

BACHELORARBEIT

Automatisierung der KI-basierten Generierung längerer Audioinhalte – Konzeption und Realisierung eines lokalen Systems mit Voice Cloning

vorgelegt am 25. September 2026
Tom Kalberg

Erstprüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Sune Maute

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG**

Fakultät Informatik und Digitale Gesellschaft
Finkenau 35
20081 Hamburg

Zusammenfassung

Diese Bachelorarbeit untersucht, wie KI-basierte Sprachsynthese lokal und automatisiert zur Erzeugung längerer Audioinhalte eingesetzt werden kann. Hierfür wurde ein modularer Python-Prototyp entwickelt, der die beiden Modelle Qwen3-TTS und VoxCPM2 über eine gemeinsame Schnittstelle integriert und außerdem Voice Cloning auf Basis einer Referenzaufnahme unterstützt. Längere Eingabetexte werden automatisch segmentiert, abschnittsweise generiert und anschließend zu einer Audiodatei zusammengefügt. Eine optionale Lautheitsnormalisierung reduziert dabei Unterschiede zwischen den einzelnen Segmenten. Die Anwendung ist über eine API-Schnittstelle und eine Weboberfläche nutzbar. Die Evaluation auf einer NVIDIA GeForce RTX 5080 Grafikkarte zeigt, dass beide Modelle schneller als Echtzeit generieren können. Gleichzeitig wurden bei längeren unsegmentierten Ausgaben qualitative Verschlechterungen beobachtet. Die segmentierte Verarbeitung konnte diese Effekte begrenzen, wobei die Lautheitsnormalisierung insbesondere bei Qwen3-TTS hörbare Unterschiede zwischen einzelnen Segmenten reduzierte.

Abstract

This bachelor's thesis investigates how AI-based speech synthesis can be used locally and automatically to generate longer audio content. To this end, a modular Python prototype was developed which integrates the two models, Qwen3-TTS and VoxCPM2, via a common interface and also supports voice cloning based on a reference recording. Longer input texts are automatically segmented, generated in sections and then merged into a single audio file. An optional loudness normalisation feature reduces differences between the individual segments. The application can be accessed via an API and a web interface. Evaluation on an NVIDIA GeForce RTX 5080 graphics card shows that both models can generate output faster than real time. At the same time, a decline in quality was observed in longer, unsegmented outputs. Segmented processing was able to limit these effects, with loudness normalisation reducing audible differences between individual segments, particularly in the case of Qwen3-TTS.

Inhaltsverzeichnis

Verwendete Hilfsmittel	6
Abkürzungsverzeichnis	7
Abbildungsverzeichnis	8
Tabellenverzeichnis	8
1 Einleitung.....	9
1.1 Ausgangslage und Motivation	9
1.2 Zielsetzung und Abgrenzung	10
1.3 Forschungsfrage und Vorgehensweise.....	11
2 Grundlagen KI-basierter Sprachsynthese	11
2.1 Neuronale Sprachsynthese und Voice Cloning.....	11
2.2 Herausforderungen bei längeren Sprachinhalten	12
2.3 Audiotechnische Grundlagen für die Weiterverarbeitung	13
3 Technologische Einordnung und Auswahl der TTS-Modelle	14
3.1 Anforderungen an lokal betriebene TTS-Systeme.....	14
3.2 Qwen3-TTS.....	15
3.3 VoxCPM2	15
3.4 Betrachtete Alternativen und Auswahlbegründung	16
3.5 Verwandte Ansätze zur Long-Form-Sprachgenerierung	17
4 Entwurf des Gesamtsystems	18
4.1 Anforderungen und Randbedingungen	18
4.2 Architektur und Komponenten.....	20
4.3 Einheitliche TTS-Abstraktion	22
4.4 Long-Form-Verarbeitung.....	22
5 Realisierung des Prototyps	23
5.1 Entwicklungs- und Laufzeitumgebung	24
5.2 Integration der TTS-Engines und Sprecherprofile.....	25
5.3 Segmentierung und Long-Form-Pipeline.....	26
5.4 Lautheitanpassung, Zusammenführung und Export	27
5.5 API, Jobverarbeitung und Weboberfläche	28

5.6	Installation und reproduzierbarer Systemaufbau	32
6	Evaluation.....	33
6.1	Teststrategie und Versuchsaufbau	33
6.2	Technische Messgrößen.....	34
6.3	Long-Form-Stabilität und Audioqualität	35
6.4	Ergebnisse	36
6.5	Einordnung der Ergebnisse	40
7	Grenzen des entwickelten Ansatzes.....	41
8	Ethische und rechtliche Aspekte des Voice Cloning	43
9	Fazit und Ausblick.....	45
9.1	Fazit und Beantwortung der Forschungsfrage	45
9.2	Ausblick	46
	Literaturverzeichnis.....	49
	Anhang 1: Evaluationsmaterial	52
1.1	Testtext des Kurzbenchmarks	52
1.2	Testtext der Long-Form-Evaluation.....	52
1.3	Segmentierung des Long-Form-Testtexts.....	53
1.4	Rohdaten der technischen Messungen	55
1.5	Lautheitswerte der Long-Form-Segmente	56
	Anhang 2: Ausgewählte Implementierungsdetails.....	56
2.1	Textsegmentierung.....	56
2.2	Clipping-sichere Lautheitsnormalisierung	57
2.3	Zentraler Long-Form-Workflow	59
2.4	FIFO-Jobverwaltung	60
	Anhang 3: Anwendungsschnittstelle	61
3.1	Generierung mit Preset-Sprecher	61
3.2	Generierung mit benutzerdefiniertem Sprecher	62
3.3	Abfrage eines abgeschlossenen Jobs.....	63
	Anhang 4: Installationsanleitung.....	64
4.1	Voraussetzungen	64

4.2	Einrichtung von WSL	64
4.3	Prüfung der NVIDIA GPU	65
4.4	Installation von Miniconda	65
4.5	Automatisierte Installation des Prototyps	66
4.6	Start der Anwendung	66
4.7	Bedienung der Weboberfläche.....	67
4.8	Erzeugte Audiodateien und Neustart	67
4.9	Häufige Probleme	68
Anhang 5: Softwarestand und Reproduzierbarkeit.....		69
5.1	Bereitgestellter Softwarestand	69
5.2	Verwendete Modellrevisionen	69
5.3	Dateien zur Reproduzierbarkeit	70
Eigenständigkeitserklärung		71

Verwendete Hilfsmittel

Im Rahmen dieser Bachelorarbeit wurde generative KI unterstützend eingesetzt. Die Nutzung beschränkte sich auf die vereinzelte Unterstützung bei der Implementierung einzelner Funktionen innerhalb des Quellcodes sowie auf sprachliche Überarbeitung und Rechtschreibprüfung des schriftlichen Teils. Die Konzeption, technische Entscheidungen, Evaluation und inhaltliche Ausarbeitung der Arbeit wurden eigenständig durchgeführt.

Abkürzungsverzeichnis

AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
FIFO	First In, First Out
GPU	Graphics Processing Unit
HAW	Hochschule für Angewandte Wissenschaften
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
LU	Loudness Unit
LUFS	Loudness Units relative to Full Scale
LTS	Long Term Support
RTF	Real-Time Factor
TTS	Text-to-Speech
UTF-8	8-bit Unicode Transformation Format
UUID	Universally Unique Identifier
VRAM	Video Random Access Memory
WAV	Waveform Audio File Format
WSL	Windows Subsystem for Linux

Abbildungsverzeichnis

Abbildung 1: Schematischer Aufbau des entwickelten Long-Form-TTS-Systems, Quelle: Eigene Darstellung	21
Abbildung 2: Vereinfachte Darstellung der Projektstruktur, Quelle: Eigene Darstellung	25
Abbildung 3: Gekürzter Ausschnitt der gemeinsamen Schnittstelle, Quelle: Eigener Quellcode.....	25
Abbildung 4: Weboberfläche des entwickelten Prototyps, Quelle: Eigener Screenshot.....	31
Abbildung 5: Integrierte Lautheit der sieben unveränderten Long-Form-Segmente, Quelle: Eigene Darstellung	38

Tabellenverzeichnis

Tabelle 1: Auswahl betrachteter TTS-Modelle, Quelle: Eigene Darstellung.....	16
Tabelle 2: Wesentliche Systemanforderungen, Quelle: Eigene Darstellung.....	19
Tabelle 3: Wesentliche Komponenten der entwickelten Umgebung, Quelle: Eigene Darstellung	24
Tabelle 4: Wichtigste HTTP-Endpunkte der Anwendung, Quelle: Eigene Darstellung	29
Tabelle 5: Technische Messwerte der TTS-Modelle, Quelle: Eigene Darstellung	36
Tabelle 6: Lautheitswerte der segmentierten Long-Form-Generierung, Quelle: Eigene Darstellung...	37
Tabelle 7: Erfüllung der wesentlichen Systemanforderungen, Quelle: Eigene Darstellung	40

1 Einleitung

Fortschritte im Bereich generativer KI haben in den vergangenen Jahren dazu geführt, dass synthetisch erzeugte Sprache eine zunehmend natürliche Qualität erreicht. Moderne Text-to-Speech-Systeme sind nicht mehr ausschließlich auf vortrainierte Stimmen beschränkt, sondern können mit Hilfe von Voice Cloning auch neue Sprecher auf Basis kurzer Referenzaufnahmen nachbilden (Xie et al., 2025). Dadurch ergeben sich verschiedenste Einsatzmöglichkeiten, u. a. für automatisierte Sprachbeiträge, Podcasts, Nachrichtentexte oder andere Formen längerer Audioinhalte.

Die praktische Nutzung solcher Systeme besteht jedoch nicht allein darin, einen Text an ein Sprachmodell zu übergeben und anschließend eine Audiodatei zu erhalten. Besonders bei längeren Eingabetexten wird man mit zusätzlichen technischen Herausforderungen konfrontiert, z. B. hinsichtlich der Konsistenz und Kohärenz über längere Sprachabschnitte hinweg (Pan et al., 2026). Für das in dieser Arbeit entwickelte System ergeben sich daraus u. a. Anforderungen an eine geeignete Segmentierung des Textes, eine konsistente Sprecherstimme über mehrere Abschnitte hinweg, möglichst gleichbleibende Lautheit sowie die automatische Zusammenführung der einzelnen Sprachsegmente. Gleichzeitig sollen die verwendeten Modelle lokal betrieben werden können, um unabhängig von kommerziellen Cloud-Diensten zu bleiben und die Verarbeitung auf eigener Hardware durchzuführen.

Diese Arbeit beschäftigt sich daher mit der Frage, wie sich aktuelle KI-basierte Verfahren zur Sprachsynthese zu einem modularen lokalen System verbinden lassen, welches längere Audioinhalte automatisiert erzeugt und optional Voice Cloning unterstützt. Im Mittelpunkt steht dabei nicht die Entwicklung eines eigenen TTS-Modells, sondern die technische Konzeption und Einbindung bestehender Open-Source-Modelle innerhalb einer praktisch nutzbaren Anwendung.

1.1 Ausgangslage und Motivation

Der Ausgangspunkt dieser Arbeit ist ein Forschungsprojekt eines Doktoranden an der HAW Hamburg. Im Rahmen dieses Projekts wird ein System entwickelt, das Nachrichtentexte automatisiert umformuliert und dabei unterschiedliche Formen von politischem Framing berücksichtigt. Um die Wirkung dieser Varianten nicht ausschließlich anhand der Texte untersuchen zu können, sollen daraus verschiedene Medieninhalte mit einem virtuellen Sprecher erzeugt werden. Für diesen Anwendungsfall ist insbesondere eine automatisierte Verarbeitung relevant, da eine größere Anzahl unterschiedlicher Textvarianten erzeugt und weiterverarbeitet werden soll.

Eine vorherige Bachelorarbeit von Alexandre Kaul beschäftigte sich bereits mit der Automatisierung dieser Generierung auf audiovisueller Ebene. Dabei wurde ein System entwickelt, das Text-to-Speech und KI-basierte Videogenerierung miteinander verbindet und daraus kurze Sprechervideos erzeugt

(Kaul, 2026). Der ursprüngliche Ansatz der aktuellen Bachelorarbeit bestand zuerst darin, das System von Herrn Kaul zu erweitern und zusätzliche KI-Modelle in die bestehende Architektur zu integrieren.

Bei der praktischen Vorbereitung kristallisierte sich jedoch heraus, dass die vollständige Audio-Video-Pipeline aufgrund der hohen Anforderungen an den verfügbaren Grafikspeicher auf der für diese Arbeit vorhandenen Consumer-Hardware nicht lokal betrieben werden konnte. In Abstimmung mit dem Doktoranden wurde der Schwerpunkt dieser Arbeit deshalb auf die reine Sprachgenerierung verlagert, die ohne Videoanteil deutlich geringere Anforderungen an den Grafikspeicher stellt. Gleichzeitig ergab sich daraus die Möglichkeit, einen bisher nur als Teil der Videopipeline betrachteten Bereich gezielt weiterzuentwickeln: die automatisierte Erzeugung längerer zusammenhängender Audioinhalte.

1.2 Zielsetzung und Abgrenzung

Ziel dieser Arbeit ist die Konzeption und Realisierung eines lokal ausführbaren Systems zur automatisierten Generierung längerer Sprachinhalte. Das System soll auf handelsüblicher Consumer-Hardware betrieben werden können und mehrere aktuelle Open-Source-TTS-Modelle über eine gemeinsame Schnittstelle ansprechen können. Neben der Sprachgenerierung mit vordefinierten Sprecherprofilen soll auch die Verwendung eigener Referenzaufnahmen für Voice Cloning möglich sein.

Ein besonderer Schwerpunkt liegt auf der Verarbeitung längerer Eingabetexte. Diese sollen automatisiert in geeignete Abschnitte unterteilt, einzeln synthetisiert und anschließend wieder zu einer zusammenhängenden Audiodatei zusammengefügt werden. Zusätzliche Verarbeitungsschritte innerhalb der Anwendung sollen dabei u. a. eine Angleichung der Lautheit und die Konfiguration von Pausen zwischen den erzeugten Sprachsegmenten ermöglichen. Für die praktische Nutzung soll die Generierung sowohl über eine Programmierschnittstelle als auch über eine einfache Weboberfläche zugänglich sein.

Die Arbeit konzentriert sich auf die Integration und Verarbeitung bestehender TTS-Modelle. Die Entwicklung oder das Training eines eigenen Sprachmodells soll hierbei nicht Bestandteil des Projekts sein. Ebenso wird die zuvor betrachtete Videogenerierung, wie bereits erläutert, bewusst ausgeklammert. Der Schwerpunkt liegt auf der lokalen Ausführung mit NVIDIA-Grafikkarten und den im Rahmen der Arbeit ausgewählten Modellen Qwen3-TTS und VoxCPM2, auf die später noch genau eingegangen wird. Das Ziel ist hierbei nicht ein vollständiger Vergleich aller verfügbaren TTS-Systeme, sondern die Untersuchung, wie sich ausgewählte Modelle in eine gemeinsame Pipeline zur Generierung von Long-Form-Audio integrieren lassen.

1.3 Forschungsfrage und Vorgehensweise

Aus der beschriebenen Ausgangslage und Zielsetzung ergibt sich folgende Forschungsfrage:

„Wie kann ein lokales, modulares KI-System zur automatisierten Generierung längerer Audioinhalte mit optionalem Voice Cloning auf Consumer-Hardware umgesetzt werden?“

Zur Beantwortung dieser Frage werden zunächst die technischen Grundlagen moderner Sprachsynthese sowie die besonderen Herausforderungen bei der Erzeugung längerer Audioinhalte betrachtet. Anschließend werden geeignete und lokal ausführbare TTS-Modelle betrachtet und die Auswahl der im Prototyp verwendeten Modelle entsprechend begründet.

Darauf aufbauend werden die Architektur und die zentralen Verarbeitungsschritte des entwickelten Systems beschrieben. Dazu gehören insbesondere die einheitliche Anbindung der TTS-Modelle, die Segmentierung längerer Texte, die Verarbeitung der erzeugten Sprachsegmente sowie die Bereitstellung der Generierung über eine entsprechende Schnittstelle und eine Weboberfläche. Abschließend wird der entwickelte Prototyp anhand technischer und qualitativer Kriterien evaluiert. Die Ergebnisse der Evaluation werden anschließend im Hinblick auf die Forschungsfrage sowie die Grenzen des gewählten Ansatzes und mögliche Weiterentwicklungen eingeordnet.

2 Grundlagen KI-basierter Sprachsynthese

Für die Entwicklung eines Systems zur automatisierten Erzeugung längerer Audioinhalte sind einige grundlegende Konzepte moderner Sprachsynthese relevant. Dieses Kapitel beschränkt sich auf diejenigen Aspekte, die für das Verständnis der später verwendeten TTS-Modelle und der entwickelten Pipeline notwendig sind. Dazu gehören die grundlegende Funktionsweise neuronaler Text-to-Speech-Systeme, die Konditionierung auf eine bestimmte Sprecherstimme, die besonderen Anforderungen bei längeren Sprachinhalten sowie einige ausgewählte audiatechnische Grundlagen der Weiterverarbeitung.

2.1 Neuronale Sprachsynthese und Voice Cloning

Text-to-Speech-Systeme wandeln geschriebenen Text automatisiert in gesprochene Sprache um. Moderne Systeme verwenden hierfür heutzutage überwiegend neuronale Modelle, die aus großen Mengen aufgezeichneter Sprache und zugehöriger Textdaten lernen können, sprachliche Zusammenhänge abzubilden. Während frühere Verfahren häufig aus mehreren voneinander getrennten Verarbeitungsschritten bestanden, können aktuelle Modelle einen großen Teil dieser Verarbeitung innerhalb eines gemeinsamen Systems abbilden (Xie et al., 2025).

Neben dem Inhalt des gesprochenen Textes muss ein TTS-System bei der Generierung auch Eigenschaften wie Aussprache, Sprechtempo, Betonung und Sprechcharakter berücksichtigen. Bei Systemen mit mehreren möglichen Stimmen werden dafür zusätzliche Sprecherinformationen in den Generierungsprozess einbezogen. Diese Sprecherkonditionierung ermöglicht es dem Modell, denselben Text mit unterschiedlichen Stimmen zu erzeugen (Xie et al., 2025).

Eine besondere Form dieser Konditionierung ist das sogenannte Voice Cloning. Dabei wird eine Aufnahme einer Person zur Hand genommen, mit der das Modell bisher nicht explizit trainiert wurde, und als Referenz für die Sprachgenerierung verwendet. Aus dieser Aufnahme werden charakteristische Merkmale der Stimme abgeleitet und auf neu zu generierende Sprache übertragen. Moderne Systeme können dies teilweise bereits anhand kurzer Referenzaufnahmen durchführen, ohne dass für jede neue Stimme ein erneutes Training des gesamten Modells erforderlich ist. Dieses Verfahren wird häufig als Zero-Shot Voice Cloning beziehungsweise Zero-Shot Text-to-Speech bezeichnet (Azzuni & El Saddik, 2025). Abhängig vom verwendeten Modell kann neben der Referenzaufnahme auch das zugehörige Transkript benötigt werden. Die konkrete Art der Sprecherkonditionierung unterscheidet sich daher teilweise deutlich zwischen den verschiedenen TTS-Systemen. Für eine gemeinsame Verarbeitung mehrerer Modelle muss diese unterschiedliche Handhabung entsprechend berücksichtigt werden.

2.2 Herausforderungen bei längeren Sprachinhalten

Die Generierung längerer Sprachinhalte stellt andere Anforderungen an ein TTS-System als die Synthese einzelner Sätze oder kurzer Textabschnitte. Mit zunehmender Länge einer Generierung wird es schwieriger, Eigenschaften wie Sprechercharakter, Sprechtempo, Betonung und allgemeine Verständlichkeit über den gesamten Verlauf hinweg konsistent zu halten. Auch aktuelle Systeme zeigen bei längeren Ausgaben weiterhin deutliche Schwächen hinsichtlich der langfristigen sprachlichen und akustischen Konsistenz (Pan et al., 2026).

