Fakultät Wirtschaft

Studiengang Data Science und Künstliche Intelligenz

Implementierung einer KI-gestützten Applikation mit Python zur Kuratierung von YouTube-Inhalten

Projektbericht

Verfasser:	**Adrian Siebnich,**
	Melvin Manning,
	Ricky Nguyen,
Kurs:	**WDS24B**
Dualer Partner:	**Robert Bosch GmbH, Karlsruhe**
Abgabedatum:	**25.04.2025**
GitHub:	https://github.com/melvinmng/DataScience_Projekt

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich den vorliegenden Projektbericht mit dem Thema „Implementierung einer KI-gestützten Applikation mit Python zur Kuratierung von YouTube-Inhalten“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 25.04.2025 _____

Adrian Siebnich

Karlsruhe, 25.04.2025 _____

Melvin Manning

Karlsruhe, 25.04.2025 _____

Ricky Nguyen

Kurzfassung

Als eines der weltweit größten sozialen Netzwerke bietet YouTube die Plattform für unzählige Videos in vielfältigen Kategorien. Nutzer können gleichermaßen von dieser Auswahl profitieren oder überfordert sein.

Der vorliegende Projektbericht dokumentiert die Implementierung einer KI-gestützten Applikation in Python, in deren Zentrum die Frage steht, wie sich Inhalte von YouTube nutzerspezifisch strukturieren lassen. Durch den Einsatz künstlicher Intelligenz sollen Videos zusammengefasst und auf Clickbait analysiert werden können. Dadurch sollen zeitliche, qualitative sowie interessensbezogene Mehrwerte erzielt werden.

Das Resultat dieser Zielsetzung ist eine multifunktionale Anwendung, welche Schnittstellen zu YouTube und Gemini anbindet und mit welcher der Nutzer mittels einer grafischen Darstellung durch Streamlit in Interaktion treten kann.

Abstract

As one of the world's largest social networks, YouTube offers a platform for countless videos in a broad variety of categories. Users can benefit from this selection or be overwhelmed in equal measure.

This project report documents the implementation of an AI-supported application in Python, which focuses on the question of how YouTube content can be structured in a user-specific way. By using artificial intelligence, videos can be summarized and analyzed for clickbait. The aim is to achieve added value in terms of time, quality and interest.

The result of this objective is a multifunctional application that connects interfaces to YouTube and Gemini and with which the user can interact by means of a graphical representation through Streamlit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API – Application Programming Interface

BERT – Bidirectional Encoder Representations from Transformers

CLI – Command Line Interface

FoBO – Fear of Better Options

FoMO – Fear of Missing Out

GUI – Graphical User Interface

Gemini API – Generative Language API

NLP – Natural Language Processing

UI – User Interface

YouTube API – YouTube Data API v3

1 Umgang mit der Informationsdichte auf YouTube

1.1 Motivation

Nach der Gründung 2005 und der Akquisition durch Google 2006 (US Securities and Exchange Comission, 2006) zählt YouTube heute zu den weltweit meistbesuchten sozialen Netzwerken. Nutzer können Videos aus einer Vielzahl an Kategorien konsumieren oder eigene Inhalte produzieren. Die Plattform beschränkt sich nicht auf den privaten Bereich, sondern wird auch professionell betrieben (YouTube, 2025). Angesichts der daraus resultierenden Dichte an verfügbarem Videomaterial stellt sich die Frage, wie sich diese Inhalte sinnvoll organisieren lassen, um den Zugang zu relevanten Themen zu erleichtern.

1.2 Markt- und Literaturrecherche

Mittlerweile ist YouTube als Freemium-Modell aufgebaut. Auf die Veröffentlichung des kostenpflichtigen „YouTube Red“ (Leske, 2015) erfolgte mit „YouTube Premium“ (YouTube, 2018) eine Umbenennung. Bereits angebotene KI-basierte Funktionen sind oft regional beziehungsweise auf die kostenpflichtige Version eingeschränkt, etwa Funktionen zur Konversation mit Sprachassistenten, Videozusammenfassungen, personalisierte Zusammenstellungen von Musik, Ideenvorschläge für Videoersteller oder Audioübersetzungen (YouTube, 2024).

Der Chrome Web Store bietet mehrere Erweiterungen, mit denen KI-generierte Zusammenfassungen für YouTube-Videos möglich sein sollen. Beispielsweise arbeitet die Erweiterung „YouTube Summary with ChatGPT & Claude“ von glasp.co mit Modellen von ChatGPT, Claude, Mistral AI und Gemini (Google, 2025).

Puranik et al. (2024) schlagen eine Systemarchitektur vor, bei der ein Nutzer den Link zu einem YouTube-Video in einer Webanwendung einfügen kann und eine textbasierte sowie eine visuelle Zusammenfassung erhält. Das Natural Language Processing (NLP) wird in Python implementiert und stützt sich auch ein vortrainiertes Bidirectional Encoder Representations from Transformers (BERT) Modell, bei dem Sätze aus beiden Richtungen gelesen werden und ein Kontext der Wörter hergestellt wird.

