzt `updatedAt` automatisch auf das aktuelle Datum + Uhrzeit, sobald ein Datensatz in der jobs-Tabelle geändert wird.

Anhang 3: API-Funktionen

Das Datenbank Interface unter `backend/src/db/jobs.js` bietet öffentliche Funktionen zur Verwendung in der Applikation.

Anhang 3: createJob(options)

Legt einen neuen Job in der Datenbank an.

Parameter:

Name	Typ	Pflicht	Standard	Beschreibung
jobId	string	Ja	-	Eindeutige Job-ID
comfyPromptId	string	Nein	null	ComfyUI Prompt-ID
statusComfy	string	Nein	'queued'	Initialer ComfyUI-Status
statusPostProcessing	string	Nein	'queued'	Initialer Nachbearbeitungs-Status
workflow	object	Nein	null	Workflow-Objekt (wird zu JSON serialisiert)

Rückgabe: Das neu erstellte Job-Objekt.

Beispiel Code:

```
const { createJob } = require('../db/jobs');
const uuid = require('uuid');

const job = createJob({
  jobId: uuid.v4(), // neue UUID
  workflow: { "text": "Hallo Welt, dies ist ein Test Text!", [...] },
});
```

Anhang 3: updateJob(jobId, patch)

Aktualisiert einzelne Felder eines bestehenden Jobs. Felder, die im patch-Objekt nicht angegeben werden, bleiben unverändert.

Parameter:

Name	Typ	Beschreibung
jobId	string	ID des zu aktualisierenden Jobs
patch	object	Zu ändernde Felder

Folgende Felder können im `patch`-Objekt übergeben werden:

Feld	Typ	Beschreibung
comfyPromptId	string	ComfyUI Prompt-ID
statusComfy	string	Neuer ComfyUI-Status
statusPostProcessing	string	Neuer Nachbearbeitungs-Status
running	boolean	Aktiv-Flag des Jobs
workflow	object	Workflow-Objekt (wird zu JSON serialisiert)
lastErrorStage	string	Stufe des letzten Fehlers (comfy oder postprocessing)
lastErrorMessage	string	Fehlermeldung des letzten fehlgeschlagenen Schritts

Beispiel Code:

```
const jobsDb = require('../db/jobs');

jobsDb.updateJob(job.jobId, {
  comfyPromptId: comfyResponse.prompt_id,
  statusComfy: 'processing',
});
```

Anhang 3: getJob(jobId)

Gibt einen einzelnen Job anhand seiner ID zurück.

```
const job = getJob('5a8a8b2c-4a90-4d49-adbb-e18dc209398a');
// job.workflow ist ein JS-Objekt (nicht JSON string)
```

Rückgabe: Job-Objekt oder null, wenn nicht gefunden.

Anhang 3: getJobs()

Gibt alle Jobs sortiert nach createdAt aufsteigend zurück.

```
const jobs = getJobs();
```

Anhang 3: getRunningJob()

Gibt den aktuell laufenden Job zurück (`running = 1`). Es kann und darf nur ein Job aktiv sein.

```
const running = getRunningJob();
```

Rückgabe: Job-Objekt oder null, wenn kein Job läuft.

Anhang 3: getNextPendingJob()

Gibt den ältesten ausstehenden Job zurück, der noch nicht läuft und bei dem `statusComfy` oder `statusPostProcessing` den Wert `queued, processing` oder `retry` hat.

```
const next = getNextPendingJob();
```

Anhang 3: setRunning(jobId, running)

Setzt das running-Flag eines Jobs.

```
// Job als "aktiv" gesetzt
setRunning('5a8a8b2c-4a90-4d49-adbb-e18dc209398a', true);
// Job als "inaktiv" gesetzt
setRunning('5a8a8b2c-4a90-4d49-adbb-e18dc209398a', false);
```

Anhang 3: deleteJob(jobId)