Eine Möglichkeit, diese Problematik zu umgehen, besteht darin, lange Eingabetexte nicht als Ganzes zu verarbeiten, sondern in kürzere Abschnitte aufzuteilen. So kann die eigentliche Sprachgenerierung jeweils mit einem begrenzten Textsegment durchgeführt werden. Hierdurch entsteht jedoch eine neue Herausforderung, da die separat erzeugten Audioteile anschließend wieder zu einem möglichst zusammenhängenden Ergebnis vereint werden müssen. Unterschiede zwischen einzelnen Segmenten können sich z. B. in Lautstärke, Sprechweise oder Pausen bemerkbar machen und dadurch den Eindruck einer durchgehenden Aufnahme maßgeblich beeinträchtigen.

Die Segmentierung des Textes läuft dabei nicht beliebig ab. Zu kurze Abschnitte erhöhen die Anzahl der Übergänge zwischen separat erzeugten Sprachsegmenten, während zu lange Abschnitte erneut die bereits erläuterten Nachteile längerer Einzelgenerierungen mit sich bringen können. Gleichzeitig sollten sprachlich zusammengehörige Elemente möglichst erhalten bleiben, damit Sätze nicht an ungeeigneten

Stellen getrennt werden. Die Verarbeitung längerer Texte erfordert daher einen Kompromiss zwischen ausreichend kurzen Generierungsabschnitten und einem möglichst natürlichen sprachlichen Zusammenhang.

2.3 Audiotechnische Grundlagen für die Weiterverarbeitung

Die von einem TTS-Modell erzeugte Sprache muss für die weitere Verarbeitung in einem einheitlichen Format behandelt werden. Eine grundlegende Eigenschaft ist hierbei die Abtastrate: Sie gibt an, wie viele Abtastwerte eines Audiosignals pro Sekunde gespeichert werden und wird in Hertz angegeben. Werden mehrere Audioschnipsel miteinander verbunden, müssen sie mit derselben Abtastrate vorliegen oder zuvor entsprechend umgewandelt werden. Für reine Sprachinhalte ist zudem oftmals ein einkanales Mono-Audiosignal ausreichend.

Neben dem technischen Format spielt die Lautstärke der erzeugten Segmente ebenfalls eine wichtige Rolle. Der Spitzenpegel eines digitalen Audiosignals beschreibt den höchsten auftretenden Signalwert und ist insbesondere dann relevant, wenn man Übersteuerungen beziehungsweise Clipping vermeiden möchte. Für die wahrgenommene Lautheit einer Aufnahme ist der Spitzenpegel allein jedoch nur bedingt aussagekräftig, da die menschliche Lautheitswahrnehmung von deutlich mehr Faktoren beeinflusst wird (International Telecommunication Union, 2023).

Zur vergleichbaren Bewertung der wahrgenommenen Lautheit kann die gemessene Lautheit in LUFS (Loudness Units relative to Full Scale) angegeben werden. Diese Einheit berücksichtigt das Audiosignal über einen bestimmten Zeitraum und ermöglicht dadurch eine bessere Einschätzung der wahrgenommenen Lautheit als die alleinige Betrachtung des Spitzenpegels (European Broadcasting Union, 2023; International Telecommunication Union, 2023). Wenn bei der Sprachsynthese einzelne Sprachsegmente unabhängig voneinander erzeugt werden, können sich deren Lautheitswerte teilweise deutlich unterscheiden. Durch eine nachträgliche Anpassung des Signalpegels lassen sich solche Unterschiede reduzieren, wobei gleichzeitig darauf geachtet werden muss, dass durch die Verstärkung keine Übersteuerung entsteht.

Die einzelnen Sprachsegmente müssen schließlich zu einem gemeinsamen Audiofile zusammengesetzt werden. Dabei beeinflussen auch die Übergänge zwischen den Segmenten das Endergebnis, denn insbesondere die Dauer der Pause zwischen zwei Segmenten kann bestimmen, ob die zusammengesetzte Sprache natürlich oder unterbrochen wirkt. Abtastrate, Lautheit, Spitzenpegel und das Einfügen von Pausen bilden damit die wesentlichen audiotechnischen Grundlagen für die spätere Weiterverarbeitung der erzeugten Sprachsegmente.

3 Technologische Einordnung und Auswahl der TTS-Modelle

Auf Grundlage der zuvor beschriebenen Anforderungen wurden verschiedene lokal ausführbare TTS-Modelle betrachtet. Entscheidend war dabei nicht nur die erreichbare Sprachqualität - für den vorgesehenen Anwendungsfall mussten die Modelle auch hinsichtlich ihrer Hardwareanforderungen, der Unterstützung von Voice Cloning sowie ihrer technischen Einbindung in eine gemeinsame Anwendung geeignet sein.

Die folgenden Abschnitte beschreiben zuerst die daraus abgeleiteten Auswahlkriterien. Anschließend werden die beiden eingesetzten Modelle Qwen3-TTS und VoxCPM2 vorgestellt und ihre Auswahl gegenüber den betrachteten Alternativen begründet.

3.1 Anforderungen an lokal betriebene TTS-Systeme

Eine grundlegende Anforderung war die vollständige lokale Ausführung der Sprachgenerierung. Die Nutzung eines externen Cloud-Dienstes soll dabei für den gesamten Generierungsprozess nicht erforderlich sein. Ein geeignetes Modell für diese Arbeit muss also lokal ausführbar und über eine programmierbare Schnittstelle in die geplante Anwendung integrierbar sein.

Gleichzeitig war die verfügbare Hardware ebenfalls ein wichtiges Auswahlkriterium. Als Entwicklungsplattform stand eine einzelne NVIDIA-Grafikkarte (GeForce RTX 5080) mit 16 GB Grafikspeicher zur Verfügung, deshalb waren TTS-Modelle, die eine deutlich darüber hinausgehende Hardware voraussetzen, für diese Arbeit ungeeignet. Die Zielsetzung bestand nicht darin, möglichst große bzw. komplexe TTS-Modelle auszuführen, sondern Modelle zu wählen, die auf einer leistungsfähigen, aber dennoch dem Consumer-Bereich zugehörigen Grafikkarte zuverlässig eingesetzt werden können.

Für den geplanten Einsatz mussten die Modelle außerdem die Generierung in deutscher Sprache und eine Form des Voice Cloning unterstützen. Die Erzeugung einer neuen Stimme sollte auf Basis einer kurzen Referenzaufnahme mit zugehörigem Referenztext möglich sein, ohne dass im Voraus ein sprecherspezifisches Modell trainiert werden muss. Da mehrere TTS-Modelle innerhalb einer Anwendung genutzt werden sollen, bietet sich außerdem eine direkte Einbindung über Python an.

Zusammengefasst ergaben sich damit die Anforderungen eines lokalen Betriebs auf einer einzelnen Consumer-GPU, deutscher Sprachgenerierung, referenzbasierten Voice Clonings sowie einer Integration der verschiedenen Modelle in ein gemeinsames System. Auf dieser Grundlage wurden Qwen3-TTS und VoxCPM2 als TTS-Engines ausgewählt.

3.2 Qwen3-TTS

Qwen3-TTS ist eine Familie mehrsprachiger Text-to-Speech-Modelle. Diese Modelle unterstützen insgesamt zehn Sprachen, inkl. Deutsch, und bieten je nach Version bzw. Ausführung unterschiedliche Funktionen. Für das im Zuge dieser Arbeit entwickelte System wird die Variante **Qwen3-TTS-12Hz-1.7B-Base** verwendet. Sie umfasst rund 1,7 Milliarden Modellparameter und ist insbesondere für referenzbasiertes Voice Cloning vorgesehen (Hu et al., 2026). Für das Voice Cloning werden eine kurze Referenzaufnahme sowie der zugehörige Referenztext an das Modell übergeben und anhand dieser Grundlage kann Qwen3-TTS anschließend neue Texte mit den Merkmalen der Referenzstimme generieren. Laut den Entwicklern ist die Sprecherkonditionierung bereits mit kurzen Referenzaufnahmen von wenigen Sekunden möglich. Die verwendete Base-Variante erzeugt außerdem Audiodaten mit einer Abtastrate von 24 kHz und kann direkt über ein Python-Paket in eine lokale Anwendung eingebunden werden (Hu et al., 2026).

Für diese Arbeit ist Qwen3-TTS insbesondere aufgrund der Kombination aus deutschsprachiger Sprachsynthese, Voice Cloning und lokaler Ausführbarkeit relevant. Die konkrete Einbindung in das entwickelte System sowie das Verhalten des Modells bei längeren Texten werden in den späteren Kapiteln betrachtet.

3.3 VoxCPM2

VoxCPM2 ist ein mehrsprachiges Text-to-Speech-Modell von OpenBMB. Das Modell umfasst rund zwei Milliarden Parameter und unterstützt 30 Sprachen, ebenfalls inkl. Deutsch. Im Unterschied zu vielen tokenbasierten TTS-Systemen verfolgt VoxCPM2 einen tokenizerfreien Ansatz und kombiniert autoregressive Sprachmodellierung mit einem diffusionsbasierten Verfahren zur Audioerzeugung (Zhou et al., 2026). Für das Voice Cloning kann auch hier eine kurze Referenzaufnahme mit entsprechendem Referenztext verwendet werden. Die generierte Sprache wird anschließend mit einer Abtastrate von 48 kHz ausgegeben (Zhou et al., 2026). Auch VoxCPM2 kann über ein entsprechendes Python-Paket lokal installiert und in eigene Anwendungen integriert werden.

Für den entwickelten Prototyp stellt VoxCPM2 damit eine zweite TTS-Engine dar, die zwar dieselben grundlegenden Anforderungen wie Qwen3-TTS erfüllt, sich jedoch deutlich bzgl. Modellarchitektur, Ressourcenbedarf und Generierungsverhalten von diesem unterscheidet.

3.4 Betrachtete Alternativen und Auswahlbegründung

Im Rahmen der Modellauswahl wurden neben Qwen3-TTS und VoxCPM2 noch weitere TTS-Modelle betrachtet. Entscheidend für diese Arbeit waren hierbei die im Voraus definierten Anforderungen. Die folgende Tabelle zeigt eine Auswahl der Modelle, die im Rahmen der Voruntersuchung näher betrachtet wurden, sowie eine kurze Begründung, warum das jeweilige Modell ausgewählt wurde oder nicht.

Tabelle 1: Auswahl betrachteter TTS-Modelle, Quelle: Eigene Darstellung

Modell	Relevante Eigenschaften	Einordnung
Qwen3-TTS	Deutsch, Voice Cloning, 1.7B Parameter, 24 kHz, Python-Integration, geringer VRAM-Bedarf	Ausgewählt: geringe Hardwareanforderungen und einfache Integration
VoxCPM2	Deutsch, Voice Cloning, ca. 2B Parameter, 48 kHz, tokenizerfreier Ansatz, Python-Integration	Ausgewählt: Qualitativ interessante Ergänzung zu Qwen, mit höherem VRAM-Bedarf
FireRedTTS2	Deutsch, Zero-Shot Voice Cloning, Long-Form-Generierung, Streaming- und Dialogfunktionen	Nicht gewählt: Stärker auf Dialog und Streaming ausgerichtet
MOSS-TTS-Nano	Deutsch, Voice Cloning, ca. 0,1B Parameter, 48 kHz, automatische Long-Text-Segmentierung, sehr geringer Ressourcenbedarf	Nicht gewählt: Zum Auswahlzeitpunkt größere Unsicherheit bzgl. Qualität und Reife
CosyVoice3	Deutsch, Zero-Shot Voice Cloning, 0,5B-Modell, Streaming, Steuerung von Sprechstil und Geschwindigkeit	Nicht gewählt: Komplexere Installation und Laufzeitumgebung ohne klaren Vorteil
OmniVoice	Deutsch, Zero-Shot Voice Cloning, sehr große Sprachabdeckung, zusätzliche Voice-Design-Funktionen	Nicht gewählt: kein klarer Vorteil gegenüber den ausgewählten Modellen

Technische Modellangaben nach Hu et al. (2026), Zhou et al. (2026), FireRedTeam (2025), OpenMOSS Team (2026), QwenAudio (o. D.) sowie Zhu et al. (2026). Die Auswahlentscheidung stellt eine eigene Bewertung im Rahmen dieser Arbeit dar.

Die Entscheidung für Qwen3-TTS und VoxCPM2 beruhte nicht darauf, ein einzelnes Modell als grundsätzlich besser einzustufen. Stattdessen wurden zwei Modelle ausgewählt, die die grundlegenden Anforderungen dieser Arbeit erfüllen und sich bzgl. ihrer technischen Eigenschaften voneinander unterscheiden. Beide unterstützen deutschsprachige Sprachsynthese, referenzbasiertes Voice Cloning und eine direkte lokale Einbindung über Python.

Qwen3-TTS wurde insbesondere aufgrund seines vergleichsweise geringen Ressourcenbedarfs und der unkomplizierten Integration ausgewählt. VoxCPM2 tritt dabei als ein technisch anders aufgebautes Modell mit höherem Ressourcenbedarf auf und ergänzt Qwen3-TTS dadurch sinnvoll für die Untersuchung einer gemeinsamen, modellunabhängigen Long-Form-Pipeline. Einige Voruntersuchungen bestätigten außerdem, dass beide Modelle auf der verfügbaren Referenzhardware ausführbar sind.

Die Auswahl stellt damit einen Kompromiss zwischen Hardwareanforderungen, Voice-Cloning-Funktionalität, deutschsprachiger Generierung und technischer Integrierbarkeit dar. Durch die Verwendung der zwei unterschiedlichen Modelle ist es außerdem möglich, zu untersuchen, inwieweit die entwickelte Verarbeitungspipeline unabhängig von den Besonderheiten eines einzelnen TTS-Modells eingesetzt werden kann.

3.5 Verwandte Ansätze zur Long-Form-Sprachgenerierung

Neben den betrachteten TTS-Modellen existieren natürlich bereits Ansätze, die sich mit der Generierung längerer Sprachinhalte beschäftigen. Einige Open-Source-Projekte verfolgen dabei einen ähnlichen Ansatz: Voice Studio integriert z. B. mehrere lokal ausführbare TTS-Engines, darunter Qwen3-TTS und VoxCPM2, innerhalb einer gemeinsamen Anwendung und unterstützt ebenfalls Voice Cloning sowie einen lokalen Betrieb (Raza, 2026). LocalText2Voice und Text2Audio richten sich stärker auf die Erzeugung längerer Inhalte wie Hörbücher aus und verarbeiten umfangreiche Texte segmentweise, bevor die erzeugten Audioteile anschließend wieder zusammengesetzt werden (Stiffl, 2026; mooja77, 2026). Andere Arbeiten wie Deep Dubbing oder Audiobook-CC konzentrieren sich ebenfalls auf die automatische Erzeugung längerer Hörbuchinhalte mit mehreren Sprechern sowie auf kontextsensitive und emotionale Sprachgenerierung (Dai et al., 2025; Liu et al., 2025).

Die vorliegende Arbeit unterscheidet sich von diesen Ansätzen vor allem durch ihren Fokus auf eine modellunabhängige Verarbeitungspipeline. Anstatt ein neues TTS-Modell oder ein spezielles Verfahren zur Long-Form-Generierung zu entwickeln, werden bestehende lokal ausführbare Modelle über eine gemeinsame Schnittstelle integriert. Die Verarbeitung längerer Inhalte wird hierbei außerhalb der eigentlichen TTS-Modelle durch Segmentierung, Nachbearbeitung und anschließende Zusammenführung realisiert. Dadurch können verschiedenste TTS-Modelle innerhalb desselben Systems verwendet werden.

Der eigene Beitrag dieser Arbeit liegt daher nicht in der Entwicklung eines neuen TTS-Modells, sondern in der Konzeption, Implementierung und Evaluation einer gemeinsamen Verarbeitungsschicht für bereits bestehende TTS-Modelle. Dazu gehören insbesondere die einheitliche TTS-Abstraktion, die automatische Long-Form-Segmentierung, die segmentübergreifende und clipping-sichere Lautheitsangleichung sowie der Zugriff über eine lokale API mit sequentieller Jobverwaltung. Außerdem wird untersucht, wie sich zwei unterschiedlich aufgebaute TTS-Modelle innerhalb derselben Pipeline bei längeren deutschsprachigen Eingaben verhalten und welche Grenzen sich aus der segmentweisen Verarbeitung ergeben.

4 Entwurf des Gesamtsystems

Ausgehend von den beschriebenen Anforderungen und den ausgewählten TTS-Modellen wurde eine Systemarchitektur entworfen, die die Sprachgenerierung von den anderen Verarbeitungsschritten trennt. Im Mittelpunkt steht dabei ein modularer Aufbau, bei dem unterschiedliche TTS-Modelle in dieselbe Systemarchitektur eingebunden werden können. Die Verarbeitung längerer Texte, die Nachbearbeitung der erzeugten Segmente sowie die Bereitstellung über eine Anwendungsschnittstelle sollen möglichst unabhängig vom jeweils eingesetzten Sprachmodell erfolgen. Die folgenden Abschnitte beschreiben zunächst die Anforderungen und Randbedingungen des Gesamtsystems, anschließend werden dessen grundlegende Architektur, die Abstraktion der TTS-Modelle sowie das Verarbeitungskonzept für längere Texte erläutert.

4.1 Anforderungen und Randbedingungen

Eine zentrale Anforderung an das Gesamtsystem ist die vollständig lokale Verarbeitung aller für die Sprachgenerierung benötigten Daten: Texte und Referenzaufnahmen sollen für die Generierung nicht an externe Dienste übertragen werden müssen und gleichzeitig muss die Anwendung auf einer einzelnen Consumer-Grafikkarte ausführbar bleiben, da keine Server- oder Multi-GPU-Infrastruktur vorausgesetzt werden soll.

Das System soll längere deutschsprachige Texte automatisch verarbeiten können, ohne dass die Texteingabe vorher händisch in einzelne Abschnitte aufgeteilt werden muss. Die einzelnen TTS-Modelle sollen dabei austauschbar bleiben, sodass die Long-Form-Verarbeitung nicht speziell auf die interne Funktionsweise eines einzelnen Modells zugeschnitten ist.

Neben vordefinierten Sprecherprofilen soll die Verwendung eigener Referenzaufnahmen möglich sein, um selbst einen Sprecher erstellen zu können. Die dafür notwendigen Informationen, insbesondere Re-

ferenzaudio und -text, müssen daher an die jeweilige TTS-Engine weitergegeben werden können. Unterschiede in den konkreten Modellaufrufen sollen dabei für die übrigen Komponenten des Systems keine Rolle spielen.

Für die erzeugten Sprachsegmente wird außerdem eine gemeinsame Weiterverarbeitung benötigt. Dazu gehören eine optionale Angleichung der Lautheit, die Zusammenführung der Segmente mit konfigurierbaren Pausen und der Export des fertigen Ergebnisses. Als Ausgabeformat wird WAV verwendet, um entsprechende Codec-Abhängigkeiten zu vermeiden und die Audiodaten verlustfrei ohne Kompression für spätere Analysen oder eventuelle Weiterverarbeitung auszugeben.

Zuletzt soll das System nicht ausschließlich über ein Terminal bzw. die Kommandozeile bedienbar sein. Die Generierung muss über eine Programmierschnittstelle möglich sein und zusätzlich über eine einfache Weboberfläche ausgeführt werden können. Da die Sprachgenerierung GPU-intensiv ist und nur eine einzelne Grafikkarte zur Verfügung steht, müssen gleichzeitig eingereichte Generierungsaufträge nacheinander statt gleichzeitig verarbeitet werden können, um das System bzw. den Grafikkartenspeicher nicht zu überlasten. Für die spätere Bewertung des Prototyps werden die beschriebenen Randbedingungen zu acht wesentlichen Systemanforderungen zusammengefasst. Diese Anforderungen bilden die Grundlage für die abschließende Bewertung in Abschnitt 6.5.

Tabelle 2: Wesentliche Systemanforderungen, Quelle: Eigene Darstellung

ID	Systemanforderung
A1	Lokale Sprachgenerierung ohne Cloud-Dienst
A2	Betrieb auf einer einzelnen Consumer-GPU
A3	Gemeinsame Einbindung mehrerer TTS-Modelle
A4	Automatische Verarbeitung längerer Texte
A5	Voice Cloning mit Preset- und eigenen Referenzstimmen
A6	Optionale Lautheitsangleichung und WAV-Export
A7	Nutzung über API und Weboberfläche
A8	Sequentielle Verarbeitung mehrerer Aufträge

4.2 Architektur und Komponenten

Das Gesamtsystem ist in mehrere voneinander getrennte Komponenten aufgeteilt. Die eigentliche Sprachgenerierung bildet dabei nur einen Teil der Verarbeitungspipeline. Eingabetext und Sprecherinformationen durchlaufen zunächst die Long-Form-Verarbeitung, bevor die erzeugten Sprachsegmente nachbearbeitet, zusammengefügt und schließlich als Audiodatei ausgegeben werden. Über dieser Verarbeitungsschicht befindet sich die Anwendungsschnittstelle, über die Generierungsaufträge entgegengenommen und die entsprechenden Ergebnisse bereitgestellt werden können.