1.3 Zielsetzung

Das wesentliche Ziel dieses Projekts ist es, eine Applikation mit Python zu entwickeln, mit der Inhalte von YouTube nutzerspezifisch bereitgestellt werden. Durch Integration einer künstlichen Intelligenz sollen Videos zusammengefasst und vorgeschlagen werden. Die vorgefundene Informationsfülle soll dadurch strukturiert und in selektivem Umfang (visuell) aufbereitet werden. Diese Kuratierung soll in einer multifunktionalen, personalisierten und interaktiven grafischen Benutzeroberfläche dargestellt werden.

1.4 Use Cases

Die Funktionalität des Programms adressiert verschiedenste Nutzer von YouTube, wobei übergeordnet zwischen Konsumenten und Produzenten unterschieden werden kann. Auch, wenn der Fokus dieser Entwicklung auf Ersteren liegt, bietet die Anwendung aus Sicht der Ersteller einen Nutzen.

Drei grundlegende Bestandteile der YouTube App beziehungsweise Website umfassen die Suche nach Videos, aktuelle Trendvideos sowie die Möglichkeit, Kanäle zu abonnieren. Diese bilden auch einen Kern dieses Programms. Ein wesentlicher Unterschied ist die Verfügbarkeit von Zusammenfassungen. Der Mehrwert besteht folglich in einer erleichterten Entscheidungsfindung bei der Videoauswahl, Prägnanz der Inhalte und einer Zeitersparnis. Zum Beispiel kann sich ein Nutzer bei der Aneignung einer neuen Tätigkeit einfacher in ein Thema einfinden oder beim Lernen durch Erklärvideos schneller mehrere Quellen vergleichen. Neben diesen vorwiegend quantitativen Aspekten kann durch die Einbindung nutzerspezifischer Interessen auch eine qualitative Steigerung der Inhalte erzielt werden, wofür typische Filter nach Datum, Dauer oder Kategorie meist unzureichend sind.

Auch Videoproduzenten bieten die vorgeschlagenen Funktionen der Applikation wertvolle Einblicke in aktuelle Inhalte von YouTube. Denkbar ist die Nutzung des Programms zur Analyse beliebter Inhalte, insbesondere ähnlicher und somit konkurrierender Inhalte. Daraus könnte eine gezieltere Ausrichtung an eine Zielgruppe oder im professionellen Umfeld eine verbesserte Orientierung bezüglich der Monetarisierung abgeleitet werden. Ebenfalls kann die Software ein Anhaltspunkt bei der Inspiration zur Generierung neuer Inhalte sein.

2 Implementierung der Applikation

2.1 Methodik

Die Vorgehensweise bei der Softwareentwicklung orientiert sich an Elementen des Spiralmodells nach Barry W. Boehm (1986). Die dadurch gewonnene Agilität bietet im Rahmen der kurzen Entwicklungszeit zeitliche Vorteile und ermöglicht es, als Entwickler und Nutzer zugleich die Anforderungen an das Projekt im Arbeitsprozess anzupassen, woraus eine progressive Näherung an die Zielsetzung folgt.