Löscht einen Job dauerhaft aus der Datenbank.

```
const deleted = deleteJob('5a8a8b2c-4a90-4d49-adbb-e18dc209398a');
// true = erfolgreich gelöscht, false = Job nicht gefunden
```

Anhang 3: setFinishedTimeStamp(jobId)

Setzt das Feld `finishedAt` auf den aktuellen Zeitstempel. Wird intern vom Worker aufgerufen, sobald die Nachbearbeitung erfolgreich abgeschlossen wurde.

```
setFinishedTimeStamp('5a8a8b2c-4a90-4d49-adbb-e18dc209398a');
```

Rückgabe: Das aktualisierte Job-Objekt.

Anhang 3: Funktion hydrateJob(row)

Eine interne Hilfsfunktion, die den JSON string `workflowJson` automatisch zu einem JavaScript-Objekt unter dem Namen `workflow` deserialisiert. Alle öffentlichen Funktionen geben bereits "hydrierte" Objekte zurück.

Anhang 4: Aufsetzen des Projekts

Anhang 4: Voraussetzungen

Folgende Software und Hardware muss auf dem System installiert sein, bevor mit der Einrichtung begonnen werden kann:

Abhängigkeit	Mindestversion / Mindestanforderung	Hinweis
Git	beliebig	Zum installieren von ComfyUI
Node.js	22	Für Backend und Frontend
npm	- (hängt von Node.js ab)	Mit Node.js mit installieren
Python	3.12 (3.13 könnte instabil sein)	Für ComfyUI
ffmpeg	beliebig	Für Videoverarbeitung
Docker & Docker Compose	beliebig	Nur bei Docker-basiertem Setup*
NVIDIA GPU	24GB<= VRAM mit CUDA 12<= Support	Benötigt für KI-Inferenz
SSD / HDD	50GB<= Freier Speicherplatz	Benötigt für KI-Modelle
RAM (Arbeitsspeicher)	32GB<=	Benötigt für KI-Inferenz

Unterstützte Betriebssysteme: Arch Linux (und Arch-Derivate) sowie Ubuntu / Debian basierte Systeme.
Das InstallationsScript erkennt das Betriebssystem automatisch und verwendet den entsprechenden Paketmanager (pacman bzw. apt).

*bei Docker-basiertem Setup wird kein Node.js / npm benötigt.

Anhang 4: ComfyUI Installieren

ComfyUI bildet das KI-Backend des Projekts. Es verarbeitet die Workflows zur Video- und Audiogenerierung.

Die Installation kann automatisiert per Script oder manuell erfolgen.

Die Installation dauert etwa 15-30 Minuten, je nach Internet Anbindung.

Anhang 4: Automatisierte Installation (Skript)

Das mitgelieferte Script `install-comfy.sh` übernimmt folgende Schritte vollständig:

- Aktualisierung des Paketmanagers
- Installation von System-Abhängigkeiten (Git, ffmpeg, Python 3.12, PortAudio)
- Erstellen einer Python-Virtual-Environment
- Installation von comfy-cli und ComfyUI selbst

- Installation aller benötigten Custom Nodes:
  - `tts_audio_suite`
  - `comfyui-qwen3-tts`
  - `comfyui-qwen-asr`
  - `ComfyUI-WanVideoWrapper`
  - `comfyui-videohelpersuite`
  - `rgthree-comfy`
- Download aller benötigten Modelle für Infinite Talk (Wan 2.1, CLIP Vision, wav2vec2, …)
- Kopieren von Audio- und Bild-Assets in die richtigen Verzeichnisse

Ausführung:

```
# Script ausführbar machen und starten
chmod +x install-comfy.sh
./install-comfy.sh
```

Das Script fragt an zwei Stellen nach einer Bestätigung:

1. Vor Beginn der Installation
2. Vor der ComfyUI-Installation, da hier die GPU-Auswahl interaktiv erfolgt, bitte die passende GPU-Option auswählen

Hinweis für Arch Linux: Wird Python 3.12 nicht auf dem System gefunden, installiert das Script es automatisch aus dem AUR. Dafür muss ein AUR-Helper (paru oder yay) installiert sein.