Den Kern des Systems bildet eine gemeinsame Verarbeitungspipeline. Sie übernimmt die Aufteilung längerer Eingabetexte und steuert die segmentweise Sprachgenerierung über die ausgewählte TTS-Engine. Die konkrete Engine ist dabei von der weiteren Verarbeitung getrennt. Qwen3-TTS und VoxCPM2 werden über eine gemeinsame Schnittstelle angesprochen, sodass Segmentierung und nachfolgende Verarbeitungsschritte nicht direkt von einem bestimmten Modell abhängig sind.

Nach der Sprachgenerierung werden die einzelnen Segmente an eine separate Nachbearbeitungsstufe übergeben. Dort kann ihre Lautheit optional angeglichen werden. Anschließend werden die Segmente unter Berücksichtigung einer optional konfigurierbaren Pause zu einem durchgehenden Audiofile zusammengesetzt, das im Anschluss exportiert wird.

Sprecherinformationen werden ebenfalls als eigenständiger Bestandteil behandelt. Ein Sprecherprofil fasst die für das Voice Cloning benötigte Referenzaufnahme und den zugehörigen Referenztext zusammen. Dabei spielt es für die weitere Verarbeitung keine Rolle, ob ein bereits hinterlegtes Sprecherprofil oder ein vom Benutzer erstelltes Custom-Sprecherprofil verwendet wird.

Die Bedienung des Systems erfolgt über eine Anwendungsschicht mit Programmierschnittstelle und Weboberfläche. Eingereichte Generierungsaufträge werden zuerst in eine Warteschlange aufgenommen und dann nacheinander an die Verarbeitungspipeline übergeben. Dadurch bleibt die ressourcenintensive Sprachgenerierung auf nur einen Auftrag zur gleichen Zeit beschränkt, während mehrere Aufträge bereits im Voraus eingereicht werden können und in der Warteschlange platziert werden.

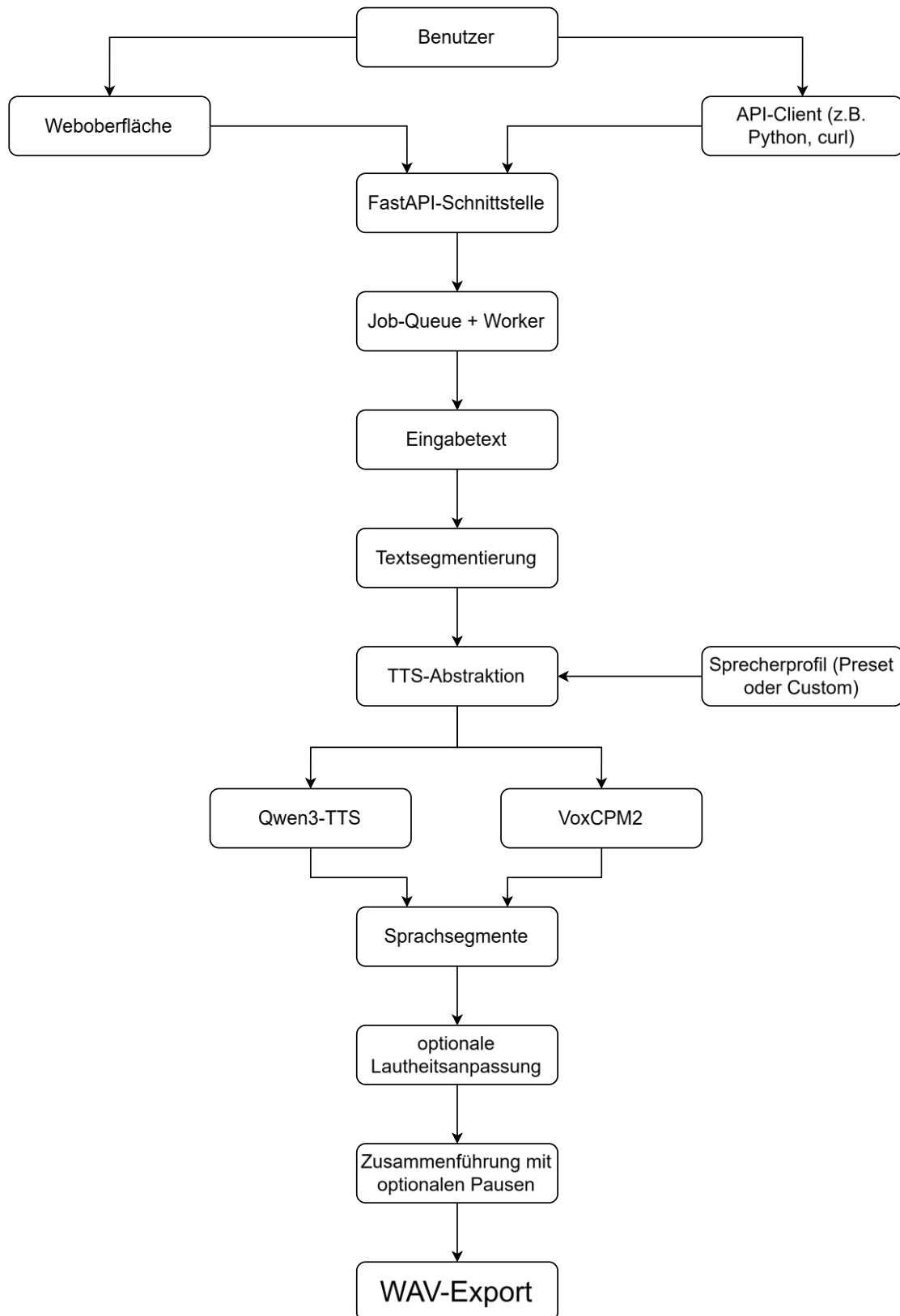


Abbildung 1: Schematischer Aufbau des entwickelten Long-Form-TTS-Systems, Quelle: Eigene Darstellung

4.3 Einheitliche TTS-Abstraktion

Da Qwen3-TTS und VoxCPM2 unterschiedliche Programmierschnittstellen und Modellparameter verwenden, sollen diese Unterschiede nicht in die übrige Verarbeitungspipeline übernommen werden. Stattdessen wird eine gemeinsame Schnittstelle verwendet, über die jedes eingebundene Modell angesprochen werden kann. Die Long-Form-Verarbeitung muss also nicht wissen, welches konkrete TTS-Modell für einen Auftrag verwendet wird. Als gemeinsame Eingaben werden der zu generierende Text und das ausgewählte Sprecherprofil, oder im Falle eines Custom-Sprechers die Referenzaufnahme des Sprechers sowie der zugehörige Referenztext, verwendet. Diese Informationen reichen für die in dieser Arbeit eingesetzten Voice-Cloning-Verfahren aus und können unabhängig vom konkreten Modell an die ausgewählte Engine übergeben werden. Modellspezifische Parameter und Aufrufe werden innerhalb der jeweiligen Engine gekapselt.

Auch das Ergebnis einer Generierung wird entsprechend vereinheitlicht. Unabhängig davon, mit welchem TTS-Modell die Sprache generiert wurde, liefert die gemeinsame Schnittstelle die Audiodaten zusammen mit der zugehörigen Abtastrate zurück. Nachfolgende Komponenten können die erzeugten Sprachsegmente dadurch einheitlich verarbeiten, ohne die Besonderheiten der jeweiligen Modellbibliothek berücksichtigen zu müssen.

Durch eine derartige Trennung kann die Verarbeitungspipeline unabhängig von den aktuell verwendeten Modellen aufgebaut werden. Weitere TTS-Engines könnten grundsätzlich ergänzt werden, sofern sie die gemeinsame Schnittstelle implementieren und die für die nachfolgenden Verarbeitungsschritte benötigten Audiodaten bereitstellen. Die konkrete Umsetzung der Schnittstelle für Qwen3-TTS und VoxCPM2 wird in Abschnitt 5.2 genauer beschrieben.

4.4 Long-Form-Verarbeitung

Wie in Abschnitt 2.2 beschrieben, können längere Eingaben bei der Sprachsynthese zu Problemen hinsichtlich Stimmcharakter, Prosodie und allgemeiner Konsistenz führen (Pan et al., 2026). Der Ansatz in dieser Arbeit ist deshalb, längere Texte nicht in einem einzelnen Generierungsschritt zu verarbeiten. Stattdessen wird der Eingabetext zunächst in kleinere Abschnitte zerlegt, die anschließend unabhängig voneinander synthetisiert werden.

Die Segmentierung orientiert sich dabei in erster Linie an Satzgrenzen. Dadurch sollen sprachlich zusammengehörige Einheiten möglichst erhalten und Trennungen innerhalb eines Satzes vermieden werden. Gleichzeitig muss berücksichtigt werden, dass Satzzeichen nicht zwangsläufig das Ende eines Satzes markieren müssen. Vor allem Punkte innerhalb häufiger Abkürzungen dürfen daher auf keinen Fall als Trennstelle interpretiert werden.

Einzelne Sätze werden anschließend zu größeren Segmenten zusammengefasst. Hierfür werden eine angestrebte Segmentlänge und eine maximale Segmentlänge verwendet. Die Zielgröße dient dazu, Segmente mit einer möglichst gleichmäßigen Länge zu erzeugen, während die Maximalgröße verhindert, dass einzelne Generierungsabschnitte zu lang werden. Überschreitet bereits ein einzelner Satz die zulässige maximale Länge, kann dieser als eventuelle Ausnahme an Wortgrenzen weiter unterteilt werden.

Die erzeugten Textsegmente werden in ihrer ursprünglichen Reihenfolge mit derselben ausgewählten TTS-Engine und demselben Sprecher (Preset oder Custom) synthetisiert. Da jedes Segment unabhängig generiert wird, können zwischen den einzelnen Audioabschnitten Unterschiede in der wahrgenommenen Lautheit entstehen. Die Verarbeitungspipeline sieht deshalb eine optionale Lautheitsangleichung vor, durch die solche Unterschiede zwischen den Segmenten reduziert werden können. Die Normalisierung bleibt bewusst optional, damit bei Bedarf auch die unveränderten Modellausgaben erhalten bleiben und untersucht werden können.

Nach der optionalen Nachbearbeitung werden die erzeugten Segmente wieder in ihrer ursprünglichen Reihenfolge zusammengesetzt. Zwischen zwei Segmenten wird eine konfigurierbare Pause eingefügt, da die Länge dieser Übergänge den Höreindruck des zusammengesetzten Sprachbeitrags maßgeblich beeinflussen kann. Die Ausgabe erfolgt anschließend als WAV-Datei. Segmentierung, Sprachgenerierung, Nachbearbeitung und Zusammenführung bilden damit getrennte Verarbeitungsschritte, die unabhängig voneinander angepasst werden können.

Die konkreten Regeln und Parameter der Segmentierung werden in Abschnitt 5.3 weiter beschrieben und die technische Umsetzung der Lautheitsangleichung, der Zusammenführung und des Audioexports wird anschließend in Abschnitt 5.4 erläutert.

5 Realisierung des Prototyps

Auf Grundlage des in Kapitel 4 beschriebenen Systementwurfs wurde der Prototyp als eigenständige Python-Anwendung umgesetzt. Dieses Kapitel beschreibt die praktische Realisierung der einzelnen Systembestandteile - im Mittelpunkt stehen dabei die konkrete Einbindung der TTS-Modelle, die Implementierung der Long-Form-Pipeline und der Audioverarbeitung sowie die Bereitstellung der Anwendung über API und Weboberfläche. Die Darstellung konzentriert sich auf diejenigen Klassen, Funktionen und Abläufe, die für das Verständnis der Implementierung wesentlich sind. Ausgewählte Quellcodeabschnitte zu den zentralen Verarbeitungsschritten werden ergänzend in Anhang 2 dargestellt. Außerdem wird dieser Arbeit der vollständige Quellcode des entwickelten Systems beigelegt und kann entsprechend eingesehen werden.

5.1 Entwicklungs- und Laufzeitumgebung

Der Prototyp wurde unter WSL mit Ubuntu 26.04 LTS entwickelt und getestet. Als Entwicklungsplattform kam hierbei eine NVIDIA GeForce RTX 5080 Grafikkarte mit 16 GB Grafikspeicher zum Einsatz. Die Anwendung ist auf eine CUDA-fähige NVIDIA-Grafikkarte ausgelegt, da beide eingebundenen TTS-Modelle während der Sprachgenerierung auf der GPU ausgeführt werden.

Die Anwendung selbst basiert auf Python 3.12.14 und wird innerhalb einer Conda-Umgebung mit der Bezeichnung `longform-tts` betrieben. Beide TTS-Engines werden innerhalb derselben Python-Umgebung ausgeführt. Eine Trennung in unterschiedliche Modellprozesse oder separate Laufzeitumgebungen war nicht erforderlich, da sich die Abhängigkeiten von Qwen3-TTS und VoxCPM2 in einer gemeinsamen Umgebung ohne Probleme betreiben ließen.

Tabelle 3: Wesentliche Komponenten der entwickelten Umgebung, Quelle: Eigene Darstellung

Komponente	Version / Spezifikation	Funktion
Betriebssystem	WSL / Ubuntu 26.04 LTS	Entwicklungs- und Referenzumgebung
GPU	NVIDIA GeForce RTX 5080, 16 GB VRAM	Ausführung der TTS-Modelle
NVIDIA-Treiber	610.43.02	CUDA-Zugriff unter WSL
Python	3.12.14	zentrale Anwendungs Laufzeit
PyTorch	2.14.0 mit CUDA 13.0	GPU-basierte Modellausführung
Qwen-TTS	0.1.1	Python-Paket für Qwen3-TTS
VoxCPM	2.0.3	Python-Paket für VoxCPM2
Transformers	4.57.3	Modell- und Transformer-Infrastruktur
Tokenizers	0.22.2	Tokenizer-Abhängigkeit
FastAPI	0.141.1	Programmierschnittstelle der Anwendung
pyloudnorm	0.2.0	Loudness-Messung und -Normalisierung

Der Quellcode ist als Python-Paket unter `src/longform_tts` organisiert. Die einzelnen Aufgabengebiete des Systems sind dabei auf getrennte Module verteilt: Die TTS-spezifischen Implementierungen befinden sich im Modul `engines`, während `pipeline` die Segmentierung, Generierung und Zusammenführung der Sprachsegmente enthält. Die Lautheitsverarbeitung ist separat unter `postprocessing` untergebracht. Die restlichen Module übernehmen die Verwaltung von Sprecherprofilen, den WAV-Export, die FastAPI-Anwendung sowie die statische Weboberfläche.

Abbildung 2: Vereinfachte Darstellung der Projektstruktur, Quelle: Eigene Darstellung

src/longform_tts	
├─ api/	FastAPI und Jobverwaltung
├─ engines/	TTS-Modelle und gemeinsame Schnittstelle
├─ export/	WAV-Export
├─ pipeline/	Segmentierung und Long-Form-Verarbeitung
├─ postprocessing/	Lautheitsverarbeitung
├─ speakers/	Sprecherprofile
└─ web/	Weboberfläche

Diese Aufteilung bildet die bereits im Systementwurf beschriebene Trennung der Verantwortlichkeiten auch innerhalb des Quellcodes ab. Die folgenden Abschnitte betrachten die einzelnen Bereiche und deren Zusammenspiel genauer. Die für eine erneute Installation benötigten Abhängigkeiten und Installationsschritte werden getrennt davon in Abschnitt 5.6 behandelt.

5.2 Integration der TTS-Engines und Sprecherprofile

Die beiden TTS-Modelle wurden als getrennte Engine-Klassen in das Modul `engines` integriert. Grundlage bildet die abstrakte Klasse `TTSEngine`, die einen einheitlichen Aufruf für die Sprachgenerierung definiert. Jede Engine erhält den zu synthetisierenden Text sowie Referenzaudio und Referenztext und gibt das Ergebnis in Form eines `TTSResult` zurück. Dieses enthält unabhängig vom verwendeten Modell die erzeugten Audiodaten und deren Abtastrate.

Abbildung 3: Gekürzter Ausschnitt der gemeinsamen Schnittstelle, Quelle: Eigener Quellcode

```
class TTSEngine(ABC):

    @abstractmethod
    def generate(
        self,
        text: str,
        reference_audio: str,
        reference_text: str,
    ) -> TTSResult:

        pass
```

Die konkreten Implementierungen `QwenEngine` und `VoxCPMEngine` übernehmen anschließend die Übersetzung dieses gemeinsamen Aufrufs auf die jeweilige Modellbibliothek. Bei `Qwen3-TTS` wird dazu die Voice-Cloning-Funktion des `qwen-tts`-Pakets verwendet. `VoxCPM2` besitzt eine andere Schnittstelle und erwartet die Referenzaufnahme u. a. als Prompt- und Stimmreferenz. Diese Unterschiede bleiben dabei innerhalb der jeweiligen Engine gekapselt und beide Implementierungen wandeln ihre Ausgabe anschließend in ein NumPy-Audiosignal um und geben dieses zusammen mit der jeweiligen Abtastrate als `TTSResult` zurück.

Die Auswahl der Engine erfolgt über die Funktion `create_engine()`. Nach außen werden dafür lediglich die Kennungen `qwen` und `voxcpm` verwendet. Erst bei der Erstellung der ausgewählten Engine wird die zugehörige Implementierung importiert und das jeweilige Modell geladen. Dadurch muss die Long-Form-Pipeline keine Kenntnis über die konkrete Modellklasse besitzen.

Auch die Sprecherinformationen werden innerhalb der Anwendung vereinheitlicht. Die Datenklasse `SpeakerProfile` enthält dafür lediglich einen Anzeigenamen, den Pfad zur Referenzaufnahme und den zugehörigen Referenztext. Bereits vorhandene Preset-Sprecher werden über die Klasse `SpeakerLibrary` aus dem Verzeichnis `data/speakers` geladen und jeder Sprecher wird dabei durch einen eigenen Unterordner mit einer `reference.wav` und einer `reference.txt` repräsentiert. Neue Preset-Sprecher können dadurch ohne Änderungen an der eigentlichen TTS-Pipeline ergänzt werden, indem einfach ein neuer Unterordner erstellt und mit den beiden entsprechenden Dateien gefüllt wird.

Benutzerdefinierte Sprecher werden nach demselben Prinzip verarbeitet. Die über die Anwendung bereitgestellte Referenzaufnahme und das zugehörige Transkript werden ebenfalls in ein `SpeakerProfile` überführt. Für die anschließende Generierung besteht damit kein Unterschied mehr zwischen einem hinterlegten Preset und einer neu bereitgestellten Referenzstimme. Der zentrale Long-Form-Workflow erhält in beiden Fällen lediglich ein Sprecherprofil und kann dessen Referenzdaten unabhängig von ihrer Herkunft an die ausgewählte TTS-Engine weitergeben.

5.3 Segmentierung und Long-Form-Pipeline

Die in Abschnitt 4.4 beschriebene Segmentierung ist im Modul `pipeline` umgesetzt. Ausgangspunkt ist die Funktion `segment_text()`, die zunächst den Eingabetext in einzelne Sätze zerlegt und diese anschließend zu größeren Textsegmenten zusammenfasst. Für die Satzerkennung werden die Satzzeichen Punkt, Ausrufezeichen und Fragezeichen berücksichtigt. Damit Punkte innerhalb häufiger Abkürzungen wie „z. B.“, „Dr.“ oder „bzw.“ nicht fälschlicherweise als Satzende interpretiert werden, werden diese vor der Erkennung vorübergehend geschützt und anschließend wiederhergestellt.

Die erkannten Sätze werden danach anhand zweier Grenzwerte gruppiert. In dem erstellten Prototyp werden eine angestrebte Segmentlänge von `target_chars = 300` Zeichen und eine maximale Länge von `max_chars = 400` Zeichen verwendet. Solange ein weiterer Satz innerhalb der Zielgröße bleibt, wird er dem aktuellen Segment hinzugefügt. Liegt die resultierende Länge zwischen Ziel- und Maximalwert, wird jeweils geprüft, ob das Segment mit oder ohne den zusätzlichen Satz näher an der Zielgröße von 300 Zeichen liegt. Die maximale Länge von 400 Zeichen wird dabei nicht überschritten. Ist bereits ein einzelner Satz länger als dieser Grenzwert, wird er ausnahmsweise an Wortgrenzen weiter unterteilt.