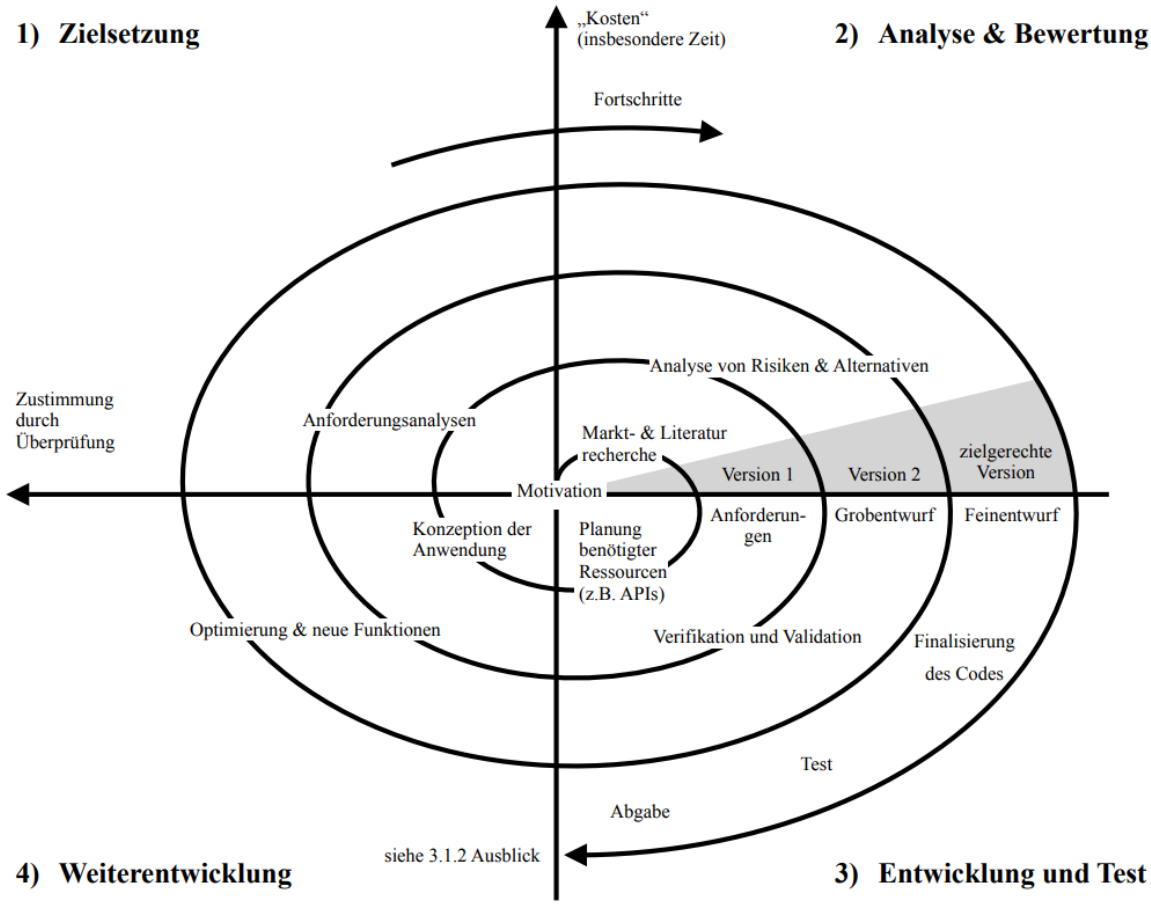

Abb. 1: Methodik zur Entwicklung der Anwendung; eigene Darstellung nach (Boehm, 1986)

Für die Implementierung des Programms wird als Sprache Python gewählt. Eine Top-Down-Ansicht erlaubt ausgehend von der Konzeption und Zielsetzung eine Modularisierung beziehungsweise funktionale Gruppierung des Codes. Anhand dessen werden im Folgenden auch die Projektschritte strukturiert.

2.2 Konezptionelle Architektur der Anwendung

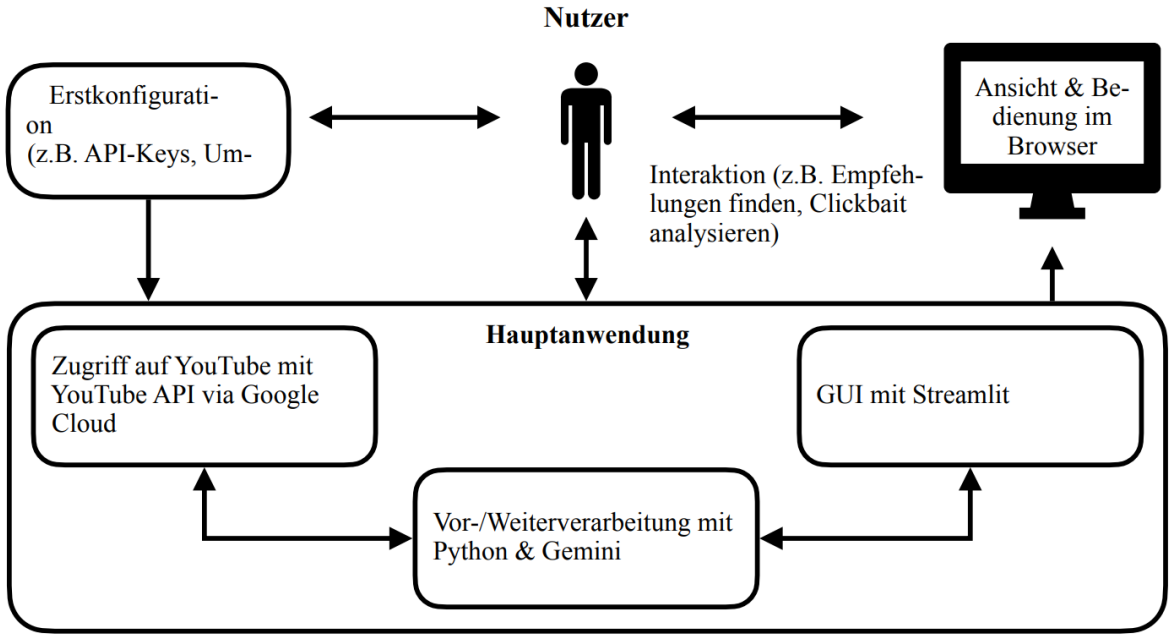

Abb. 2: Konzeptionelle Architektur der Anwendung

2.3 Projektschritte bei der Implementierung

2.3.1 Schnittstellen

Zwei Application Programming Interfaces (APIs) ermöglichen die Kommunikation zwischen verschiedenen Python-Skripten und YouTube beziehungsweise Gemini zur KI-basierten Sprachverarbeitung: die YouTube Data API v3 (im Folgenden YouTube API) und die Generative Language API (im Folgenden Gemini API).

Diese APIs wurden gewählt, da sie kostenlos zur Verfügung stehen und beide über die Google Cloud Console aktiviert werden können, was nutzerseitig zu einer gewissen Vereinheitlichung führt..