Das ComfyUI-Installationsverzeichnis ist standardmäßig `./comfy` (relativ zum Arbeitsverzeichnis beim Aufruf des Scripts). Dies kann im Script angepasst werden.

Anhang 4: Manuelle Installation

Wie folgt kann die Installation manuell durchgeführt werden.

Schritt 1: Systemabhängigkeiten installieren

Arch Linux:

```
# System updaten & mirrors aktualisieren
sudo pacman -Syu
# Dependencies installieren
sudo pacman -S git ffmpeg portaudio

# Python 3.12 ggf. aus dem AUR installieren (z. B. via paru)
paru -S python312
# oder mit yay
yay -S python312
```

Ubuntu / Debian:

```
sudo apt update
sudo apt install -y git ffmpeg python3 python3-venv python3-pip portaudio19-dev
```

Schritt 2: Python Virtual Environment erstellen

```
# Arch Linux
python3.12 -m venv comfy-env

# Ubuntu / Debian
python3 -m venv comfy-env

# Wichtig, da wir unsere Abhängigkeiten
# in der virtuellen Umgebung installieren wollen
source comfy-env/bin/activate
```

Schritt 3: comfy-cli und ComfyUI installieren

```
pip install --upgrade pip
pip install --upgrade comfy-cli

# ComfyUI installieren, GPU-Auswahl während der Installation beachten!
comfy --workspace ./comfy install
```

Schritt 4: ComfyUI Nodes installieren

```
# TTS / Audio
comfy --workspace ./comfy node install tts_audio_suite
comfy --workspace ./comfy node install comfyui-qwen3-tts
comfy --workspace ./comfy node install comfyui-qwen-asr

# Video
comfy --workspace ./comfy node install ComfyUI-WanVideoWrapper
comfy --workspace ./comfy node install comfyui-videohelpersuite
comfy --workspace ./comfy node install rgthree-comfy
```

Schritt 5: Modelle herunterladen (für Infinite Talk)

```
# Wan 2.1 Video-Modell (GGUF)
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/city96/Wan2.1-I2V-14B-480P-gguf/resolve/main/wan
2.1-i2v-14b-480p-Q4_K_M.gguf?download=true" \
  --relative-path models/diffusion_models \
  --filename wan2.1-i2v-14b-480p-Q4_K_M.gguf

# LightX2V LoRA
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/Kijai/WanVideo_comfy/resolve/main/Lightx2v/light
x2v_I2V_14B_480p_cfg_step_distill_rank64_bf16.safetensors?download=true" \
  --relative-path models/loras \
  --filename lightx2v_I2V_14B_480p_cfg_step_distill_rank64_bf16.safetensors

# InfiniteTalk-Modell
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/Kijai/WanVideo_comfy_GGUF/resolve/main/InfiniteT
alk/Wan2_1-InfiniteTalk_Single_Q4_K_M.gguf?download=true" \
  --relative-path models/diffusion_models \
  --filename Wan2_1-InfiniteTalk_Single_Q4_K_M.gguf

# CLIP Vision
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/Comfy-Org/Wan_2.1_ComfyUI_repackaged/resolve/mai
n/split_files/clip_vision/clip_vision_h.safetensors?download=true" \
```

```
  --relative-path models/clip_vision \
  --filename clip_vision_h.safetensors

# Wav2Vec2 (Audio-Encoder)
mkdir -p ./comfy/models/wav2vec2
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/Kijai/wav2vec2_safetensors/resolve/main/wav2vec2
-chinese-base_fp16.safetensors?download=true" \
  --relative-path models/wav2vec2 \
  --filename wav2vec2-chinese-base_fp16.safetensors

# Text-Encoder (UMT5)
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/ALGOTECH/WanVideo_comfy/resolve/main/umt5-xxl-en
c-bf16.safetensors?download=true" \
  --relative-path models/clip \
  --filename umt5-xxl-enc-bf16.safetensors

# VAE
comfy --workspace ./comfy model download \
  --url "https://huggingface.co/Comfy-Org/Wan_2.1_ComfyUI_repackaged/resolve/mai
n/split_files/vae/wan_2.1_vae.safetensors?download=true" \
  --relative-path models/vae \
  --filename wan_2.1_vae.safetensors
```