Die Parameter 300 und 400 Zeichen wurden als Baseline auf Grundlage vorangegangener Tests gewählt. Eine zunächst untersuchte kürzere Konfiguration mit 200 Zeichen Zielgröße und 275 Zeichen Maximalgröße erzeugte mehr Segmentgrenzen, ohne die Unterschiede in der Lautheit zwischen den einzelnen Segmenten erkennbar zu reduzieren. Die größere Konfiguration stellt daher einen Kompromiss zwischen begrenzter Länge der Einzelgenerierungen und einer möglichst geringen Anzahl künstlich erzeugter Übergänge dar.

Die eigentliche segmentweise Sprachgenerierung übernimmt die Klasse `LongFormService`. Sie erhält beim Erzeugen eines Auftrags eine Implementierung der zuvor beschriebenen `TTSengine` sowie die beiden Segmentierungsparameter. Der Eingabetext wird zunächst vollständig segmentiert und anschließend werden die entstandenen Abschnitte in ihrer ursprünglichen Reihenfolge an dieselbe TTS-Engine übergeben. Referenzaufnahme und Referenztext bleiben dabei für alle Segmente eines Auftrags identisch.

Das Ergebnis jedes Aufrufs wird als `GeneratedSegment` gespeichert. Neben den erzeugten Audio-daten enthält ein solches Segment seine Position innerhalb des Gesamttextes, den zugehörigen Text und die jeweilige Abtastrate. Die Liste dieser Segmente bildet die Schnittstelle zu den darauffolgenden Verarbeitungsschritten. Der übergeordnete Workflow `generate_longform_audio()` übergibt sie anschließend an die optionale Lautheitsanpassung und die Zusammenführung zum fertigen Audiosignal. Diese beiden Schritte werden im folgenden Abschnitt näher beschrieben.

5.4 Lautheitsanpassung, Zusammenführung und Export

Nach der segmentweisen Sprachgenerierung können die erzeugten Audioabschnitte optional hinsichtlich ihrer Lautheit angeglichen werden. Die Umsetzung befindet sich im Modul `postprocessing` und verwendet die Bibliothek `pyloudnorm` zur Messung der integrierten Lautheit in LUFS (Steinmetz & Reiss, 2021). Für jedes erzeugte Segment wird zunächst ein eigener Lautheitswert bestimmt. Als gemeinsamer Zielwert wird anschließend der Median der gemessenen Segmentlautheiten verwendet. Im Gegensatz zu einer fest vorgegebenen Ziellautheit orientiert sich die Anpassung damit an der von der

jeweiligen TTS-Engine erzeugten Ausgabe. Einzelne besonders laute oder leise Segmente beeinflussen den Zielwert durch die Verwendung des Medians zudem weniger stark als bei einem Mittelwert.

Vor der eigentlichen Normalisierung der Lautheit wird außerdem geprüft, ob die notwendige Verstärkung bei einem der Segmente zu einem Spitzenwert außerhalb des zulässigen Wertebereichs führen würde. Als Sicherheitsgrenze wird daher ein maximaler Peak von 0,999 verwendet. Ist das ursprünglich bestimmte Lautheitsziel für mindestens ein Segment zu hoch, wird der gemeinsame Zielwert entsprechend abgesenkt. Dadurch können weiterhin alle Segmente auf denselben Lautheitswert gebracht werden, ohne dass hierfür ein zusätzlicher Limiter oder nachträgliches Clipping erforderlich ist.

Anschließend werden die Segmente durch die Funktion `assemble_segments()` in ihrer ursprünglichen Reihenfolge zu einem gemeinsamen Audiosignal zusammengesetzt. Zwischen zwei aufeinanderfolgenden Segmenten wird dabei digitale Stille, also eine Pause, mit einer konfigurierbaren Dauer eingefügt. Als Standardwert verwendet der Prototyp eine zusätzliche Pause von 0,25 Sekunden. Nach dem letzten Segment wird keine weitere Pause angehängt.

Vor der Zusammenführung überprüft das System, ob alle Segmente dieselbe Abtastrate besitzen und als einkanaliges Audiosignal vorliegen. Ein automatisches Resampling ist nicht Bestandteil der Pipeline. Da innerhalb eines Generierungsauftrags jeweils dieselbe TTS-Engine verwendet wird, besitzen die regulär erzeugten Segmente eines Auftrags dieselbe Abtastrate. Qwen3-TTS liefert entsprechend eine zusammengesetzte Ausgabe mit 24 kHz und VoxCPM2 eine Ausgabe mit 48 kHz.

Das zusammengesetzte Audiosignal wird abschließend als `AssembledAudio` an die Exportkomponente übergeben. Die Funktion `save_wav()` speichert die Audiodaten zusammen mit ihrer Abtastrate über die Bibliothek `soundfile` als WAV-Datei. Der Export bleibt damit von der eigentlichen Sprachgenerierung und der vorherigen Nachbearbeitung getrennt.

5.5 API, Jobverarbeitung und Weboberfläche

Für die Nutzung außerhalb direkter Python-Aufrufe wurde die bestehende Verarbeitungspipeline um eine Anwendungsschicht auf Basis von FastAPI ergänzt. FastAPI ist ein Python-Webframework zur Entwicklung von Programmierschnittstellen und stellt auf Grundlage von OpenAPI u. a. automatisch eine interaktive API-Dokumentation bereit (FastAPI, o. D.). Die API des Prototyps enthält dabei keine eigene Implementierung der TTS- oder Long-Form-Logik. Stattdessen startet sie anhand der übergebenen Parameter die ausgewählte TTS-Engine und einen `LongFormService` und ruft anschließend den bereits beschriebenen Workflow `generate_longform_audio()` auf. Dadurch verwenden direkte Core-Aufrufe und Generierungen über die Webanwendung dieselbe Verarbeitungspipeline.

Tabelle 4: Wichtigste HTTP-Endpunkte der Anwendung, Quelle: Eigene Darstellung

Method	Endpoint	Function
GET	/	Weboberfläche ausliefern
GET	/health	Status der Anwendung prüfen
GET	/engines	verfügbare TTS-Engines auflisten
GET	/speakers	vorhandene Preset-Sprecher auflisten
GET	/jobs	alle Jobs der aktuellen Sitzung abrufen
GET	/jobs/{job_id}	Status eines einzelnen Jobs abrufen
POST	/jobs/generate	Generierung mit Preset-Sprecher einreichen
POST	/jobs/generate/custom	Generierung mit Custom-Sprecher einreichen
GET	/audio/{audio_id}	erzeugte WAV-Datei abrufen

Die Generierungsendpunkte führen die Sprachsynthese nicht innerhalb des eingehenden HTTP-Requests aus. Stattdessen wird für jeden Auftrag zunächst ein Job angelegt und mit dem Status `queued` in eine FIFO-Warteschlange eingereiht. Die API kann dadurch unmittelbar mit dem HTTP-Status 202 `Accepted` antworten, während die eigentliche und normalerweise länger dauernde GPU-Verarbeitung im Hintergrund erfolgt.

Für die Verwaltung dieser Aufträge wurde die Klasse `JobManager` implementiert. Sie hält die Informationen des Auftrags während der Laufzeit der Anwendung im Arbeitsspeicher vor und verwendet einen einzelnen Hintergrund-Worker. Dieser entnimmt die Aufträge nacheinander aus der Warteschlange und setzt ihren Zustand während der Verarbeitung von `queued` auf `running`. Nach erfolgreichem Abschluss erhält der Job den Status `done` und die Ergebnisinformationen werden gespeichert. Tritt während der Verarbeitung ein Fehler auf, wird ausschließlich der betroffene Auftrag als `failed` markiert - der Worker kann danach mit dem nächsten Eintrag fortfahren. Da genau ein Worker verwendet wird, wird zu jedem Zeitpunkt höchstens ein TTS-Auftrag auf der GPU ausgeführt.

Die Jobverwaltung ist bewusst speicherintern umgesetzt. Ein Neustart der Anwendung entfernt daher die vorhandenen Job-Metadaten und die Warteschlange. Bereits vollständig exportierte WAV-Dateien bleiben davon unberührt, da diese separat im Ausgabeverzeichnis gespeichert werden. Für den auf eine lokale Einzelplatzanwendung ausgelegten Prototyp wurde auf eine zusätzliche Datenbank verzichtet.

Für jeden über die API eingereichten Generierungsauftrag wird einmalig ein eigener zufälliger Seed erzeugt und an die ausgewählte TTS-Engine übergeben. Innerhalb eines Auftrags wird dieser Seed für

alle Segmente beibehalten. Dadurch können bei einer erneuten Generierung desselben Textes unterschiedliche Ausgaben entstehen, während die einzelnen Segmente eines Long-Form-Auftrags unter denselben Zufallsbedingungen erzeugt werden. Der verwendete Seed wird zusammen mit den Ergebnisinformationen des Jobs zurückgegeben.

Preset- und Custom-Sprecher verwenden zwei unterschiedliche Anfrageformate. Bei einem Preset-Sprecher genügt eine JSON-Anfrage mit Text, Engine, Sprecher-ID und den optionalen Verarbeitungseinstellungen. Für einen benutzerdefinierten Sprecher wird dagegen `multipart/form-data` verwendet, da zusätzlich eine WAV-Referenzaufnahme und wahlweise ein eingegebener oder als Textdatei hochgeladener Referenztext übertragen werden müssen. Die hochgeladene Audiodatei wird in einem temporären Verzeichnis zwischengespeichert, da sie auch nach Abschluss des ursprünglichen HTTP-Requests für die spätere Ausführung des Queue-Jobs verfügbar bleiben muss. Nach der Verarbeitung wird dieses Verzeichnis wieder entfernt.

Die Weboberfläche wurde ohne zusätzliches Frontend-Framework mit HTML, CSS und Vanilla-JavaScript umgesetzt und direkt durch FastAPI angesprochen. Beim Laden der Seite werden die verfügbaren TTS-Engines und Preset-Sprecher über die API abgefragt. Für eine Generierung können anschließend Modell und Sprecher ausgewählt, der zu synthetisierende Text eingegeben sowie die Lautheitsanpassung und die Dauer der Segmentpausen konfiguriert werden. Alternativ zur Auswahl des Preset-Sprechers kann in den Custom-Modus gewechselt und eine eigene Referenzaufnahme mit passendem Referenztext bereitgestellt werden.

Beispielhafte API-Aufrufe für Preset- und benutzerdefinierte Sprecher sowie die Abfrage eines abgeschlossenen Auftrags sind in Anhang 3 aufgeführt.

Long-Form TTS

Lokale Generierung längerer Sprachinhalte mit Voice Cloning.



Generierung

Zu sprechender Text: ?

Hier den zu synthetisierenden Text eingeben...

TTS-Engine: ?

VoxCPM2 (v2.0.3)

Sprecher

☒ Preset-Sprecher ☐ Eigene Stimme

Sprecher: Tom Kalberg

Audio-Einstellungen

☒ Loudness-Normalisierung ?

Pause zwischen Segmenten: ? 0,25 Sekunden

Audio generieren

Auftrag wurde eingereicht.

Engines und Sprecher wurden geladen.

[API-Dokumentation öffnen](#)

Aufträge ?

23.09.26, 19:15:26 | VoxCPM | Obwohl das Wetter am frühen Morgen noch recht kühl war, wurde es im L... Wartet

23.09.26, 19:15:23 | VoxCPM | Der nächste Termin findet am Donnerstag, dem 24. September, um 14:30... Wartet

23.09.26, 19:15:19 | Qwen | Wie klingt eine Stimme, wenn sie eine einfache Frage stellt? Und verä... Wird generiert

23.09.26, 19:15:15 | Qwen | Heute ist ein ruhiger Morgen, und draußen scheint bereits die Sonne.... Abgeschlossen



[WAV herunterladen](#)

Abbildung 4: Weboberfläche des entwickelten Prototyps, Quelle: Eigener Screenshot

Nach dem Einreichen eines Auftrags bleibt die Oberfläche weiterhin bedienbar, sodass weitere Jobs unmittelbar in die Warteschlange aufgenommen werden können. Die Jobliste fragt den aktuellen Zustand einmal pro Sekunde über die API ab und zeigt die Zustände `queued`, `running`, `done` und `failed` entsprechend an. Nach erfolgreicher Generierung wird das Ergebnis direkt über einen HTML-Audioplayer abspielbar und kann zusätzlich als WAV-Datei heruntergeladen werden.

5.6 Installation und reproduzierbarer Systemaufbau

Für die erneute Einrichtung des Prototyps werden die verwendete Python-Version und die wesentlichen Abhängigkeiten innerhalb des Projektverzeichnisses festgehalten. Conda ermöglicht die Verwaltung von voneinander getrennten Python-Umgebungen und deren Beschreibung über eine spezielle Environment-Datei, die auch zur erneuten Einrichtung einer Umgebung verwendet werden kann (Conda, o. D.). Die Datei `environment.yml` definiert hierfür die Conda-Umgebung `longform-tts` mit Python 3.12.14 sowie den grundlegenden Paketwerkzeugen. Die übrigen für den Prototyp benötigten Python-Pakete und deren genutzte Versionen sind separat in `requirements.txt` festgelegt.

Die Installation des gesamten Systems wird durch das Skript `install.sh` automatisiert. Dieses prüft zunächst grundlegende Voraussetzungen wie eine Linux- beziehungsweise WSL-Umgebung, eine vorhandene Conda-Installation und die Erreichbarkeit einer NVIDIA-GPU. Anschließend wird eine neue Conda-Umgebung angelegt, die getestete PyTorch-Version mit CUDA-Unterstützung installiert und danach die übrigen Projektabhängigkeiten eingerichtet. Abschließend wird das Projekt selbst in der frisch eingerichteten Umgebung installiert.

Nach Abschluss der Installation wird automatisch das Skript `check_environment.py` ausgeführt. Dieses kontrolliert u. a. die Python-Version, die Verfügbarkeit von CUDA, zentrale Paketversionen sowie die Imports von Qwen-TTS, VoxCPM und dem eigenen Python-Paket. Dadurch können Abweichungen von der getesteten Referenzumgebung bereits unmittelbar nach der Installation erkannt und zurückgemeldet werden.

Die Modelle selbst sind nicht Bestandteil des Projektverzeichnisses. Die in der Implementierung festgelegten Modellrevisionen werden beim ersten Einsatz automatisch heruntergeladen und anschließend lokal zwischengespeichert. Eine ausführliche Schritt-für-Schritt-Anleitung für die Einrichtung unter Windows mit WSL, den Start der Anwendung sowie typische Fehlerfälle ist in Anhang 4 und noch ausführlicher in der Datei `ANLEITUNG.md` enthalten. Der für diese Arbeit maßgebliche Softwarestand, die verwendeten Modellrevisionen sowie die für die Reproduzierbarkeit relevanten Dateien sind in Anhang 5 dokumentiert.

6 Evaluation

Nachdem in den vorherigen Kapiteln Konzeption und Realisierung des Prototyps beschrieben wurden, wird im Folgenden untersucht, inwieweit das entwickelte System die für diese Arbeit relevanten Anforderungen erfüllt. Im Mittelpunkt stehen dabei die technische Ausführbarkeit auf der verwendeten Consumer-Hardware, das Verhalten der beiden integrierten TTS-Engines sowie die Stabilität der entwickelten Long-Form-Verarbeitung.

Die Evaluation kombiniert hierfür technische Messgrößen mit einer qualitativen Betrachtung der erzeugten Audiodaten. Neben Laufzeit und Grafikspeicherbedarf werden insbesondere die Lautheitskonsistenz zwischen den erzeugten Segmenten sowie auffällige Veränderungen von Stimme, Aussprache und Sprechweise betrachtet. Ziel ist dabei nicht, einen umfassenden Benchmark der verwendeten TTS-Modelle durchzuführen, sondern die praktische Eignung des entwickelten Systems und der gewählten Verarbeitungsschritte zu beurteilen. Eine Nutzerstudie ist hierbei nicht Bestandteil der Evaluation. Qualitative Aussagen zur Audioqualität beruhen daher auf einer projektbezogenen Hörbeurteilung durch den Verfasser und werden entsprechend nicht als allgemeingültige Bewertung der wahrgenommenen Sprachqualität verstanden.

6.1 Teststrategie und Versuchsaufbau

Die Evaluation wird auf derselben Referenzumgebung durchgeführt, die bereits in Abschnitt 5.1 beschrieben wurde. Verwendet werden WSL mit Ubuntu 26.04 LTS, Python 3.12.14 sowie eine NVIDIA GeForce RTX 5080 mit 16 GB Grafikspeicher. Beide TTS-Engines werden innerhalb derselben Conda-Umgebung ausgeführt. Als Sprecherreferenz wird in allen Evaluationsläufen das Preset-Sprecherprofil `tom_kalberg` verwendet, das von dem Verfasser dieser Arbeit auf Basis seiner eigenen Stimme erstellt wurde. Im Unterschied zur regulären Nutzung über API und Weboberfläche, bei der jeder neue Auftrag einen eigenen zufälligen Seed erhält, wird für die direkten Evaluationsläufe ein fester Seed von 42 verwendet. Dadurch sollen die Zufallsbedingungen zwischen wiederholten Generierungen möglichst konstant gehalten werden.

Die Untersuchung wurde hierbei in drei Bereiche unterteilt. Zunächst werden die grundlegenden Funktionen des Gesamtsystems überprüft - dazu gehören die Generierung mit beiden TTS-Engines, die Verwendung von Preset- und benutzerdefinierten Sprecherprofilen, die automatische Segmentierung längerer Texte, die optionale Lautheitsanpassung, die Zusammenführung der Segmente sowie der WAV-Export. Zusätzlich wird geprüft, ob mehrere über die Anwendung eingereichte Aufträge in der vorgesehenen FIFO-Reihenfolge nacheinander verarbeitet werden.

Im zweiten Teil werden technische Messgrößen der beiden TTS-Engines erfasst. Dafür wird jeweils derselbe kurze Testtext mit derselben Sprecherreferenz verwendet. Gemessen werden die benötigte Generierungszeit, die Länge der erzeugten Audiodatei und der während der Generierung belegte Grafikspeicher. Aus Generierungszeit und Audiodauer wird zusätzlich der Real-Time Factor bestimmt, um die Verarbeitungsgeschwindigkeit unabhängig von der Länge der erzeugten Ausgabe vergleichen zu können. Die genaue Definition der Messgrößen wird in Abschnitt 6.2 näher erläutert.

Der dritte Teil konzentriert sich auf die Verarbeitung längerer Texte. Hierfür wird für beide Engines derselbe längere deutschsprachige Testtext verwendet. Die segmentierte Generierung erfolgt mit der in Abschnitt 5.3 festgelegten Baseline von 300 Zeichen Zielgröße und 400 Zeichen maximaler Segmentlänge, zusätzlich wird derselbe Text ohne die entwickelte Segmentierung direkt an die jeweilige TTS-Engine übergeben. Dadurch kann untersucht werden, welche Unterschiede zwischen einer längeren Einzelgenerierung und der segmentierten Long-Form-Verarbeitung auftreten.

Für die segmentierten Ausgaben werden die Lautheitswerte der einzelnen Abschnitte separat erfasst. Betrachtet werden insbesondere Mittelwert, Standardabweichung und Spannweite der integrierten Lautheit. Außerdem werden sowohl die unveränderte als auch die normalisierte Gesamtfassung des Audiofiles erzeugt. Die qualitative Hörbeurteilung konzentriert sich auf deutlich erkennbare Auffälligkeiten hinsichtlich Verständlichkeit, Aussprache, Stimmcharakter, Prosodie, Lautheitsverlauf und Übergängen zwischen den Segmenten.

Für den Vergleich der beiden TTS-Engines werden identische Eingabetexte, Sprecherreferenzen und Verarbeitungseinstellungen verwendet. Dadurch sollen Unterschiede, die auf unterschiedliche Testbedingungen zurückzuführen sind, eliminiert werden.

Die verwendeten Testtexte, die resultierende Segmentierung sowie die vollständigen Rohdaten der technischen Messungen und Lautheitsanalyse sind in Anhang 1 dokumentiert.