Ein limitierender Faktor bei der privaten, unbezahlten Nutzung der YouTube API ist die Beschränkung der täglich verfügbaren Menge an Zugriffen. Das Kostenkontingent von 10.000 Punkten pro Tag genügt den meisten Anfragen im Rahmen dieses Projekts. Durch Caching und lokales Speichern, beispielsweise in Tabellen, können Zugriffe mit hohen Kosten, das heißt häufig benötigte oder punktintensivere Operationen, reduziert werden. Zugleich sinken dadurch die Ladezeiten. Eine Alternative zur offiziellen YouTube API ist das

open source Python-Modul yt-dlp (Python Package Index, 2025). Das Command Line Interface bietet mitunter ähnliche Funktionen wie die YouTube API. Standardmäßig greift das Programm auf die offizielle API zu, yt-dlp ergänzt dies um eine experimentelle Funktionalität, die ohne Kostenkontingent und API Key auskommt. Dies macht die offizielle Schnittstelle nicht obsolet, sondern dient der verbesserten Zuverlässigkeit und Unabhängigkeit.

Bei der Gemini API ist hinsichtlich der Einheitlichkeit des Google „Ökosystems“ ein möglicher Bias des Sprachmodells kritisch zu bewerten. Im Rahmen dieses Projekts wurde darauf verzichtet, jedoch ermöglicht die modulare Gestaltung des Programmes alternativ oder erweiternd die Einbindung anderer oder eigens antrainierter Sprachmodelle.

2.3.2 Schlüsselerzeugung und Erstkonfiguration

Zur Nutzung der Schnittstellen muss ein Nutzer seine API-Schlüssel individuell per Google Cloud Console generieren lassen und diese lokal hinterlegen. Der Speicherort dafür ist eine .env-Datei, welche die Keys aufbewahrt. Zur Steigerung der Benutzerfreundlichkeit besteht die Möglichkeit, bei der ersten Nutzung die Schlüssel auf der GUI einzutragen, sodass das Programm das Abspeichern der Umgebungsvariablen automatisiert. Später lassen sich die Keys in den Einstellungen verwalten.

Zugriffe auf nutzerspezifische Daten bedürfen einem Zwischenschritt. Das ist beispielsweise für den Zugang zu den Abonnements relevant. Als dritte Umgebungsvariable kommt daher die Kanal-ID von YouTube in Frage. Sie ist optional und ermöglicht Funktionen zum Abrufen abonnierter Kanäle sowie KI-basierter, personalisierter Videovorschläge.

Kritisch zu betrachten ist in diesem Kontext die Implementierung für das Abrufen der Abonnements. Zwar ermöglicht die YouTube API einen Zugriff, jedoch muss der Nutzer dafür seine Abonnements öffentlich sichtbar machen. Dadurch ergeben sich Nachteile hinsichtlich des Datenschutzes und der Benutzerfreundlichkeit insofern, als dass weitere Vorbereitungen notwendig sind.

Eine Alternative dazu ist Auth0, der „De-facto-Standard für Online-Authentifizierung“ (Ferry, et al., 2015, p. 75). Dieser Standard ermöglicht eine Verifizierung mit Tokens. Bei dieser Lösung könnte ein Nutzer nach erfolgreicher Anmeldung auf sein Google-Konto zugreifen, ohne sein Passwort an die Applikation weiterzugeben (Ferry, et al., 2015, pp. 75-76). Google Cloud Console bietet das Einrichten eines 0Auth-2.0-Zustimmungsbildschirms

an. Die Integration in die bestehende API-Implementierung ist technisch problemlos möglich, jedoch erfordert Google die Verifizierung eines Projekts, sofern 0Auth genutzt wird. Andernfalls kann das Projekt nur eingeschränkt genutzt werden, beispielsweise müssten alle Nutzer oder Tester manuell eingetragen werden (Google, 2025).

Aus diesen Gründen basiert die finale Implementierung auf einer verbesserten Zwischenlösung, die erfordert, Abonnements kurzzeitig öffentlich zugänglich zu machen, um sie daraufhin lokal in einer csv-Datei zu speichern. Für zukünftige Weiterentwicklungen ist eine etablierte Authentifizierungsmethode wie Auth0 die präferierte Wahl.

2.3.3 Sprachverarbeitung

Die Basis der Sprachverarbeitung bildet das Transkript des jeweiligen Videos. Kombiniert mit einem Prompt lassen sich daraus mit Gemini Zusammenfassungen generieren. Bei der Instanz des Modells handelt es sich um „Gemini 2.0 Flash“. Es bietet neben der kostenfreien Verfügbarkeit weitere Vorteile hinsichtlich Geschwindigkeit, Suchfunktionen und Unterstützung multimodaler Ein- und Ausgabedatentypen. So sind Anforderungen an ein moderate Antwortzeiten, der Recherche von Hintergrundinformationen zu Videoinhalten sowie dem Verarbeiten der Metadaten gedeckt. Des Weiteren erlaubt das Kontextfenster in Höhe von einer Million Tokens eine umfangreiche Nutzung (Google, 2025).