Schritt 6: Assets kopieren

```
# Audio-Assets für TTS Audio Suite
mkdir -p ./comfy/custom_nodes/tts_audio_suite/voices_examples/bachelor
cp -r assets/audio/* ./comfy/custom_nodes/tts_audio_suite/voices_examples/bachel
or/

# Audio-Assets in den Input-Ordner
cp -r assets/audio/* ./comfy/input/

# Bild-Assets in den Input-Ordner
cp -r assets/images/* ./comfy/input/

# Video-Assets in den Backend Asset-Ordner
cp -r assets/videos/* ./backend/data/assets/
```

Anhang 4: ComfyUI starten

```
# Script ausführbar machen und starten
chmod +x launch-comfy.sh
./launch-comfy.sh
```

oder manuell (identisch zu dem was im Script steht):

```
# env Variable anlegen mit dem Pfad zum Comy Verzeichnis
INSTALL_DIRECTORY="$PWD/comfy"

# in virtuelle Python Umgebung wechseln
source comfy-env/bin/activate

# comfy starten
comfy --workspace "$INSTALL_DIRECTORY" launch
```

Anhang 4: Backend aufsetzen

Das Backend ist eine Node.js/Express.js-Anwendung, die als API-Schnittstelle zwischen dem Nutzer und ComfyUI fungiert.

> **Konfiguration** ist in einer separaten Konfigurationsdokumentation (Anhang 2) beschrieben.

Anhang 4: Manuell (Node.js)

```
# In das Backend-Verzeichnis wechseln
cd backend

# Abhängigkeiten installieren
npm install

# Backend starten
npm start

# alternativ Entwicklungsumgebung starten
npm run dev
```

Das Backend läuft standardmäßig auf Port **4000**.

Anhang 4: Mit Docker Compose

Im Backend-Verzeichnis liegt eine `docker-compose.yml`, mit der das Backend containerisiert gestartet werden kann.

```
cd backend

# --build -> kann ausgelassen werden, wenn das Docker Image bereits gebaut wurde
# -d -> im Hintergrund laufen lassen (detached mode)
docker compose up --build -d
```

Das Docker Image wird automatisch mithilfe der Dockerfile gebaut. Der Container läuft im host-Netzwerkmodus, damit es ComfyUI auf 127.0.0.1 direkt erreichen kann. Dies muss ggf. angepasst werden, wenn die Systeme auf verschiedenen Hosts laufen. Die Anwendung ist anschließend auf Port 4000 des Hosts erreichbar.

Anhang 4: Frontend aufsetzen

Das Frontend ist eine eigenständige Node.js/Express.js-Anwendung und kommuniziert mit dem Backend über die API.

> **Konfiguration** ist in einer separaten Konfigurationsdokumentation (Anhang 2) beschrieben.

Anhang 4: Manuell (Node.js)

```
# In das Frontend-Verzeichnis wechseln
cd frontend

# Abhängigkeiten installieren
npm install

# Frontend starten
npm start

# alternativ Entwicklungsumgebung starten
npm run dev
```

Anhang 4: Mit Docker Compose

Im Frontend-Verzeichnis liegt ebenfalls eine docker-compose.yml:

```
cd frontend

docker compose up --build -d
```

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

Automatisierung der KI-basierten Generierung von audiovisuellen Medieninhalten

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum	Unterschrift