6.2 Technische Messgrößen

Zur technischen Bewertung der beiden TTS-Engines werden die Generierungszeit, die Dauer der erzeugten Audiodatei sowie der während der Generierung benötigte Grafikspeicher betrachtet. Die Messungen erfolgen mit demselben kurzen deutschsprachigen Testtext und derselben Sprecherreferenz. Die Modelle werden hierbei vor der eigentlichen Zeitmessung vollständig geladen, sodass der erstmalige Download der Modellgewichte und die Ladezeit des Modells nicht mit in die gemessene Generierungszeit einfließen.

Die Generierungszeit beschreibt die Zeitspanne zwischen dem Start der eigentlichen Sprachsynthese und dem Vorliegen des erzeugten Audiosignals. Da die resultierende Audiodauer zwischen den beiden Modellen trotz identischer Texteingabe unterschiedlich ausfallen kann, wird zusätzlich der Real-Time

Factor (RTF) berechnet (Pan et al., 2026). Dieser ergibt sich aus dem Verhältnis von Generierungszeit zu erzeugter Audiodauer:

$$\text{RTF} = \text{Generierungszeit} / \text{Audiodauer}$$

Ein RTF von 1 bedeutet, dass für die Erzeugung einer Sekunde Audio ungefähr eine Sekunde Rechenzeit benötigt wird. Werte unter 1 kennzeichnen eine Generierung, die schneller als Echtzeit erfolgt, während Werte über 1 einer langsameren Verarbeitung entsprechen. Der RTF ermöglicht dadurch einen direkten Vergleich der Verarbeitungsgeschwindigkeit als die alleinige Betrachtung der absoluten Generierungszeit.

Als zweite technische Größe wird der GPU-Speicherbedarf während der Sprachgenerierung erfasst. Hierfür werden die von PyTorch bereitgestellten CUDA-Speicherstatistiken verwendet. Betrachtet wird insbesondere der maximale während eines Generierungslaufs tatsächlich belegte Speicher. Ergänzend wird der vom CUDA-Speicherverwalter reservierte Speicher dokumentiert, der aufgrund des verwendeten Caching Allocators höher als der unmittelbar durch Tensoren belegte Speicher ausfallen kann (PyTorch, o. D.).

Die Messungen dienen nicht als allgemeiner Leistungsbenchmark der beiden Modelle, sondern zur Einordnung ihres Ressourcenbedarfs innerhalb der verwendeten Referenzumgebung. Insbesondere die gemessenen Laufzeiten und Speicherwerte sind von der eingesetzten GPU, den verwendeten Softwareversionen und der jeweiligen Modellkonfiguration abhängig.

6.3 Long-Form-Stabilität und Audioqualität

Für die Bewertung der Long-Form-Verarbeitung wird derselbe längere Testtext mit beiden TTS-Engines jeweils in zwei Varianten generiert. In der ersten Variante wird der vollständige Text ohne vorherige Segmentierung in einem einzelnen Aufruf an die jeweilige Engine übergeben. In der zweiten Variante erfolgt die Verarbeitung über die entwickelte Long-Form-Pipeline mit einer Zielgröße von 300 und einer maximalen Segmentlänge von 400 Zeichen. Die entstandenen Sprachsegmente werden anschließend mit einer Pause von 0,25 Sekunden zusammengesetzt.

Für die technische Betrachtung der segmentierten Generierung werden zunächst die unveränderten Ausgaben der TTS-Engines verwendet und für jedes Segment wird die integrierte Lautheit in LUFS bestimmt. Aus diesen Werten werden anschließend Mittelwert, Standardabweichung und Spannweite berechnet, wodurch sich beurteilen lässt, wie stark sich die Lautheit zwischen unabhängig erzeugten Segmenten verändert. Dann wird dieselbe Ausgabe zusätzlich mit aktivierter Lautheitsnormalisierung zusammengesetzt, sodass der Einfluss dieser Nachbearbeitung gesondert betrachtet werden kann.

Dazu erfolgt außerdem eine qualitative Hörbeurteilung der erzeugten Audiodateien. Dabei werden insbesondere Verständlichkeit und Aussprache sowie auffällige Veränderungen von Stimmcharakter, Sprechtempo, Prosodie und Lautheit betrachtet. Bei der segmentierten Variante wird zusätzlich darauf

geachtet, ob Übergänge zwischen den einzeln erzeugten Abschnitten hörbar auffallen oder der Eindruck einer zusammenhängenden Sprachaufnahme erhalten bleibt.

Der direkte Vergleich zwischen segmentierter und unsegmentierter Generierung soll dabei nicht als allgemeiner Qualitätsbenchmark der beiden TTS-Modelle verstanden werden. Untersucht wird vielmehr, ob die im Prototyp eingesetzte Segmentierung bei dem gewählten Long-Form-Test erkennbare Vorteile mit sich bringt. Da keine Nutzerstudie durchgeführt wird, werden qualitative Unterschiede ausschließlich als Beobachtungen innerhalb der durchgeführten Testläufe dokumentiert.

6.4 Ergebnisse

Vor der Durchführung der technischen Messungen wurden zunächst die grundlegenden Funktionen des Gesamtsystems überprüft. Beide TTS-Engines konnten erfolgreich über die gemeinsame Verarbeitungspipeline verwendet werden. Sowohl Preset- als auch benutzerdefinierte Sprecherprofile ließen sich verarbeiten, längere Texte wurden automatisch segmentiert, optional normalisiert und anschließend als WAV-Datei exportiert. Auch mehrere über die Anwendung eingereichte Generierungsaufträge wurden in der vorgesehenen FIFO-Reihenfolge sauber nacheinander verarbeitet.

Zunächst wurden die technischen Eigenschaften der beiden TTS-Engines anhand eines kurzen, identischen Testtextes untersucht. Jede Engine wurde hierfür dreimal mit derselben Sprecherreferenz und einem festen Seed von 42 ausgeführt. Tabelle 5 fasst die gemessenen Werte zusammen.

Tabelle 5: Technische Messwerte der TTS-Modelle, Quelle: Eigene Darstellung

Messgröße	Qwen3-TTS	VoxCPM2
Mittlere Generierungszeit	14,364 s	7,716 s
Mittlere Audiodauer	16,800 s	17,120 s
Mittlerer Real-Time-Factor	0,855	0,451
Max. belegter GPU-Speicher	5,305 GB	9,547 GB
Max. reservierter GPU-Speicher	5,814 GB	11,449 GB
Abtastrate	24 kHz	48 kHz

Beide Modelle erreichten unter den verwendeten Testbedingungen einen Real-Time Factor unter 1 und konnten die Sprachausgabe damit schneller als in Echtzeit erzeugen. VoxCPM2 war mit einem mittleren RTF von 0,451 deutlich schneller als Qwen3-TTS mit 0,855, benötigte gleichzeitig jedoch erheblich mehr Grafikspeicher. Der maximale belegte GPU-Speicher lag bei Qwen3-TTS bei rund 5,3 GB und

bei VoxCPM2 bei rund 9,5 GB. Beide Modelle konnten damit problemlos innerhalb der 16 GB Grafikspeicher der verwendeten RTX 5080 betrieben werden.

Für den Long-Form-Test wurde ein längerer deutschsprachiger Text mit beiden Engines über die entwickelte Segmentierung verarbeitet. Bei einer Zielgröße von 300 und einer maximalen Länge von 400 Zeichen entstanden jeweils sieben Segmente. Die gemessenen Lautheitswerte der unveränderten Modellausgaben sind in Tabelle 6 zusammengefasst.

Tabelle 6: Lautheitswerte der segmentierten Long-Form-Generierung, Quelle: Eigene Darstellung

Messgröße	Qwen3-TTS	VoxCPM2
Anzahl Segmente	7	7
Mittlere Lautheit	-29,87 LUFS	-23,94 LUFS
Standardabweichung	2,47 LU	0,55 LU
Spannweite	7,44 LU	1,66 LU
Zielwert der Normalisierung	-28,70 LUFS	-24,22 LUFS

Zwischen den beiden Engines zeigten sich deutliche Unterschiede bei der Lautheitsstabilität sowie der durchschnittlichen Lautheit. Die sieben Qwen3-TTS-Segmente lagen zwischen -27,75 und -35,19 LUFS, was eine Spannweite von 7,44 LU ergibt. Besonders das dritte Segment fiel mit -35,19 LUFS deutlich gegenüber den übrigen Abschnitten ab. Bei VoxCPM2 lagen die Werte dagegen wesentlich näher beieinander. Hier betrug die Spannweite lediglich 1,66 LU bei einer Standardabweichung von 0,55 LU.

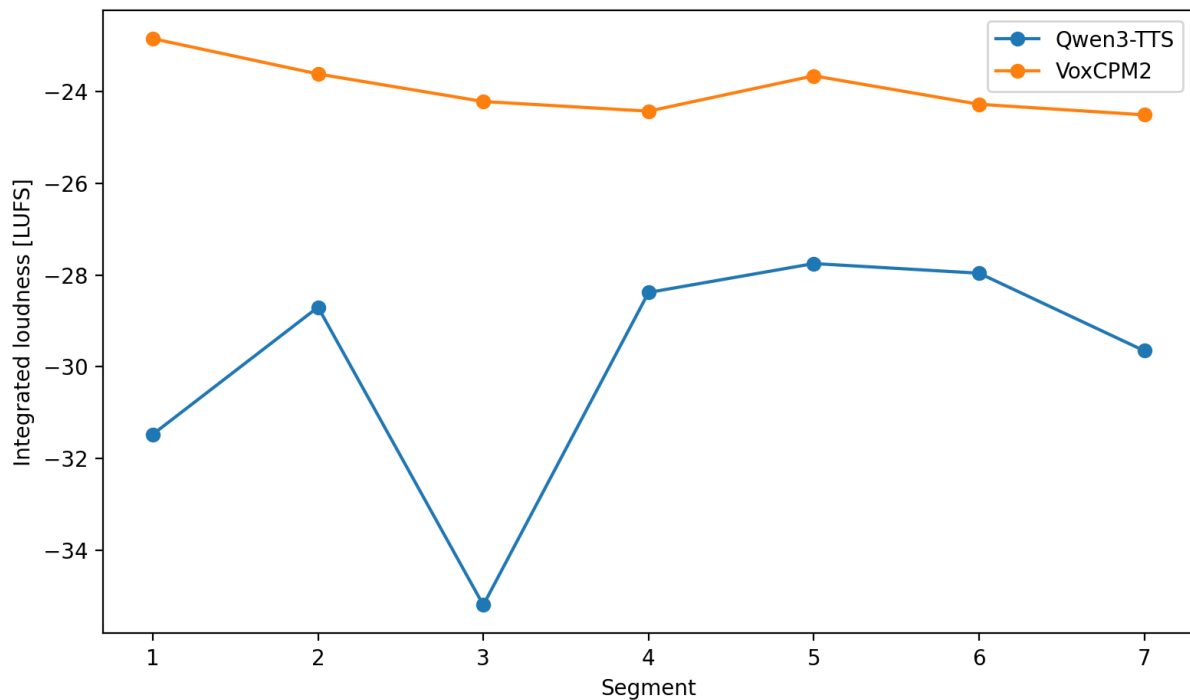


Abbildung 5: Integrierte Lautheit der sieben unveränderten Long-Form-Segmente, Quelle: Eigene Darstellung

Dieser Unterschied war auch bei der Hörbeurteilung deutlich wahrnehmbar. In der unveränderten Qwen3-TTS-Fassung traten zwischen mehreren Segmenten erkennbare Lautheitssprünge auf, zusätzlich veränderte sich der Stimmcharakter zwischen einzelnen Abschnitten. Einige Segmente wirkten wärmer und stärker betont, während andere einen blechernerer Klang aufwiesen. Die Übergänge zwischen den unabhängig erzeugten Abschnitten waren dadurch teilweise deutlich hörbar.

Durch die Lautheitsnormalisierung wurde die Qwen3-TTS-Ausgabe hörbar gleichmäßiger. Insbesondere Lautheitssprünge an den Segmentgrenzen wurden reduziert. Unterschiede innerhalb eines einzelnen Segments sowie Veränderungen von Stimmcharakter und Betonung blieben jedoch bestehen, da die verwendete Normalisierung lediglich einen konstanten Gain auf das jeweilige Segment anwendet.

Die segmentierte VoxCPM2-Ausgabe war bereits ohne Normalisierung wesentlich gleichmäßiger. Lautheit und Stimmcharakter blieben über die sieben Abschnitte vergleichsweise konstant, und die Segmentübergänge waren nur teilweise wahrnehmbar. Die Normalisierung führte daher im Vergleich zu Qwen3-TTS zu einer geringeren zusätzlichen Verbesserung, insbesondere hinsichtlich kleiner Lautheitsunterschiede an den Übergängen.

Zusätzlich wurde der Testtext ohne die entwickelte Segmentierung in einem einzelnen Generierungsaufwurf verarbeitet. Qwen3-TTS erzeugte dabei eine vollständige Audiodatei mit einer Dauer von 96,80 Sekunden und einer integrierten Lautheit von -36,78 LUFS. Der gesamte Text wurde ohne Auslassungen oder Wiederholungen gesprochen. Im Verlauf der Aufnahme wurde die Stimme jedoch zunehmend monotoner und leiser. Gleichzeitig nahmen undeutliche Aussprache und schwächere Betonung deutlich zu.

Im direkten Vergleich zur segmentierten Fassung wirkte die unsegmentierte Ausgabe dadurch deutlich weniger natürlich.

Bei VoxCPM2 führte die im Prototyp eingesetzte Kombination aus Prompt-Audio, Prompt-Text und Preset-Sprecher, der aus Referenzaudio und -text besteht, bei der unsegmentierten Generierung reproduzierbar zu einer unvollständigen Ausgabe. Ein größerer Abschnitt in der Mitte des Testtextes wurde übersprungen. Das Verhalten trat auch mit einem veränderten Seed erneut auf. Da diese Ausgabe für einen vollständigen Long-Form-Vergleich nicht geeignet war, wurde ergänzend ein unsegmentierter Lauf durchgeführt, bei dem ausschließlich die Referenzaufnahme ohne zugehörigen Text für das Voice Cloning verwendet wurde.

In dieser Variante wurde der Text vollständig erzeugt. Die Ausgabe hatte eine Dauer von 94,24 Sekunden und eine integrierte Lautheit von -22,05 LUFS. Die Lautheit blieb über den Verlauf vergleichsweise stabil und es traten keine Auslassungen oder Wiederholungen auf. Dennoch verschlechterte sich der Höreindruck mit zunehmender Dauer. Die Stimme wurde zunehmend blechern und verlor einen Teil ihres ursprünglichen warmen Stimmcharakters. Auch die Betonung wirkte gegen Ende zunehmend unnatürlich. Diese Verschlechterung war weniger ausgeprägt als bei Qwen3-TTS, blieb im direkten Vergleich zur segmentierten VoxCPM2-Ausgabe jedoch deutlich wahrnehmbar. Da für diesen ergänzenden Lauf die Sprecherkonditionierung gegenüber der regulären Konfiguration des Prototyps verändert wurde, ist die Ausgabe nicht als vollständig kontrollierter Vergleich zur segmentierten Variante zu verstehen. Sie dient vielmehr als ergänzende Beobachtung des Verhaltens von VoxCPM2 bei einer längeren Einzelgenerierung.

6.5 Einordnung der Ergebnisse

Die Evaluation zeigt, dass sich die in Abschnitt 4.1 formulierten Systemanforderungen mit dem entwickelten Prototyp weitgehend umsetzen lassen. Tabelle 7 fasst ihren im Rahmen dieser Arbeit erreichten Stand zusammen.

Tabelle 7: Erfüllung der wesentlichen Systemanforderungen, Quelle: Eigene Darstellung

ID/ Anforderung	Ergebnis	Einordnung
A1 Lokale Sprachgenerierung ohne Cloud-Dienst	Erfüllt	Beide TTS-Modelle werden vollständig lokal ausgeführt.
A2 Betrieb auf einer einzelnen Consumer-GPU	Erfüllt	Beide Engines konnten auf der verwendeten RTX 5080 mit 16 GB VRAM betrieben werden.
A3 Gemeinsame Einbindung mehrerer TTS-Modelle	Erfüllt	Qwen3-TTS und VoxCPM2 werden über dieselbe abstrahierte TTS-Schnittstelle angesprochen.
A4 Automatische Verarbeitung längerer Texte	Erfüllt	Längere Texte werden automatisch segmentiert, abschnittsweise generiert und anschließend wieder zusammengesetzt.
A5 Voice Cloning mit Preset- und eigenen Referenzstimmen	Erfüllt	Beide Varianten werden durch ein gemeinsames Sprecherprofil in die Verarbeitung eingebunden.
A6 Optionale Lautheitsangleichung und WAV-Export	Erfüllt	Die erzeugten Segmente können normalisiert und mit konfigurierbaren Pausen als WAV-Datei ausgegeben werden.
A7 Nutzung über API und Weboberfläche	Erfüllt	Die Generierung ist sowohl direkt über FastAPI als auch über die entwickelte Weboberfläche zugänglich.
A8 Sequentielle Verarbeitung mehrerer Aufträge	Erfüllt	Eine FIFO-Warteschlange mit einem einzelnen Worker verhindert parallele GPU-Generierungen.

Die technischen Messungen zeigen gleichzeitig, dass sich die beiden integrierten Modelle hinsichtlich ihrer praktischen Eigenschaften unterscheiden. Qwen3-TTS benötigt deutlich weniger Grafikspeicher, während VoxCPM2 im durchgeführten Kurztest eine höhere Generierungsgeschwindigkeit erreichte. Die gemeinsame Architektur ermöglicht es dadurch, unterschiedliche TTS-Modelle mit jeweils eigenen Stärken und Ressourcenanforderungen innerhalb derselben Long-Form-Pipeline zu verwenden.

Besonders relevant für die Zielsetzung dieser Arbeit ist das Ergebnis der Long-Form-Tests. Bei beiden Modellen traten bei längeren unsegmentierten Generierungen relevante Auffälligkeiten auf. Die entwickelte Segmentierung begrenzt diese Probleme, da die eigentliche Sprachsynthese jeweils nur auf deutlich kürzeren Textabschnitten erfolgt. Gleichzeitig entstehen dadurch jedoch neue Übergänge zwischen unabhängig erzeugten Segmenten. Insbesondere bei Qwen3-TTS waren Unterschiede in Lautheit und Stimmcharakter zwischen diesen Abschnitten deutlich hörbar. Die optionale Lautheitsangleichung konnte einen Teil dieser Übergänge reduzieren, löst jedoch erwartungsgemäß keine Veränderungen von Prosodie oder Stimmcharakter innerhalb der einzelnen Segmente.

Die Ergebnisse verdeutlichen damit einen grundlegenden Kompromiss der gewählten Long-Form-Verarbeitung. Die Segmentierung verhindert zwar nicht sämtliche qualitativen Unterschiede zwischen einzelnen Generierungen, reduziert jedoch die deutlich stärkeren Stabilitätsprobleme, die bei sehr langen Einzelgenerierungen beobachtet wurden. Für den untersuchten Anwendungsfall stellt die Kombination aus automatischer Segmentierung, einheitlicher Sprecherkonditionierung, optionaler Lautheitsangleichung und anschließender Zusammenführung daher einen praktikablen Ansatz zur lokalen Erzeugung längerer Audioinhalte dar.

Die Evaluation bezieht sich auf die verwendete Referenzhardware, die ausgewählten Modellversionen und die durchgeführten Testtexte. Insbesondere die qualitative Hörbeurteilung stellt keine allgemeingültige Bewertung der beiden TTS-Modelle dar. Aussagen zur Übertragbarkeit auf andere Hardware, Sprecher, Sprachen oder deutlich längere Inhalte können aus den durchgeführten Tests daher nur eingeschränkt abgeleitet werden. Diese Einschränkungen werden im folgenden Kapitel näher betrachtet.

7 Grenzen des entwickelten Ansatzes

Obwohl der entwickelte Prototyp die in Kapitel 4 formulierten Anforderungen weitgehend erfüllt, ergeben sich aus der gewählten Architektur, der verwendeten Hardware und den eingesetzten TTS-Modellen verschiedene Einschränkungen. Diese betreffen sowohl die technische Zugänglichkeit und Erweiterbarkeit des Systems als auch die erreichbare Qualität längerer Sprachgenerierungen.