Zu den wesentlichen Herausforderung zählen die mitunter unzureichende Verfügbarkeit der Transkripte, die Qualität der Transkripte, der Umgang mit großen Zeitspannen hinsichtlich der Videodauer sowie den Umgang mit dem aus Nutzersicht möglicherweise unerwünschten Verraten von Informationen (z.B. „Spoiler“ bei Sportergebnissen). Um zukünftig auch bei fehlenden Transkripten ein zielgerechtes Ergebnis bereitstellen zu können, sind weitere Schritte Quellen für Zusammenfassungen in Erwägung zu ziehen. Dabei könnte es sich zum Beispiel um Funktionalitäten handeln, die selbst Audio transkribieren und automatische Untertitel erzeugen können.

2.3.4 Clickbait-Analyse

Neben der quantitativen Auswahl etwa auf Basis der Videodauer sollen auch qualitative Aspekte, darunter vor allem eine Bewertung hinsichtlich „Clickbait“ in Betracht gezogen werden. Eine allgemein gültige Definition von Clickbait existiert nicht (Kemm, 2022). In diesem

Fall meint Clickbait die Technik, Inhalte auf YouTube so zu gestalten, dass sie von möglichst vielen Nutzern abgerufen werden, wobei die Inhalte gar nicht oder in geringerem Maß als beworben übermittelt werden.

Aus diesem Ansatz ergibt sich das Problem, eine sinnvolle Metrik für die Übereinstimmung von beworbenen und tatsächlichen Videos zu definieren. Konkrete Parameter hierfür können Titel, Thumbnail (Vorschaubild) und insbesondere deren Vergleich mit dem Inhalt des Videos, das heißt Bildsequenzen oder Transkript sein.

Thumbnails können vom Ersteller des Videos selbst gewählt werden, weshalb Ansätze wie „Smart Thumbnails“ von YouTube (2009) weiter ausgebaut werden könnten. Das Hauptziel ist dabei, repräsentative Ausschnitte eines Videos zu identifizieren.

2.3.5 Web-Applikation mit Streamlit

Um die Funktionalitäten der Applikation mit der Bedienung beziehungsweise den Use Cases aus Nutzersicht zu verknüpfen, ist ein User Interface (UI) sinnvoll. Der Anteil an Bestandteilen eines Command Line Interface (CLI) ist bewusst kleinstmöglich gehalten. Stattdessen kommt ein Graphical User Interface (GUI) zum Einsatz.

Das open-source Framework „Streamlit” liefert in diesem Kontext die benötigte Infrastruktur. Es funktioniert nach dem Client-Server-Prinzip, wobei das Backend in Python dem Server und das Frontend im Browser dem Client entspricht und beide auf derselben Maschine laufen (Streamlit, 2025).

Dem Nutzer wird durch die grafische Ausgabe eine möglichst intuitive Interaktion unabhängig des technischen Verständnisses angeboten, sodass lediglich ein Skript über die Kommandozeile gestartet werden muss. Zugleich erleichtert dies aus Entwicklersicht den Fokus auf Hintergrundprozesse, wenngleich sich Front- und Backend gegenseitig bedingen. So basiert der Aufbau der Oberfläche auf Tabs und einer Seitenleiste, was implizit mit der funktionalen Gruppierung des Codes einhergeht. Bezüglich möglichen Erweiterungen lässt sich diese Interdependenz ebenso feststellen.

2.4 Betrachtungen der Softwarequalität

2.4.1 Performance und Ladezeiten

Einer der wichtigsten Faktoren bezüglich der Nutzererfahrung sind Lade- und somit Wartezeiten. Ohne weitere Maßnahmen würden diese im Vergleich zur direkten Nutzung von YouTube oder Gemini deutlich höher ausfallen. Die Begründung liegt unter anderem in den verschiedenen Zugriffen auf die YouTube und Gemini API, in der Weiterverarbeitung in Python sowie in der Übermittlung an Streamlit. Die Implementierung der Optimierungsmaßnahmen bezieht sich vor allem auf die letzten beiden Punkte.

Nutzerseitig führen entsprechende Bedienelemente, etwa Knöpfe, dazu, dass Inhalte selektiver und erst auf Abruf geladen werden, was übermäßiges Laden eventuell gar nicht nachgefragter Inhalte mindert.

Streamlit integriert bereits Threads in der Architektur und unterstützt offiziell kein Multithreading, was eigenes Multithreading erschwert. Unterschieden wird zwischen serverseitigen Threads und Script Threads. Bei Letzteren existiert ein Thread je ausgeführtem Skript einer Sitzung (Session) (Streamlit, 2025). Bei dieser Applikation tritt das im Kontext der Funktion st.session_state auf, welche es ermöglicht, „[…] Variablen zwischen verschiedenen Durchläufen für jede Benutzersitzung gemeinsam zu nutzen […] Der Sitzungsstatus bleibt auch innerhalb einer mehrseitigen Anwendung bestehen.“ (Streamlit, 2025). Durch die entsprechende Anpassung der Logik bei der Tabstruktur kann festgestellt werden in welchem Tab sich der Nutzer befindet oder befunden hat. Das reduziert das erneute Laden beim Wechseln des Tabs. Zusätzlich wird diese Technik beispielsweise genutzt, um Auflistungen von Videos, Suchen oder Statusinformationen zum Aufrufen von Knöpfen zur Laufzeit beizubehalten.

Weitere Ressourcen wie Rechenleistung, Zeit und API-Aufrufe können durch Caching eingespart werden, indem Funktionen nicht mehrmals aufgerufen werden und Objekte auch nicht mehrfach erzeugt werden müssen, sondern über einen Zwischenspeicher erhalten bleiben. Dafür werden die Daten „[…] über Pickle serialisiert (in Bytes umgewandelt) […]“ (Streamlit, 2025). Ein wesentlicher Unterschied zu st.session_state ist, dass mit dem Decorator @st.cache_data(ttl=3600) Ergebnisse beziehungsweise Aufrufe einer Funktion für eine gewissen Zeitraum (hier eine Stunde) im Cache zwischengespeichert werden.

Zusätzlich existieren im Programm einzelne Funktionalitäten, für die lokale Dateien im Projektverzeichnis zum Speichern vorgesehen sind. Neben den Abonnements gehören die nutzerspezifischen Dateien für Interessen, Videos zum später Anschauen sowie deren Verlauf dazu. Außerdem werden abgesendete Rückmeldungen aus dem Feedback-Tab in einer Datei im Projektverzeichnis abgelegt. Bei einem tatsächlichen Deployment ist hierfür eine Datenbankanbindung oder Weiterleitung an eine E-Mail-Adresse in Erwägung zu ziehen.

Schließlich kann die deutlichste Performancesteigerung durch Multiprocessing erzielt werden. Hierfür kommt die Python-Bibliothek „multiprocessing“ zum Einsatz. Diese „[…] umgeht das Global Interpreter Lock durch die Verwendung von Subprozessen anstelle von Threads […]“ (Python, 2025). Der prozessbasierte Ansatz profitiert von der Parallelisierung über mehrere Prozessorkerne, wobei die Anzahl individuell mit der Funktion multiprocessing.cpu_count() ermittelt wird. Insgesamt werden in Kombination mit den anderen Maßnahmen zur Effizienzsteigerung Ladezeiten erheblich verringert und Ressourcen besser allokiert.

2.4.2 Testabdeckung

Die Testabdeckung des Codes beschränkt sich auf Unit Tests. Das Testen kleiner, isolierter Einheiten erweist sich im Rahmen des Projekts als sinnvoll, da eine ganzheitliche Systemverifikation insbesondere durch die Anbindung der APIs und Streamlit zu komplex ist.

Zur Unterstützung der Software Tests kommt vorrangig das Framework pytest zum Einsatz. Im Allgemeinen folgen die Testverfahren dem gängigen Ablauf bestehend aus Vorbereitung, Ausführung der zu testenden Funktion, Abgleich der Ergebnisse und der Nachbereitung in Form einer Bereinigung (pytest, 2015). Das Framwork erleichtert eine konsistente Implementierung der Tests und bietet umfangreiche Statusrückmeldungungen. An dieser Stelle sei darauf hingewiesen, dass sich drei Tests mit dem Decorator @pytest.mark.skipif auf macOS beziehungsweise Unix-basierte Betriebssysteme beschränken. Diese Tests stehen im Zusammenhang mit dem Neustart von Streamlit, der je nach Betriebssystem nicht einheitlich gehandhabt wird, was betriebssystemübergreifende Tests erschwert. Für Windows werden diese Einheiten übersprungen.

Des Weiteren hängt ein Großteil der Methoden von der Anbindung an die APIs und Streamlit ab. Ein gemeinsames Testen würde dem Verfahren mit isolierten Einheiten widersprechen, weshalb derartige solche Abhängigkeiten durch Mocking aufgelöst werden. Das Framework

unittest erleichtert das Mocking, da hierfür vordefinierte Bestandteile wie MagicMock() oder @patch genutzt werden können, um Objekte zu simulieren beziehungsweise temporär zu ersetzen.

Konkrete Beispiele für Mock-Objekte, für die MagicMock() genutzt wird, sind der YouTube Client für den Zugriff auf die API sowie einzelne Bestandteile von Streamlit wie session_state (vgl. 2.3.1), Knöpfe oder Tabs. Der Decorator @patch wird beispielsweise im Kontext mit Neustarts oder der Initialisierung von Tabs in Streamlit genutzt, um den benötigten Zustand der Applikation im zu testenden Bereich nachzubilden.

Insgesamt können durch diese Design Pattern (Entwurfsmuster) enge in lose Kopplungen überführt werden (Gamma, et al., 1994, pp. 24-25). Interne Abläufe der APIs bleiben unbeachtet. Die Abdeckung durch Tests liegt bei 57%.

2.4.3 Antwortqualität von Gemini

Die Antworten von Gemini unterliegen nicht den Unit-Tests und die Qualität wird an dieser Stelle nur anhand von Stichproben bewertet.

Das Sprachmodell ist meist dazu in der Lage, relevantes Hintergrundwissen zu aktuellen Trendvideos bereitzustellen, sofern es das Vorhandensein und die Qualität der Transkripte zulässt. Ähnliches gilt für gesuchte Videos.

Schwächen zeigt das Modell gelegentlich bei Videovorschlägen basierend auf Nutzerinteressen. Hierbei kann es zu Halluzinationen der KI kommen, bei denen scheinbare Zusammenhängen erstellt werden, die jedoch im Kontext nicht schlüssig erscheinen.