Eine wesentliche Grenze ergibt sich zunächst aus der verwendeten Hardware. Entwicklung und Evaluation erfolgten auf einer NVIDIA GeForce RTX 5080 mit 16 GB Grafikspeicher. Für die beiden ausgewählten Modelle war diese Speicherkapazität ausreichend. Die Evaluation zeigte einen maximal belegten GPU-Speicher von rund 5,3 GB für Qwen3-TTS und rund 9,5 GB für VoxCPM2. Größere oder speicherintensivere Modelle können jedoch höhere Anforderungen stellen und wären auf der Referenzhardware gegebenenfalls nur mit zusätzlichen Optimierungen ausführbar. Gleichzeitig setzt der Prototyp in seiner getesteten Form eine CUDA-fähige NVIDIA-GPU voraus. CPU-basierte Systeme und alternative GPU-Plattformen werden nicht bzw. nur teilweise unterstützt.

Auch die parallele Verarbeitung ist bewusst eingeschränkt. Die Anwendung verwendet eine einzelne GPU und genau einen Worker, sodass mehrere Aufträge über die FIFO-Warteschlange nacheinander ausgeführt werden. Für eine lokale Einzelplatzanwendung ist dieses Verhalten ausreichend. Bei mehreren gleichzeitigen Benutzern oder einer größeren Anzahl von Aufträgen würde die Wartezeit jedoch entsprechend steigen. Zusätzlich werden Jobzustände und Warteschlange ausschließlich im Arbeitsspeicher gehalten und gehen beim Neustart der Anwendung verloren.

Die gemeinsame TTS-Schnittstelle erleichtert zwar die Integration weiterer Modelle, diese Erweiterbarkeit richtet sich jedoch in erster Linie an Entwickler. Eine neue Engine muss implementiert, an die gemeinsame Schnittstelle angepasst und hinsichtlich ihrer Abhängigkeiten in die vorhandene Laufzeitumgebung integriert werden. Ein automatisches Hinzufügen oder Aktualisieren von Modellen über die Weboberfläche ist nicht vorgesehen.

Eine weitere Grenze liegt in der Long-Form-Verarbeitung selbst. Jedes Segment wird unabhängig von den vorherigen Abschnitten synthetisiert. Das jeweilige TTS-Modell besitzt daher während der Generierung keine Kenntnis über die akustische Realisierung vorheriger Segmente. Dadurch können sich Stimmcharakter, Sprechtempo, Prosodie und Lautheit zwischen zwei Abschnitten verändern. Besonders bei Qwen3-TTS waren solche Unterschiede in der Evaluation hörbar. Die Segmentierung reduziert damit die bei längeren Einzelgenerierungen beobachteten Probleme, kann jedoch keine vollständig konsistente Langzeitmodellierung ersetzen. Probleme dieser Art werden auch in aktuellen Untersuchungen zu Long-Form-TTS beschrieben (Pan et al., 2026).

Die Lautheitsnormalisierung kann nur einen Teil dieser Unterschiede ausgleichen. Sie verändert den Pegel eines vollständigen Segments und reduziert dadurch Lautheitsunterschiede zwischen den Abschnitten. Veränderungen von Stimmfarbe, Aussprache oder Prosodie sowie Lautheitsschwankungen innerhalb eines Segments bleiben davon unbeeinflusst.

Auch die Segmentierungsparameter von 300 Zeichen Zielgröße und 400 Zeichen maximaler Länge stellen keine allgemeingültig optimalen Werte dar. Sie wurden im Rahmen dieser Arbeit als praktikable Baseline gewählt und getestet. Andere TTS-Modelle, Sprecher, Sprachen oder Textarten könnten jedoch von größeren oder kleineren Segmentgrößen profitieren, was entsprechend implementiert werden müsste. Außerdem orientiert sich die Segmentierung hauptsächlich an Satzgrenzen und Zeichenanzahl. Eine semantische Analyse von Absätzen, Themenwechseln oder anderen inhaltlichen Zusammenhängen findet nicht statt.

Der Prototyp besitzt außerdem keine automatische Qualitätskontrolle der erzeugten Sprache. Auslassungen, Wiederholungen, Veränderungen der Sprecherstimme oder andere akustische Artefakte werden von dem System nicht automatisch erkannt. Das bei einer langen VoxCPM2-Einzelgenerierung beobachtete Überspringen eines Textabschnitts würde vom System daher nicht selbstständig als Fehler erkannt.

Schließlich ist die Aussagekraft der Evaluation begrenzt. Untersucht wurden zwei TTS-Modelle auf einer einzelnen Referenzhardware mit einer begrenzten Anzahl deutschsprachiger Testtexte und einer Sprecherreferenz. Die qualitative Beurteilung erfolgte durch den Verfasser und nicht im Rahmen einer Nutzerstudie. Die Ergebnisse können daher nicht ohne Weiteres auf andere Hardware, Sprecher, Sprachen, Textarten oder Modellversionen übertragen werden.

8 Ethische und rechtliche Aspekte des Voice Cloning

Die Möglichkeit, die Stimme einer Person anhand einer kurzen Referenzaufnahme nachzubilden, gehört zu den technisch interessantesten Eigenschaften moderner TTS-Systeme. Gleichzeitig ist sie einer der Aspekte, bei denen die technische Machbarkeit besonders schnell mit ethischen und rechtlichen Fragen kollidiert. Während für legitime Anwendungen z. B. die eigene Stimme oder eine ausdrücklich freigegebene Sprecheraufnahme verwendet werden kann, lässt sich dieselbe Technik grundsätzlich auch auf öffentlich verfügbare Aufnahmen einer Person anwenden, ohne dass diese aktiv an dem Vorgang beteiligt ist.

Wie konfliktträchtig dies sein kann, zeigen mehrere öffentlich diskutierte Fälle. Große Aufmerksamkeit erhielt z. B. 2024 der Streit zwischen der Schauspielerin Scarlett Johansson und OpenAI. Johansson hatte nach eigenen Angaben zuvor eine Anfrage zur Nutzung ihrer Stimme abgelehnt und kritisierte anschließend, dass die von OpenAI verwendete Stimme „Sky“ ihrer eigenen Stimme auffällig ähnlich klinge. OpenAI bestritt, dass die Stimme als Imitation von Johansson angelegt worden sei, deaktivierte jedoch zeitgleich die Nutzung dieses Stimmprofils. Der Fall verdeutlicht, dass bereits eine starke Ähnlichkeit zu einer identifizierbaren Stimme Fragen nach Zustimmung und Kontrolle über die eigene stimmliche Identität aufwerfen kann (Chmielewski & Tong, 2024).

Noch direkter betrifft diese Problematik einen Rechtsstreit aus dem Jahr 2026. Der frühere Radiomoderator David Greene wirft Google vor, für die KI-generierten Audioinhalte des Dienstes NotebookLM eine Stimme verwendet zu haben, die seine charakteristische Sprechweise, Intonation und Stimmlage nachahme. Google bestreitet dies und gibt an, dass die betreffende Stimme auf einem bezahlten professionellen Sprecher basiere (Oremus, 2026). Greene machte dagegen u. a. Ansprüche wegen der unbefugten Verwendung seiner stimmlichen Identität geltend. Ein US-Bundesgericht entschied im Juli 2026 im Rahmen einer Zuständigkeitsfrage, dass sich diese Ansprüche auf die erzeugte stimmliche Ähnlichkeit und nicht lediglich auf die Vervielfältigung bestehender Tonaufnahmen beziehen. Über die eigentliche Berechtigung der Vorwürfe wurde damit noch nicht entschieden (Greene v. Google LLC et al., 2026).

Auch im deutschen Recht ist die Stimme nicht lediglich als austauschbares Audiosignal zu betrachten. Der Bundesgerichtshof hat bereits 1999 ausdrücklich festgestellt, dass der Stimme als kennzeichnendem

Merkmal einer Persönlichkeit ein beträchtlicher wirtschaftlicher Wert zukommen kann. Das Persönlichkeitsrecht schützt dabei auch die Entscheidung darüber, ob und unter welchen Voraussetzungen solche Persönlichkeitsmerkmale von Dritten wirtschaftlich verwendet werden dürfen (Bundesgerichtshof, 1999).

Mit dem europäischen AI Act werden zudem spezifische Transparenzanforderungen für KI-generierte Inhalte eingeführt. Artikel 50 sieht u. a. vor, dass Anbieter bestimmter Systeme synthetisch erzeugte Audio-, Bild-, Video- oder Textinhalte technisch als künstlich erzeugt oder manipuliert erkennbar machen. Betreiber eines Systems, das Audioinhalte erzeugt oder manipuliert, die als Deepfake einzustufen sind, müssen grundsätzlich offenlegen, dass die Inhalte künstlich erzeugt oder verändert wurden. Der AI Act gilt grundsätzlich seit dem 2. August 2026, wobei für einzelne Pflichten Übergangsregelungen bestehen (Europäische Union, 2024).

Die praktische Relevanz dieser Fragen zeigte sich auch unmittelbar bei der Entwicklung des in dieser Arbeit vorgestellten Prototyps. Für die Tests wurden zwei vordefinierte Sprecherprofile eingerichtet. Das erste Profil basiert auf einer eigens für diese Arbeit angefertigten Aufnahme des Verfassers Tom Kalberg und wurde von ihm selbst entsprechend mit eigener Einwilligung erstellt. Als zweites Profil wurde die Stimme des Tagesschau-Sprechers Jens Riewa verwendet. Hierfür wurde eine öffentlich ausgestrahlte Aufnahme aus der Tagesschau als Referenzmaterial verwendet. Eine Einwilligung von Jens Riewa zur Erstellung dieses Sprecherprofils liegt nicht vor.

Gerade dieser zweite Fall macht die niedrige technische Hürde von Voice Cloning anschaulich. Für die Erstellung des Profils war weder eine besondere Aufnahmeumgebung noch eine aktive Mitwirkung der betroffenen Person notwendig. Eine bereits vorhandene Sprachaufnahme mit bekanntem Inhalt reichte aus, um sie als Referenz für die im Prototyp eingesetzten Modelle zu verwenden. Die Stimme von Jens Riewa wurde im Rahmen dieser Arbeit ausschließlich für interne Entwicklungs- und Testzwecke verwendet. Die entsprechende Referenzaufnahme sowie daraus erzeugte Sprachdateien werden nicht öffentlich veröffentlicht, und auch der entwickelte Prototyp wird nicht als öffentlich zugänglicher Dienst bereitgestellt. Diese Beschränkung ersetzt jedoch keine Einwilligung und soll nicht als allgemeine Aussage über die rechtliche Zulässigkeit einer solchen Nutzung verstanden werden. Vielmehr dient das Beispiel innerhalb dieser Arbeit auch dazu, eine wichtige Eigenschaft heutiger Voice-Cloning-Systeme sichtbar zu machen: Die technische Nachbildung einer fremden Stimme kann wesentlich einfacher sein als die Klärung der damit verbundenen Rechte.

Aus ethischer Sicht folgt daraus, dass insbesondere Einwilligung und Transparenz eine zentrale Rolle spielen. Eine synthetische Stimme kann Aussagen erzeugen, die die betroffene Person niemals gesprochen hat. Je überzeugender das Voice Cloning funktioniert, desto schwieriger kann es für Zuhörer werden, zwischen einer authentischen Aufnahme und einer künstlich erzeugten Äußerung zu unterscheiden. Daraus ergeben sich u. a. Risiken durch Identitätsmissbrauch, Täuschung oder betrügerische Verwendung einer vertrauten Stimme (Azzuni & El Saddik, 2025). Gleichzeitig besitzt die Technologie legitime

Einsatzmöglichkeiten, z. B. für eigene virtuelle Sprecher, barrierefreie Anwendungen oder Medienproduktionen mit ausdrücklich freigegebenen Stimmen.

Für Systeme wie den in dieser Arbeit entwickelten Prototyp bedeutet dies, dass die technische Unterstützung von Voice Cloning allein noch keine Aussage darüber trifft, welche Stimmen verwendet werden sollten. Bei einer öffentlichen oder kommerziellen Weiterentwicklung wären deshalb zusätzliche Schutzmechanismen sinnvoll. Dazu könnten eine ausdrückliche Bestätigung der Nutzungsrechte an einer hochgeladenen Referenzaufnahme, Hinweise auf die notwendige Zustimmung des Sprechers sowie eine Kennzeichnung synthetisch erzeugter Audiodateien gehören.

Voice Cloning verdeutlicht damit besonders gut die allgemeine Situation vieler generativer KI-Verfahren: Mit sinkenden technischen Hürden steigt gleichzeitig die Verantwortung für ihre Verwendung. Dass eine Stimme technisch mit wenigen Schritten nachgebildet werden kann, bedeutet nicht automatisch, dass dies auch ohne Einschränkungen geschehen sollte.

9 Fazit und Ausblick

In diesem abschließenden Kapitel werden die zentralen Ergebnisse der Arbeit zusammengeführt und im Hinblick auf die formulierte Forschungsfrage eingeordnet. Anschließend werden mögliche Weiterentwicklungen des entstandenen Prototyps und Ansatzpunkte für zukünftige Untersuchungen betrachtet.

9.1 Fazit und Beantwortung der Forschungsfrage

Ziel dieser Arbeit war die Konzeption und Realisierung eines lokalen Systems zur automatisierten Generierung längerer Sprachinhalte mit optionalem Voice Cloning. Im Mittelpunkt stand dabei die Frage, wie unterschiedliche TTS-Modelle auf Consumer-Hardware in ein gemeinsames System integriert und für längere Eingabetexte nutzbar gemacht werden können.

Die Forschungsfrage kann dahingehend beantwortet werden, dass ein solches System durch die Trennung der einzelnen Verarbeitungsschritte in klar definierte Komponenten praktisch umgesetzt werden kann. Im entwickelten Prototyp werden Qwen3-TTS und VoxCPM2 über eine gemeinsame TTS-Schnittstelle angesprochen. Längere Eingabetexte werden unabhängig vom ausgewählten Modell automatisch segmentiert, abschnittsweise synthetisiert und anschließend wieder zu einer gemeinsamen Audiodatei zusammengesetzt. Referenzaufnahmen für das Voice Cloning werden über ein gemeinsames Sprecherprofil eingebunden, während Lautheitsanpassung, Pausengestaltung und WAV-Export als eigenständige Verarbeitungsschritte erfolgen.

Die praktische Umsetzung zeigte, dass dieser Ansatz auf einer einzelnen leistungsfähigen Consumer-GPU realisierbar ist. Beide untersuchten TTS-Modelle konnten auf der verwendeten RTX 5080 mit 16

GB Grafikspeicher lokal betrieben werden und erzeugten in diversen Testdurchläufen Audio schneller als in Echtzeit. Dabei zeigten sich unterschiedliche Ressourcenprofile. Qwen3-TTS benötigte deutlich weniger Grafikspeicher, während VoxCPM2 im durchgeführten Kurztest eine höhere Generierungsgeschwindigkeit erreichte.

Besonders relevant für die Forschungsfrage ist die Verarbeitung längerer Texte. Bei langen Einzelgenerierungen traten bei beiden untersuchten Modellen relevante Qualitäts- beziehungsweise Stabilitätsprobleme auf. Bei Qwen3-TTS waren insbesondere Veränderungen von Lautheit, Betonung und Stimmcharakter deutlich wahrnehmbar. Bei VoxCPM2 führte die reguläre Konditionierung im durchgeführten Test zu einer unvollständigen Ausgabe. Ein ergänzender Lauf mit veränderter Konditionierung erzeugte den vollständigen Text, zeigte im Verlauf jedoch ebenfalls qualitative Veränderungen.

Die segmentierte Verarbeitung konnte die beobachteten Langzeitprobleme begrenzen, erzeugte jedoch ihrerseits neue Übergänge zwischen unabhängig synthetisierten Abschnitten. Insbesondere bei Qwen3-TTS traten Unterschiede in Lautheit, Prosodie und Stimmcharakter zwischen einzelnen Segmenten auf. Die implementierte Lautheitsangleichung reduzierte hörbare Pegelunterschiede, konnte jedoch Veränderungen der eigentlichen Stimme oder der Betonung nicht korrigieren.

Der entwickelte Prototyp zeigt damit, dass die lokale automatisierte Generierung längerer Sprachinhalte mit bestehenden Open-Source-TTS-Modellen technisch realisierbar ist, wenn die eigentliche Sprachsynthese durch geeignete Verarbeitungsschritte ergänzt wird. Die Trennung von TTS-Engine, Segmentierung, Audioverarbeitung und Anwendungsschicht ermöglicht es dabei, unterschiedliche Modelle innerhalb desselben Long-Form-Workflows einzusetzen. Durch die FastAPI-Schnittstelle, die Weboberfläche und die sequentielle Jobverarbeitung kann das System außerdem als eigenständige lokale Anwendung genutzt oder über die API in andere Systeme eingebunden werden.

9.2 Ausblick

Für eine Weiterentwicklung des Systems bieten sich mehrere Ansatzpunkte an. Ein naheliegender Schritt ist die Integration weiterer TTS-Modelle. Die gemeinsame Engine-Schnittstelle wurde so gestaltet, dass zusätzliche Modelle ergänzt werden können, ohne die Segmentierung und die nachfolgenden Verarbeitungsschritte grundsätzlich verändern zu müssen. Dadurch ließe sich untersuchen, ob neuere oder stärker auf Long-Form-Anwendungen ausgerichtete Modelle eine höhere Konsistenz über längere Sprachabschnitte erreichen.

Eine weitere naheliegende Erweiterung wäre die gezielte Unterstützung zusätzlicher Sprachen. Da die eingesetzten TTS-Modelle mehrsprachig ausgelegt sind, könnte insbesondere englischsprachige Sprachsynthese mit vergleichsweise geringem technischem Aufwand in den bisherigen Workflow ein-

bezogen werden. Dabei wäre zu untersuchen, ob Segmentierung, Sprecherkonditionierung und Lautheitsverhalten vergleichbare Ergebnisse wie bei den in dieser Arbeit untersuchten deutschsprachigen Texten liefern.

Auch die Segmentierung bietet weiteres Optimierungspotenzial. Die derzeit verwendeten Grenzwerte von 300 Zeichen Zielgröße und 400 Zeichen maximaler Segmentlänge haben sich im Rahmen dieser Arbeit als praktikable Baseline erwiesen, stellen jedoch keine allgemeingültig optimalen Werte dar. Denkbar wären eine automatische Anpassung an die verwendete TTS-Engine oder eine stärker semantisch orientierte Segmentierung, bei der neben Satzgrenzen auch Absätze, Themenwechsel oder andere inhaltliche Zusammenhänge berücksichtigt werden.

Darüber hinaus könnte die bisher auf einen Sprecher pro Generierungsauftrag ausgelegte Verarbeitung um mehrere Sprecher erweitert werden. Einzelne Textabschnitte könnten dabei unterschiedlichen Sprecherprofilen zugeordnet und innerhalb eines gemeinsamen Audiobeitrags zusammengeführt werden. Dadurch ließen sich beispielsweise Dialoge, Interviews oder Podcast-ähnliche Formate mit mehreren synthetischen Stimmen erzeugen.

Eine weitere Erweiterungsmöglichkeit besteht in einer automatischen Qualitätskontrolle der generierten Sprache. Der aktuelle Prototyp erkennt nicht selbstständig, wenn ein Modell Textteile auslässt, die Sprecherstimme deutlich verändert oder andere auffällige Artefakte erzeugt. Verfahren zur automatischen Spracherkennung könnten genutzt werden, um den gesprochenen Inhalt mit dem Eingabetext abzugleichen. Ergänzend könnten Sprechermerkmale analysiert werden, um stärkere Veränderungen der Stimme zwischen einzelnen Segmenten automatisch zu erkennen.

Auch die Audioverarbeitung könnte erweitert werden. Die aktuelle Lautheitsanpassung verändert lediglich den Pegel vollständiger Segmente. Weiterführende Verfahren könnten Lautheitsverläufe innerhalb eines Segments berücksichtigen oder Übergänge zwischen benachbarten Abschnitten gezielter angleichen. Zusätzlich könnten optionale Intro- und Outro-Elemente oder Hintergrundmusik in die automatische Zusammenstellung der fertigen Audiodatei integriert werden.

Für eine Nutzung außerhalb einer lokalen Einzelplatzanwendung wären außerdem Erweiterungen der Jobverwaltung sinnvoll. Dazu gehören eine persistente Speicherung von Aufträgen, die Wiederaufnahme unterbrochener Generierungen sowie die Verteilung auf mehrere Worker oder GPUs. Auch eine einfachere Verwaltung zusätzlicher TTS-Modelle über die Benutzeroberfläche könnte die Anwendung für Nutzer ohne Programmierkenntnisse zugänglicher machen.