Die Analyse bezüglich der Clickbait-Inhalte stimmt in einigen Ansätzen mit der menschlichen Wahrnehmung überein. Stellenweise erscheinen die Antworten, als wäre die Interpretation von Clickbait äußerst streng und es werde regelrecht nach kleinsten Details gesucht. Eine Hypothese dafür ist, dass die Analyse der übermittelten Metadaten ohne den gesamten Input von Bild- und Audiodateien mitunter zu unpräzise ist.

Maßnahmen zur Steigerung der Antwortqualität der KI umfassen die Anpassung der Prompts, der Implementierung einer Nutzerinteraktion in Form eines Chats mit dem Sprachmodell oder der Anbindung eines alternativen Modells. Gegebenenfalls wäre an dieser Stelle der Fokus auf ein spezialisierteres oder eigens trainiertes Modell zu richten.

2.4.4 Konzeptuelle Qualität

Schließlich soll der Qualitätsaspekt auf die Metaebene erweitert werden, um die Sinnhaftigkeit der Software zu beurteilen.

In ihrem Paper beleuchten Park & Kim (2025) den Zusammenhang zwischen dem Konsum von Zusammenfassungen auf YouTube und der Angst, etwas zu verpassen (Fear of Missing Out, FoMO) sowie der Angst vor besseren Optionen (Fear of Better Options, FoBO). Die Menge an Auswahlmöglichkeiten könne zum Zurückgreifen auf zusammengefasste Inhalte sowie Medienabhängigkeit führen. Angesichts der Möglichkeit, dass sich FoMO/FoBO und das Verlangen nach komprimierten Videoformaten gegenseitig amplifizieren könnten, sind dahingehende Funktionen der Software kritisch zu bewerten. Die Zweckmäßigkeit variiert je nach Anwendungsfall. Es ist zu differenzieren, ob eine Zusammenfassung beispielsweise aus der intrinsischen Motivation des Wissenserwerbs oder basierend auf sozialen Ängsten, unzureichend über etwas informiert zu sein, konsumiert wird. Die Vermutung, dass bisher verfügbare Inhalte zu Themen wie Politik, Wissenschaft und Bildung im Vergleich zu Unterhaltungskategorien tendentiell in längeren, detaillierteren Formaten vorliegen, deckt sich mit den Ergebnissen von Violot et al. (2024). Diese Analyse bezieht sich auf das Nutzerverhalten bei Videos mit regulärer Länge und dem Format „YouTube Shorts“ – Videos mit einer maximalen Dauer von 60 Sekunden (YouTube, 2025).

Insgesamt stellt das Programm eine Infrastruktur zur Verfügung, aus der bei adäquater Nutzung ein Mehrwert erzielt werden kann.

3 Schlussfolgerungen und Perspektiven der Anwendung

3.1.1 Retrospektive

Als finales Ergebnis dieses Projekts konnte eine Anwendung entwickelt werden, mit der sich Inhalte von YouTube nutzerspezifisch strukturieren lassen. Die Nutzung von Streamlit ermöglicht die Nutzerinteraktion mit einer GUI. Mithilfe der Schnittstelle zu Gemini wird das Programm um KI-basierte Funktionen ergänzt, mit denen sich Zusammenfassungen und personalisierte Empfehlungen generieren lassen.

Insgesamt konnten das eingangs formulierte Konzept und die Ziele umgesetzt werden, sodass der Umgang mit der gegenwärtigen Fülle an Videomaterial erleichtert wird.

Wichtige Entscheidungen im Projektverlauf umfassten Verfahren, um an nutzerspezifische Informationen wie Abonnements zu gelangen und diese durch ein Sprachmodell weiter zu verarbeiten. Ebenfalls ist die Entscheidung zugunsten von Streamlit maßgeblich an der Form des Endprodukts beteiligt. Während das Framework zwar eine Vielzahl von hilfreichen Möglichkeiten bietet, eine Benutzeroberfläche aufzubauen, so entstand zugleich die Herausforderung, Strukturen beziehungsweise die Anwendungsarchitektur so anzupassen, dass Ressourcen (insbesondere Zeit und Rechenleistung) effizient genutzt werden.

3.1.2 Ausblick

Wie durch den modulare Code bisherige Fortschritte integrieren werden konnten, gibt es auch einige Anhaltspunkte, bestehende Funktionalitäten zu verbessern und weitere einzubinden – beispielsweise in Form eines neuen Tabs. Dabei sollten zukünftige Neuerungen auf YouTube ebenfalls berücksichtigt werden.

Optimierungsmöglichkeiten bestehen in der Implementierung der Sprachverarbeitung. Hierzu wäre interessant, natives NLP anzubieten, also ein eigenes, spezialisiertes Modell anzutrainieren. Dieses könnte man explizit auf die Use Cases bezüglich der Zusammenfassungen und Clickbait-Analyse auslegen. Darüber hinaus weißt die Verfügbarkeit der Transkripte mitunter Lücken auf, welche durch zusätzliche Vorverarbeitungsschritte wie Audioverarbeitung oder Bilderkennung geschlossen werden könnten.