Zuletzt könnte eine umfangreichere Evaluation untersuchen, wie sich unterschiedliche Sprecher, Textarten, Sprachen und Segmentierungsparameter auf die resultierende Qualität auswirken. Eine Nutzerstudie mit mehreren Testpersonen könnte die in dieser Arbeit durchgeführte projektbezogene Hörbeurteilung ergänzen und untersuchen, ab welchem Ausmaß Unterschiede zwischen Segmenten tatsächlich als störend wahrgenommen werden.

Damit bietet der entwickelte Prototyp eine Grundlage für die weitere Untersuchung lokal ausführbarer Long-Form-TTS-Systeme und ihrer automatisierten Verarbeitung.

Literaturverzeichnis

- Azzuni, H., & El Saddik, A. (2025). *Voice cloning: Comprehensive survey* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2505.00579>. Abgerufen am 22. September 2026.
- Bundesgerichtshof. (1999). Urteil vom 1. Dezember 1999 – I ZR 49/97 (Marlene Dietrich). BGHZ 143, 214–232. <https://dejure.org/1999,4>. Abgerufen am 22. September 2026.
- Chmielewski, D., & Tong, A. (2024). Scarlett Johansson says OpenAI chatbot voice 'eerily similar' to hers. Reuters. <https://www.reuters.com/technology/scarlett-johansson-says-openai-chatbot-voice-eerily-similar-hers-2024-05-21/>. Abgerufen am 22. September 2026.
- Conda. (o. D.). *Managing environments*. Conda documentation. <https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html>. Abgerufen am 22. September 2026.
- Dai, Z., Chen, Y., Xu, J., Xie, L., Wang, Y., Yang, Z., Bai, B., Gao, Y., Zhou, W., Zhao, W., & Zhou, R. (2025). *Deep dubbing: End-to-end auto-audiobook system with text-to-timbre and context-aware instruct-TTS* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2509.15845>. Abgerufen am 22. September 2026.
- Europäische Union. (2024). Verordnung (EU) 2024/1689 des Europäischen Parlaments und des Rates vom 13. Juni 2024 zur Festlegung harmonisierter Vorschriften für künstliche Intelligenz (Verordnung über künstliche Intelligenz). Amtsblatt der Europäischen Union, L, 2024/1689. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj/deu>. Abgerufen am 22. September 2026.
- European Broadcasting Union. (2023). *Loudness normalisation and permitted maximum level of audio signals (EBU R 128, Version 5.0)*. <https://tech.ebu.ch/publications/r128>. Abgerufen am 22. September 2026.
- FastAPI. (o. D.). *FastAPI documentation*. <https://fastapi.tiangolo.com/>. Abgerufen am 22. September 2026.
- FireRedTeam. (2025). *FireRedTTS2* [Computer software]. GitHub. <https://github.com/FireRedTeam/FireRedTTS2>. Abgerufen am 22. September 2026.
- Greene v. Google LLC et al. (2026). Order granting motion to remand, Case No. 3:26-cv-01773-CRB. United States District Court for the Northern District of California. <https://docs.justia.com/cases/federal/district-courts/california/candce/3%3A2026cv01773/465155/38>. Abgerufen am 22. September 2026.
- Hu, H., Zhu, X., He, T., Guo, D., Zhang, B., Wang, X., Guo, Z., Jiang, Z., Hao, H., Guo, Z., Zhang, X., Zhang, P., Yang, B., Xu, J., Zhou, J., & Lin, J. (2026). *Qwen3-TTS technical report* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2601.15621>. Abgerufen am 22. September 2026.

- International Telecommunication Union. (2023). *Algorithms to measure audio programme loudness and true-peak audio level (Recommendation ITU-R BS.1770-5)*. <https://www.itu.int/rec/R-REC-BS.1770-5-202311-I>. Abgerufen am 22. September 2026.
- Kaul, A. (2026). *Automatisierung der KI-basierten Generierung von audiovisuellen Medieninhalten* [Bachelorarbeit, Hochschule für Angewandte Wissenschaften Hamburg]. REPOSIT. <https://hdl.handle.net/20.500.12738/19837>. Abgerufen am 22. September 2026.
- Liu, M., Yin, J., Zhang, X., Hao, S., Hu, Y., Lin, B., Feng, Y., Zhou, H., & Ye, J. (2025). *Audiobook-CC: Controllable long-context speech generation for multicast audiobook* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2509.17516>. Abgerufen am 22. September 2026.
- mooja77. (2026). Text2Audio [Computer software]. GitHub. <https://github.com/mooja77/Text2Audio>. Abgerufen am 22. September 2026.
- OpenMOSS Team. (2026). *MOSS-TTS-Nano* [Computer software]. GitHub. <https://github.com/OpenMOSS/MOSS-TTS-Nano>. Abgerufen am 22. September 2026.
- Oremus, W. (2026). He spent decades perfecting his voice. Now he says Google stole it. The Washington Post. <https://www.washingtonpost.com/technology/2026/02/15/david-greene-google-ai-podcast/>. Abgerufen am 22. September 2026.
- Pan, C., Yang, R., Wang, H., Zhou, Z., He, X., Guo, W., Jiang, Z., Li, R., Zhang, Y., Wen, C., Lei, K., Yin, X., Lu, J., Zhu, Z., & Zhao, Z. (2026). Comprehensive benchmarking of long-form speech generation in diverse scenarios. In M. Liakata, V. P. Moreira, J. Zhang, & D. Jurgens (Eds.), *Findings of the Association for Computational Linguistics: ACL 2026* (pp. 2365-2400). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2026.findings-acl.112>. Abgerufen am 22. September 2026.
- PyTorch. (o. D.). *CUDA semantics*. PyTorch documentation. <https://docs.pytorch.org/docs/main/notes/cuda.html>. Abgerufen am 22. September 2026.
- QwenAudio. (o. D.). *CosyVoice* [Computer software]. GitHub. <https://github.com/QwenAudio/CosyVoice>. Abgerufen am 22. September 2026.
- Raza, M. S. (2026). *Voice Studio by MSR* [Computer software]. GitHub. <https://github.com/msr-builds/voice-studio>. Abgerufen am 22. September 2026.
- Steinmetz, C. J., & Reiss, J. (2021). *pyloudnorm: A simple yet flexible loudness meter in Python*. 150th Audio Engineering Society Convention, Paper 10483. Audio Engineering Society. <https://secure.aes.org/forum/pubs/conventions/?elib=21076>. Abgerufen am 22. September 2026.
- Stifli, E. (2026). *LocalText2Voice* [Computer software]. GitHub. <https://github.com/estebanstifli/LocalText2Voice>. Abgerufen am 22. September 2026.

- Xie, T., Rong, Y., Zhang, P., Wang, W., & Liu, L. (2025). Towards controllable speech synthesis in the era of large language models: A systematic survey. In C. Christodoulopoulos, T. Chakraborty, C. Rose, & V. Peng (Eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing* (pp. 764-791). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2025.emnlp-main.40> . Abgerufen am 22. September 2026.
- Zhou, Y., Zeng, G., Liu, X., Li, X., Yu, R., Gui, J., Wu, J., Wang, Z., Shen, X., Ye, R., Zhang, Z., Zhou, J., Bai, B., Sun, W., Deng, M., Shi, Q., Wu, Z., & Liu, Z. (2026). *VoxCPM2 technical report* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2606.06928> . Abgerufen am 22. September 2026.
- Zhu, H., Ye, L., Kang, W., Yao, Z., Guo, L., Kuang, F., Han, Z., Zhuang, W., Lin, L., & Povey, D. (2026). *OmniVoice: Towards omnilingual zero-shot text-to-speech with diffusion language models* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2604.00688> . Abgerufen am 22. September 2026.

Anhang 1: Evaluationsmaterial

Dieser Anhang enthält ergänzende Materialien zu der in Kapitel 6 beschriebenen Evaluation. Aufgeführt werden die verwendeten Testtexte, die daraus erzeugte Segmentierung sowie die Rohdaten der technischen Messungen und die Lautheitswerte der einzelnen Long-Form-Segmente.

1.1 Testtext des Kurzbenchmarks

Für den in Abschnitt 6.2 beschriebenen Kurzbenchmark wurde für beide TTS-Engines derselbe deutschsprachige Testtext verwendet. Die Messungen erfolgten mit derselben Sprecherreferenz und einem festen Seed von 42.

Ein ruhiger Morgen beginnt oft unspektakulär. Die Straßen füllen sich langsam, und irgendwo wird bereits eine Tasse Kaffee auf den Schreibtisch gestellt. Technische Systeme können heute aus Text natürlich klingende Sprache erzeugen. Für längere Beiträge muss diese Sprache jedoch über mehrere Abschnitte hinweg möglichst konsistent bleiben.

1.2 Testtext der Long-Form-Evaluation

Für die in Abschnitt 6.3 beschriebene Long-Form-Evaluation wurde für beide TTS-Engines derselbe längere deutschsprachige Testtext verwendet. Dieser wurde sowohl unsegmentiert als auch mit der in Abschnitt 5.3 beschriebenen Segmentierung mit 300 Zeichen Zielgröße und 400 Zeichen maximaler Segmentlänge verarbeitet.

Ein ruhiger Morgen beginnt oft unspektakulär. Die Straßen füllen sich langsam, in den Häusern gehen die ersten Lichter an, und irgendwo wird bereits eine Tasse Kaffee auf den Schreibtisch gestellt. Viele Menschen beginnen ihren Tag mit denselben kleinen Routinen, ohne ihnen besondere Aufmerksamkeit zu schenken. Gerade diese alltäglichen Abläufe können jedoch interessant sein, wenn man sie einmal genauer betrachtet. Technische Entwicklungen verändern solche Gewohnheiten häufig schleichend. Vor einigen Jahren war es beispielsweise noch ungewöhnlich, mit einem Computer ganz selbstverständlich zu sprechen. Heute können digitale Systeme Fragen beantworten, Texte vorlesen und sogar Stimmen anhand kurzer Aufnahmen nachbilden. Dabei

entsteht schnell der Eindruck, dass die Technik bereits vollkommen selbstverständlich geworden ist. Hinter einer natürlich klingenden Stimme verbergen sich jedoch zahlreiche einzelne Verarbeitungsschritte. Besonders deutlich wird das bei längeren gesprochenen Beiträgen. Eine einzelne kurze Antwort kann überzeugend klingen, obwohl sich kleine Schwächen kaum bemerkbar machen. Bei mehreren Minuten Sprache sieht das anders aus. Das Sprechtempo sollte nicht plötzlich steigen oder fallen, die Lautstärke sollte möglichst konstant bleiben, und auch der Charakter einer Stimme darf sich im Verlauf der Aufnahme nicht ungewollt verändern. Genauso wichtig sind die Pausen zwischen den Sätzen. Sind sie zu kurz, wirkt die Sprache gehetzt. Sind sie zu lang, entsteht dagegen schnell ein unnatürlicher Rhythmus. Ein gutes Sprachsystem muss deshalb nicht nur einzelne Wörter korrekt aussprechen, sondern einen zusammenhängenden Text als glaubwürdigen gesprochenen Beitrag wiedergeben. Dieser Test untersucht deshalb eine segmentierte Verarbeitung des längeren Textes. Die einzelnen Abschnitte werden getrennt erzeugt und anschließend wieder zu einer zusammenhängenden Aufnahme zusammengesetzt. Dadurch lässt sich untersuchen, ob die Segmentierung zu einem stabileren und natürlicheren Ergebnis beitragen kann.

1.3 Segmentierung des Long-Form-Testtexts

Der in Abschnitt 1.2 aufgeführte Long-Form-Testtext wurde mit der im Prototyp verwendeten Baseline von `target_chars = 300` und `max_chars = 400` automatisch segmentiert. Dabei entstanden die folgenden sieben Textsegmente. Die angegebene Zeichenzahl umfasst jeweils den vollständigen Segmenttext einschließlich Leerzeichen und Satzzeichen.

Segment 1, 312 Zeichen

Ein ruhiger Morgen beginnt oft unspektakulär. Die Straßen füllen sich langsam, in den Häusern gehen die ersten Lichter an, und irgendwo wird bereits eine Tasse Kaffee auf den Schreibtisch gestellt. Viele Menschen beginnen ihren Tag mit denselben kleinen Routinen, ohne ihnen besondere Aufmerksamkeit zu schenken.

Segment 2, 296 Zeichen

Gerade diese alltäglichen Abläufe können jedoch interessant sein, wenn man sie einmal genauer betrachtet. Technische Entwicklungen verändern solche Gewohnheiten häufig schleichend. Vor einigen Jahren war es beispielsweise noch ungewöhnlich, mit einem Computer ganz selbstverständlich zu sprechen.

Segment 3, 330 Zeichen

Heute können digitale Systeme Fragen beantworten, Texte vorlesen und sogar Stimmen anhand kurzer Aufnahmen nachbilden. Dabei entsteht schnell der Eindruck, dass die Technik bereits vollkommen selbstverständlich geworden ist. Hinter einer natürlich klingenden Stimme verbergen sich jedoch zahlreiche einzelne Verarbeitungsschritte.

Segment 4, 221 Zeichen

Besonders deutlich wird das bei längeren gesprochenen Beiträgen. Eine einzelne kurze Antwort kann überzeugend klingen, obwohl sich kleine Schwächen kaum bemerkbar machen. Bei mehreren Minuten Sprache sieht das anders aus.

Segment 5, 304 Zeichen

Das Sprechtempo sollte nicht plötzlich steigen oder fallen, die Lautstärke sollte möglichst konstant bleiben, und auch der Charakter einer Stimme darf sich im Verlauf der Aufnahme nicht ungewollt verändern. Genauso wichtig sind die Pausen zwischen den Sätzen. Sind sie zu kurz, wirkt die Sprache gehetzt.

Segment 6, 325 Zeichen

Sind sie zu lang, entsteht dagegen schnell ein unnatürlicher Rhythmus. Ein gutes Sprachsystem muss deshalb nicht nur einzelne Wörter korrekt aussprechen, sondern einen zusammenhängenden Text als glaubwürdigen gesprochenen Beitrag wiedergeben. Dieser Test untersucht deshalb eine segmentierte Verarbeitung des längeren Textes.

Die einzelnen Abschnitte werden getrennt erzeugt und anschließend wieder zu einer zusammenhängenden Aufnahme zusammengesetzt. Dadurch lässt sich untersuchen, ob die Segmentierung zu einem stabileren und natürlicheren Ergebnis beitragen kann.

1.4 Rohdaten der technischen Messungen

Für den technischen Kurzbenchmark wurden beide TTS-Engines jeweils dreimal mit demselben Text, derselben Sprecherreferenz und einem festen Seed von 42 ausgeführt. Die folgenden Tabellen enthalten die Einzelwerte der drei Messdurchläufe. Die Modelle waren vor Beginn der Zeitmessung bereits vollständig geladen.

Qwen3-TTS

Durchlauf	Gen.-zeit [s]	Audiodauer [s]	RTF	GPU belegt [GB]	GPU reserviert [GB]
1	15,206	16,800	0,905	5,243	5,420
2	14,011	16,800	0,834	5,305	5,814
3	13,875	16,800	0,826	5,305	5,814

Die mittlere Generierungszeit betrug 14,364 s bei einer mittleren Audiodauer von 16,800 s und einem mittleren RTF von 0,855. Der maximal belegte GPU-Speicher lag bei 5,305 GB, der maximal reservierte Speicher bei 5,814 GB. Die Ausgabe erfolgte mit 24 kHz.

VoxCPM2

Durchlauf	Gen.-zeit [s]	Audiodauer [s]	RTF	GPU belegt [GB]	GPU reserviert [GB]
1	7,856	17,120	0,459	9,546	11,449
2	7,589	17,120	0,443	9,546	11,449
3	7,701	17,120	0,450	9,546	11,449

Die mittlere Generierungszeit betrug 7,716 s bei einer mittleren Audiodauer von 17,120 s und einem mittleren RTF von 0,451. Der maximal belegte GPU-Speicher lag bei 9,547 GB, der maximal reservierte Speicher bei 11,449 GB. Die Ausgabe erfolgte mit 48 kHz.

1.5 Lautheitswerte der Long-Form-Segmente

Für die sieben Segmente der Long-Form-Evaluation wurde jeweils die integrierte Lautheit der unveränderten Modellausgabe bestimmt. Die folgenden Werte bilden die Grundlage für die in Tabelle 6 des Hauptteils zusammengefassten Lautheitskennzahlen.

Segment	Qwen3-TTS [LUFS]	VoxCPM2 [LUFS]
1	-31,47	-22,85
2	-28,70	-23,62
3	-35,19	-24,22
4	-28,38	-24,43
5	-27,75	-23,66
6	-27,96	-24,28
7	-29,65	-24,51
Mittelwert	-29,87	-23,94
Standardabweichung	2,47 LU	0,55 LU
Spannweite	7,44 LU	1,66 LU
Normalisierungsziel	-28,70 LUFS	-24,22 LUFS

Anhang 2: Ausgewählte Implementierungsdetails

Dieser Anhang enthält ausgewählte Quellcodeausschnitte zentraler Komponenten des entwickelten Prototyps. Die Ausschnitte ergänzen die Implementierungsbeschreibung in Kapitel 5 und konzentrieren sich auf die für die Long-Form-Verarbeitung wesentlichen Abläufe. Der vollständige Quellcode liegt dieser Arbeit bei.

2.1 Textsegmentierung

Die Segmentierung kombiniert eine Zielgröße mit einer maximal zulässigen Segmentlänge. Der folgende Ausschnitt zeigt die zentrale Entscheidungslogik bei der Gruppierung aufeinanderfolgender Sätze. Solange das Hinzufügen eines weiteren Satzes innerhalb der Zielgröße bleibt, wird dieser übernommen. Wenn die maximale Länge überschritten wird, wird ein neues Segment begonnen. Liegt die

resultierende Länge zwischen Zielgröße und Maximalwert, wird die Variante gewählt, deren Länge näher an der Zielgröße liegt.

```
current_text = " ".join(current_sentences)
candidate_text = f"{current_text} {sentence}"
candidate_length = len(candidate_text)

if candidate_length <= target_chars:
    current_sentences.append(sentence)
    continue

if candidate_length > max_chars:
    chunks.append(current_text)
    current_sentences = [sentence]
    continue

# Zwischen target_chars und max_chars gewinnt die Variante näher am Zielwert.
current_distance = abs(target_chars - len(current_text))
candidate_distance = abs(candidate_length - target_chars)

if candidate_distance <= current_distance:
    current_sentences.append(sentence)
else:
    chunks.append(current_text)
    current_sentences = [sentence]
```

Codeblock 1: Entscheidungslogik der satzbasierten Segmentierung, Quelle: Eigener Quellcode

Die vorgelagerte Satzerkennung, der Schutz häufiger Abkürzungen sowie die Behandlung einzelner überlanger Sätze sind in derselben Quelldatei implementiert und werden hier nicht zusätzlich dargestellt.

2.2 Clipping-sichere Lautheitsnormalisierung

Die Lautheitsnormalisierung verwendet zunächst den Median der gemessenen Segmentlautheiten als gemeinsamen Zielwert. Anschließend wird für jedes Segment geprüft, welche maximale Verstärkung unter Einhaltung eines Spitzenpegels von 0,999 möglich ist. Falls der ursprüngliche Zielwert bei mindestens einem Segment zu Clipping führen würde, wird er auf den niedrigsten noch sicheren Wert abgesenkt.

```

# Der Median reagiert weniger empfindlich auf einzelne Ausreißer als der
Mittelwert.
target_lufs = float(np.median(loudness_values))

# Das gemeinsame Ziel wird bei Bedarf so weit abgesenkt,
# dass keines der Segmente durch die Verstärkung clippen kann.
max_peak = 0.999
safe_target_values = []

for segment, current_lufs in zip(
    segments,
    loudness_values,
    strict=True,
):
    peak = float(np.max(np.abs(segment.audio)))

    if peak <= 0:
        continue

    max_gain_db = 20.0 * np.log10(max_peak / peak)
    safe_target_values.append(current_lufs + max_gain_db)

if safe_target_values:
    target_lufs = min(
        target_lufs,
        min(safe_target_values),
    )

```

Codeblock 2: Bestimmung eines clipping-sicheren gemeinsamen Lautheitsziels, Quelle: Eigener Quellcode

Die eigentliche Anpassung der einzelnen Audiosignale erfolgt anschließend mit konstantem Gain auf den so bestimmten Zielwert. Die Funktionen zur LUFS-Messung und zur Anwendung des Gains werden hier aus Gründen der Übersichtlichkeit nicht zusätzlich dargestellt.

2.3 Zentraler Long-Form-Workflow

Der zentrale Long-Form-Workflow bündelt die zuvor getrennt implementierten Verarbeitungsschritte zu einem gemeinsamen Ablauf. Zunächst werden aus dem Eingabetext die Sprachsegmente erzeugt. Anschließend kann optional eine gemeinsame clipping-sichere Lautheitsnormalisierung angewendet werden. Abschließend werden die Segmente unter Berücksichtigung der konfigurierten Pausen zu einer zusammenhängenden Audiodatei zusammengesetzt.

```
# 1) Text segmentieren und jedes Segment mit derselben Stimme generieren.
segments = service.generate(
    text=text,
    reference_audio=speaker.reference_audio,
    reference_text=speaker.reference_text,
)

# 2) Optional alle Segmente auf ein gemeinsames clipping-sicheres Ziel an-
#     gleichen.
loudness_target_lufs = None
if normalize:
    segments, loudness_target_lufs = normalize_segments(segments)

# 3) Segmente mit der gewünschten zusätzlichen Pause aneinanderhängen.
assembled_audio = assemble_segments(
    segments,
    pause_seconds=pause_seconds,
)

return LongFormResult(
    segments=segments,
    assembled_audio=assembled_audio,
    loudness_target_lufs=loudness_target_lufs,
)
```

Codeblock 3: Zentraler Ablauf der Long-Form-Generierung, Quelle: Eigener Quellcode

Durch diese Bündelung bleibt die eigentliche Verarbeitungslogik unabhängig von der später hinzugefügten API und kann von unterschiedlichen Anwendungsschnittstellen wiederverwendet werden.

2.4 FIFO-Jobverwaltung

Die API verwaltet Generierungsaufträge in einer speicherinternen FIFO-Queue. Neu eingereichte Jobs werden in die Queue aufgenommen und anschließend von genau einem Hintergrund-Worker seriell verarbeitet. Dadurch können mehrere Anfragen entgegengenommen werden, ohne dass mehrere speicherintensive TTS-Generierungen gleichzeitig auf der GPU ausgeführt werden.

```
self._queue: Queue[tuple[UUID, Callable[[], dict]]] = Queue()

# Der Lock schützt den gemeinsamen Job-Speicher vor parallelem Zugriff.
self._lock = Lock()

# Ein einzelner daemonisierter Thread verhindert parallele GPU-Jobs.
self._worker = Thread(
    target=self._worker_loop,
    daemon=True,
    name="tts-job-worker",
)
self._worker.start()
```

Codeblock 4: Initialisierung der FIFO-Queue und des einzelnen Workers, Quelle: Eigener Quellcode

```
def _worker_loop(self) -> None:
    """Verarbeitet dauerhaft jeweils den nächsten Job aus der FIFO-Queue."""

    while True:
        job_id, task = self._queue.get()

        try:
            self._set_running(job_id)
            result = task()
        except Exception as exc:
            # Ein Fehler markiert nur diesen Job; der Worker läuft danach
            weiter.
            self._set_failed(job_id, str(exc))
        else:
            self._set_done(job_id, result)
        finally:
            self._queue.task_done()
```

Codeblock 5: Serielle Verarbeitung der Generierungsaufträge, Quelle: Eigener Quellcode

Die Zustände `queued`, `running`, `done` und `failed` werden während dieses Ablaufs im speicherinternen Job-Speicher aktualisiert; nach einem fehlgeschlagenen Auftrag setzt der Worker die Verarbeitung der nachfolgenden Jobs fort.

Anhang 3: Anwendungsschnittstelle

Dieser Anhang zeigt exemplarische Aufrufe der bereitgestellten HTTP-Schnittstelle. Dargestellt werden die Generierung mit einem vorhandenen Sprecherprofil, die Generierung mit einer benutzerdefinierten Referenzstimme sowie die Abfrage des resultierenden Jobstatus. Die Beispiele verwenden die lokal gestartete FastAPI-Anwendung.

3.1 Generierung mit Preset-Sprecher

```
curl -X POST http://127.0.0.1:8000/jobs/generate \
  -H "Content-Type: application/json" \
  -d '{
    "text": "Dies ist ein kurzer Beispieltext.",
    "engine": "qwen",
    "speaker": "tom_kalberg",
    "normalize": true,
    "pause_seconds": 0.25
  }'
```

Codeblock 6: Beispielanfrage für eine Generierung mit Preset-Sprecher, Quelle: Eigene Darstellung

```
{
  "id": "<UUID>",
  "status": "queued",
  "status_url": "/jobs/<UUID>"
}
```

Codeblock 7: Antwort nach erfolgreichem Einreihen des Generierungsauftrags, Quelle: Eigene Darstellung

3.2 Generierung mit benutzerdefiniertem Sprecher

Für eine benutzerdefinierte Referenzstimme wird der Endpunkt `POST /jobs/generate/custom` verwendet. Da neben den Konfigurationswerten auch eine WAV-Referenzaufnahme übertragen wird, erfolgt die Anfrage als `multipart/form-data`. Der zugehörige Referenztext kann entweder direkt als Textfeld oder alternativ als UTF-8-kodierte Textdatei übergeben werden.

```
curl -X POST http://127.0.0.1:8000/jobs/generate/custom \
  -F "text=Dies ist ein kurzer Beispieltext." \
  -F "engine=qwen" \
  -F "reference_audio=@reference.wav" \
  -F "reference_text=Dies ist der gesprochene Inhalt der Referenzaufnahme." \
  -F "normalize=true" \
  -F "pause_seconds=0.25"
```

Codeblock 8: Beispielanfrage für eine Generierung mit benutzerdefiniertem Sprecher, Quelle: Eigene Darstellung

Die erfolgreiche Anfrage wird wie bei der Preset-Generierung mit HTTP-Status 202 beantwortet. Die Antwort enthält die Job-ID, den Status `queued` sowie die URL zur Statusabfrage. Die hochgeladene Referenzaufnahme wird bis zur späteren Verarbeitung des Queue-Jobs temporär gespeichert und anschließend wieder entfernt.

3.3 Abfrage eines abgeschlossenen Jobs

Der Zustand eines eingereichten Auftrags kann über `GET /jobs/{job_id}` abgefragt werden. Während der Verarbeitung wechselt der Status von `queued` zu `running`. Nach erfolgreichem Abschluss besitzt der Job den Status `done` und enthält zusätzlich die Ergebnisinformationen der Generierung.

```
{
  "id": "<UUID>",
  "status": "done",
  "engine": "qwen",
  "result": {
    "output_file": "outputs/api/<AUDIO-UUID>.wav",
    "download_url": "/audio/<AUDIO-UUID>",
    "seed": "<SEED>",
    "segments": "<SEGMENTANZAHL>",
    "sample_rate": 24000,
    "duration_seconds": "<DAUER>",
    "loudness_target_lufs": "<LUFS>"
  },
  "error": null
}
```

Codeblock 9: Beispielhafte Antwort eines abgeschlossenen Generierungsauftrags, Quelle: Eigene Darstellung

Die erzeugte WAV-Datei kann anschließend über den in `download_url` angegebenen Endpunkt abgerufen werden.

Anhang 4: Installationsanleitung

Dieser Anhang beschreibt die Einrichtung und Inbetriebnahme des entwickelten Prototyps auf einem Windows-System mit WSL. Die Anleitung orientiert sich an der getesteten Referenzumgebung und fasst die für eine erneute Installation erforderlichen Schritte zusammen. Eine ausführlichere Fassung liegt dem Softwareartefakt zusätzlich als `ANLEITUNG.md` bei.

4.1 Voraussetzungen

Für die vollständige Nutzung des Prototyps wird ein Windows-System mit WSL2 und einer Ubuntu-Distribution vorausgesetzt. Die Sprachgenerierung benötigt eine CUDA-fähige NVIDIA-Grafikkarte. Für die Verwendung beider integrierten TTS-Engines werden mindestens 12 GB Grafikspeicher empfohlen, während die getestete Referenzumgebung über 16 GB VRAM verfügt. Zusätzlich wird während der Installation sowie beim erstmaligen Laden der Modellgewichte eine Internetverbindung benötigt. AMD-GPUs werden von der bereitgestellten Installationsroutine nicht unterstützt.

Die Modellgewichte sind nicht Bestandteil des Projektverzeichnisses. Sie werden beim ersten Einsatz der jeweiligen TTS-Engine automatisch heruntergeladen und anschließend lokal zwischengespeichert. Der erste Generierungsauftrag kann daher deutlich länger dauern als spätere Aufrufe.

4.2 Einrichtung von WSL

Der Prototyp wird unter Linux ausgeführt. Auf einem Windows-System kann hierfür WSL2 mit Ubuntu verwendet werden. Falls WSL noch nicht eingerichtet ist, wird zunächst Windows PowerShell beziehungsweise Windows Terminal mit Administratorrechten geöffnet und Ubuntu mit folgendem Befehl installiert:

```
wsl --install -d Ubuntu
```

Falls Windows anschließend zu einem Neustart auffordert, wird dieser durchgeführt. Beim ersten Start von Ubuntu werden ein Linux-Benutzername und ein Passwort festgelegt. Danach sollte die installierte Distribution einmal aktualisiert werden:

```
sudo apt update  
sudo apt upgrade -y
```

Ist WSL2 mit einer funktionsfähigen Ubuntu-Distribution bereits vorhanden, kann dieser Einrichtungsschritt übersprungen werden.

4.3 Prüfung der NVIDIA GPU

Vor der Installation sollte geprüft werden, ob die NVIDIA-GPU innerhalb von WSL verfügbar ist. Dazu wird im Ubuntu-Terminal folgender Befehl ausgeführt:

```
nvidia-smi
```

Wird eine Tabelle mit Informationen zur Grafikkarte, zum NVIDIA-Treiber und zur Speicherauslastung angezeigt, ist die GPU grundsätzlich über WSL erreichbar. Falls der Befehl fehlschlägt, sollte zunächst der NVIDIA-Treiber unter Windows aktualisiert und das System anschließend neu gestartet werden. Ein separater Linux-NVIDIA-Treiber sollte innerhalb von WSL nicht installiert werden.

4.4 Installation von Miniconda

Für die Einrichtung der Python-Umgebung wird Miniconda verwendet. Falls Conda noch nicht installiert ist, kann die aktuelle Linux-Version im Ubuntu-Terminal heruntergeladen und anschließend gestartet werden:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh  
  
bash Miniconda3-latest-Linux-x86_64.sh
```

Während der Installation kann der vorgeschlagene Installationspfad übernommen werden. Die Initialisierung von Conda für die Shell sollte aktiviert werden. Anschließend wird das Ubuntu-Terminal neu geöffnet und die Installation überprüft:

```
conda --version
```

Wird eine Versionsnummer ausgegeben, ist Miniconda erfolgreich eingerichtet. Ist Conda bereits vorhanden, kann dieser Schritt übersprungen werden.

4.5 Automatisierte Installation des Prototyps

Nachdem das Projekt in das Linux-Dateisystem von WSL kopiert wurde, wird im Ubuntu-Terminal in das Projektverzeichnis gewechselt. Anschließend kann die vollständige Einrichtung über das mitgelieferte Installationsskript gestartet werden:

```
cd ~/longform-tts  
bash install.sh
```

Das Skript prüft zunächst, ob eine Linux- beziehungsweise WSL-Umgebung, Conda und eine erreichbare NVIDIA-GPU vorhanden sind. Danach wird die Conda-Umgebung `longform-tts` angelegt, die getestete PyTorch-Version mit CUDA-Unterstützung installiert und anschließend werden die übrigen Python-Abhängigkeiten sowie das Projekt selbst eingerichtet. Abschließend wird die Installation automatisch mit `check_environment.py` überprüft.

Ist bereits eine Conda-Umgebung mit dem Namen `longform-tts` vorhanden, bricht das Installationsskript ab. Für eine saubere Neuinstallation muss diese Umgebung zunächst entfernt werden. Nach erfolgreichem Abschluss kann die neu angelegte Umgebung mit `conda activate longform-tts` aktiviert werden.

4.6 Start der Anwendung

Nach erfolgreicher Installation wird zunächst die Conda-Umgebung aktiviert und anschließend aus dem Projektverzeichnis der lokale FastAPI-Server gestartet:

```
cd ~/longform-tts  
conda activate longform-tts  
uvicorn longform_tts.api.app:app --no-access-log
```

Solange der Server läuft, bleibt das Terminal geöffnet. Die Weboberfläche kann anschließend im Browser unter `http://127.0.0.1:8000/` aufgerufen werden. Zusätzlich stellt FastAPI unter `http://127.0.0.1:8000/docs` eine automatisch erzeugte interaktive API-Dokumentation bereit.

4.7 Bedienung der Weboberfläche

Nach dem Aufruf der Weboberfläche kann zunächst der zu synthetisierende Text eingegeben werden. Anschließend wird eine der beiden verfügbaren TTS-Engines ausgewählt. Für die Sprecherkonditionierung kann entweder ein vorhandenes Preset-Sprecherprofil verwendet oder in den Modus für eine eigene Referenzstimme gewechselt werden. In diesem Fall werden eine WAV-Datei mit der Referenzaufnahme sowie das zugehörige Transkript benötigt. Das Transkript kann direkt eingegeben oder als Textdatei hochgeladen werden.

Zusätzlich kann die Lautheitsnormalisierung aktiviert beziehungsweise deaktiviert und die Pause zwischen den erzeugten Segmenten angepasst werden. Der Standardwert für die Segmentpause beträgt 0,25 Sekunden. Mit der Schaltfläche „Audio generieren“ wird der Auftrag anschließend in die Warteschlange eingereicht.

Der aktuelle Zustand des Auftrags wird in der Jobliste angezeigt. Dabei werden die Zustände `queued`, `running`, `done` und `failed` unterschieden. Nach erfolgreichem Abschluss kann die erzeugte Audio-Datei direkt im Browser abgespielt oder als WAV-Datei heruntergeladen werden. Mehrere Aufträge können nacheinander eingereicht werden und werden durch die FIFO-Jobverwaltung seriell verarbeitet.

4.8 Erzeugte Audiodateien und Neustart

Die über die Weboberfläche erzeugten WAV-Dateien werden im Projektverzeichnis unter `outputs/api/` gespeichert. Die Job-Metadaten selbst werden nur im Arbeitsspeicher gehalten. Beim Neustart der Anwendung wird die Jobliste daher zurückgesetzt, bereits vollständig erzeugte WAV-Dateien bleiben jedoch im Ausgabeverzeichnis erhalten.

Zum Beenden der Anwendung kann der laufende Uvicorn-Prozess im Ubuntu-Terminal mit `Strg+C` gestoppt werden. Für einen späteren Neustart genügt es normalerweise, erneut in das Projektverzeichnis zu wechseln, die Conda-Umgebung zu aktivieren und den Server zu starten:

```
cd ~/longform-tts
conda activate longform-tts
uvicorn longform_tts.api.app:app --no-access-log
```

Anschließend ist die Weboberfläche wieder unter `http://127.0.0.1:8000/` erreichbar.

4.9 Häufige Probleme

conda: command not found

Wird Conda nach der Installation nicht gefunden, sollte das Ubuntu-Terminal zunächst geschlossen und neu geöffnet werden. Falls der Befehl weiterhin nicht verfügbar ist, kann die Conda-Initialisierung manuell geladen werden:

```
source ~/miniconda3/etc/profile.d/conda.sh
```

nvidia-smi funktioniert nicht

In diesem Fall ist die NVIDIA-GPU innerhalb von WSL nicht korrekt verfügbar. Zunächst sollte der NVIDIA-Treiber unter Windows aktualisiert und das System anschließend neu gestartet werden. Ein separater NVIDIA-Treiber innerhalb von WSL sollte nicht installiert werden.

Die Conda-Umgebung longform-tts existiert bereits

Das Installationsskript erwartet für eine reproduzierbare Neuinstallation eine noch nicht vorhandene Umgebung. Eine bestehende Umgebung kann daher zunächst entfernt werden:

```
conda env remove -n longform-tts  
  
bash install.sh
```

Der erste Generierungsauftrag dauert ungewöhnlich lange

Beim ersten Einsatz einer TTS-Engine werden die zugehörigen Modellgewichte automatisch heruntergeladen. Abhängig von Internetverbindung und Modellgröße kann dieser Vorgang mehrere Minuten dauern. Nach dem ersten Download werden die Dateien lokal zwischengespeichert und bei späteren Aufrufen wiederverwendet.

Weboberfläche ist nicht erreichbar

In diesem Fall sollte geprüft werden, ob der Uvicorn-Prozess im Ubuntu-Terminal weiterhin läuft und dort keine Fehlermeldung ausgegeben wurde. Anschließend kann die Weboberfläche erneut unter <http://127.0.0.1:8000/> aufgerufen werden.

Anhang 5: Softwarestand und Reproduzierbarkeit

Dieser Anhang dokumentiert den für die Arbeit maßgeblichen Softwarestand und die für eine Reproduktion relevanten Artefakte. Neben dem bereitgestellten Quellcode werden die verwendeten Modellrevisionen sowie die zentralen Dateien zur Wiederherstellung der getesteten Softwareumgebung aufgeführt.

5.1 Bereitgestellter Softwarestand

Der vollständige Quellcode des entwickelten Prototyps wird den Prüfern als Bestandteil des Softwareartefakts bereitgestellt. Der identische Abgabestand ist sowohl auf den der Arbeit beiliegenden USB-Datenträgern als auch über ein privates GitHub-Repository verfügbar. Alle in dieser Arbeit beschriebenen Implementierungsdetails, Installationsschritte und Evaluationen beziehen sich auf diesen Softwarestand.

5.2 Verwendete Modellrevisionen

Um Veränderungen der extern bereitgestellten Modellgewichte zu vermeiden und den für die Evaluation verwendeten Zustand reproduzierbar zu halten, werden beide TTS-Modelle über fest vorgegebene Hugging-Face-Revisionen geladen. Beim erstmaligen Einsatz lädt der Prototyp den jeweils angegebenen Snapshot herunter und verwendet diesen anschließend als lokale Modellquelle.

TTS-Engine	Hugging-Face-Modell	Verwendete Revision
Qwen3-TTS 1.7B Base	Qwen/Qwen3-TTS-12Hz-1.7B-	fd4b254389122332181a7
	Base	c3db7f27e918eec64e3
VoxCPM2	openbmb/VoxCPM2	32279effe8c19989596f0
		5d353d1447f51d9e915

Die Revisionen sind direkt in den jeweiligen Engine-Adaptern hinterlegt und werden beim Aufruf von `snapshot_download()` explizit übergeben. Dadurch wird nicht automatisch der jeweils aktuelle Stand des Modellrepositorys verwendet.

5.3 Dateien zur Reproduzierbarkeit

Zur Wiederherstellung der getesteten Softwareumgebung enthält das Softwareartefakt mehrere ergänzende Konfigurations- und Installationsdateien. Die wichtigsten Dateien und ihre jeweilige Funktion sind in der folgenden Tabelle zusammengefasst.

Datei	Funktion
<code>environment.yml</code>	Definition der Conda-Umgebung und Festlegung der verwendeten Python-Version sowie grundlegender Paketwerkzeuge
<code>requirements.txt</code>	Festlegung der getesteten Python-Abhängigkeiten und ihrer Versionen
<code>conda-explicit.txt</code>	Expliziter Snapshot der in der Referenzumgebung installierten Conda-Pakete für <code>linux-64</code>
<code>install.sh</code>	Automatisierte Einrichtung der Conda-Umgebung, Installation der Abhängigkeiten und abschließende Prüfung
<code>check_environment.py</code>	Überprüfung von Python, zentralen Paketversionen, CUDA-Verfügbarkeit und Projektimporten
<code>ANLEITUNG.md</code>	Ausführliche Schritt-für-Schritt-Anleitung zur Installation, Inbetriebnahme und Fehlerbehebung

Zusammen mit den festgelegten Modellrevisionen und dem bereitgestellten Quellcode ermöglichen diese Dateien eine möglichst genaue Rekonstruktion des für Entwicklung und Evaluation verwendeten Softwarestands.

Eigenständigkeitserklärung

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

Automatisierung der KI-basierten Generierung längerer Audioinhalte — Konzeption und Realisierung eines lokalen Systems mit Voice Cloning

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