Zuletzt gilt es, den Fokus auf die Nutzer zu richten, Erkenntnisse aus anderen Perspektiven zu erlangen. Der Feedback-Tab hält diesbezüglich einen ersten konzeptuellen Ansatz fest.

Durch das Einbeziehen weiterer Tester sollten Optimierungsbedarf und Anreize für neue Funktionen die bisherige Recherchen und Entwicklungen nicht abdecken konnten, bestimmt werden. Dies entspricht der personalisierten Kuratierung – einem der Grundgedanken dieses Projekts.

4 Literaturverzeichnis

Boehm, B. W., 1986. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes,* 11(4), pp. 14-24.

Ferry, E., Raw, J. O. & Curran, K., 2015. Security evaluation of the OAuth 2.0 framework. *Information and Computer Security*, 23(1), pp. 73-101.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1 Hrsg. Upper Saddle River: Prentice Hall.

Google, 2025. *Gemini-Modelle.* [Online] Available at: https://ai.google.dev/gemini-api/docs/models?hl=de [Zugriff am 10 April 2025].

Google, 2025. *OAuth-Zustimmungsbildschirm konfigurieren und Bereiche auswählen.* [Online] Available at: https://developers.google.com/workspace/guides/configure-oauth-consent?hl=de [Zugriff am 10 April 2025].

Google, 2025. *YouTube Summary with ChatGPT & Claude.* [Online] Available at: https://chromewebstore.google.com/detail/youtube-summary-with-chat/nmmicjeknamkfloonkhhcjmomieiodli [Zugriff am 10 April 2025].

Kemm, R., 2022. The Linguistic and Typological Features of Clickbait in Youtube Video Titles. *Social Communication,* 8(1), pp. 66-80.

Leske, M., 2015. *Meet YouTube Red, the ultimate YouTube experience.* [Online] Available at: https://blog.youtube/news-and-events/red/ [Zugriff am 10 April 2025].

Park, I.-Y. & Kim, H.-M. K., 2025. The impact of FoMO and FoBO on YouTube summary video consumption: A social identity perspective. *Telecommunications Policy,* 49(2), pp. 1-13.

Puranik, G. M., Kamath, N., Nakshatra, A. & Dusane, G., 2024. YouTube Video Summarizer: A Web Based Application for Concise Visual and Textual Summary. *Journal of Analysis and Computation (JAC),* 18(1), pp. 37-45.

pytest, 2015. *Anatomy of a test.* [Online] Available at: https://docs.pytest.org/en/stable/explanation/anatomy.html [Zugriff am 10 April 2025].

Python Package Index, 2025. *yt-dlp.* [Online] Available at: https://pypi.org/project/yt-dlp/ [Zugriff am 10 April 2025].

Python, 2025. *multiprocessing — Process-based parallelism.* [Online] Available at: https://docs.python.org/3/library/multiprocessing.html#introduction [Zugriff am 10 April 2025].

Streamlit, 2025. *Caching overview.* [Online] Available at: https://docs.streamlit.io/develop/concepts/architecture/caching [Zugriff am 10 April 2025].

Streamlit, 2025. *Multithreading in Streamlit.* [Online] Available at: https://docs.streamlit.io/develop/concepts/design/multithreading [Zugriff am 10 April 2025].

Streamlit, 2025. *Session State.* [Online] Available at: https://docs.streamlit.io/develop/api-reference/caching-and-state/st.session_state [Zugriff am 10 April 2025].

Streamlit, 2025. *Understanding Streamlit's client-server architecture.* [Online] Available at: https://docs.streamlit.io/develop/concepts/architecture/architecture [Zugriff am 10 April 2025].

US Securities and Exchange Comission, 2006. *Google To Acquire YouTube for $1.65 Billion in Stock.* Mountain View, Kalifornien: s.n.

Violot, C., Elmas, T., Bilogrevic, I. & Humbert, M., 2024. *Shorts vs. Regular Videos on YouTube: A Comparative Analysis of User Engagement and Content Creation Trends.* Stuttgart, Association for Computing Machinery.

YouTube, 2009. *Smart Thumbnails on YouTube.* [Online] [Zugriff am 10 April 2025].

YouTube, 2018. *Introducing YouTube Premium.* [Online] Available at: https://blog.youtube/news-and-events/introducing-youtube-premium/ [Zugriff am 10 April 2025].

YouTube, 2024. *YouTube's AI power-up: How we got even more helpful this year.* [Online] Available at: https://blog.youtube/inside-youtube/2024-in-youtube-ai/ [Zugriff am 10 April 2025].

YouTube, 2025. *Kurzvideos auf YouTube erstellen und Einnahmen damit erzielen.* [Online] Available at: https://www.youtube.com/intl/de_ALL/creators/shorts/#:~:text=Was%20sind%20YouTube%20Shorts%3F,ausprobieren%20oder%20witzige%20Ideen%20umsetzen. [Zugriff am 10 April 2025].

YouTube, 2025. *YouTube-Partnerprogramm: Überblick und Voraussetzungen.* [Online] Available at: https://support.google.com/youtube/answer/72851?hl=de&co=GENIE.Platform%3DAndroid [Zugriff am 10 April 2025].