Bearbeitungsbeginn: 01.03.2024

Vorgelegt am: 30.08.2024

# Thesis

zur Erlangung des Grades

**Bachelor of Sience**

im Studiengang Medieninformatik

an der Fakultät Digitale Medien


***Felix Pönitzsch***

***Matrikelnummer: 265132***


## ViperGPT: Automatisierte Softwareentwicklung mit Large Language und Computer Vision Models


*Erstbetreuer:* *Prof. Dr. Uwe Hahne*

*Zweitbetreuer:* *Prof. Dr. Ruxandra Lasowski*

# Abstract

Die vorliegende Bachelorthesis befasst sich mit der automatisierten Softwareentwicklung unter Einsatz von Large Language Models (LLMs) und Computer Vision Modellen, wobei das ViperGPT-Framework im Mittelpunkt steht. ViperGPT kombiniert Code-Generierungsmodelle mit Vision-and-Language-Modellen, um durch die Generierung und Ausführung von Python-Code präzise Ergebnisse für komplexe visuelle Aufgaben zu liefern, ohne dass zusätzliches Training erforderlich ist.

Im Rahmen der Thesis werden zentrale Forschungsfragen behandelt: Wie funktioniert ein solches KI-Werkzeug? Werden Softwareentwickler durch die Inkorporation von KI obsolet? Ist eine Ausbildung in der grafischen Datenverarbeitung weiterhin notwendig?

Die vorläufigen Ergebnisse zeigen, dass größere Language Models (LLMs) tendenziell besseren Code generieren, während Zensur zu schlechteren Ergebnissen führt. Trotz der fortschrittlichen Fähigkeiten der Modelle ist keiner der generierten Codes vollkommen korrekt, was die Notwendigkeit von menschlichen Entwicklern und einer fundierten Ausbildung bestätigt.

Der Aufbau der Thesis gliedert sich in mehrere Abschnitte: Eine Einführung und historische Entwicklung der generativen KI, eine detaillierte Erläuterung der Fachbegriffe und genutzten Werkzeuge, sowie eine eingehende Betrachtung von ViperGPT, einschließlich Installation, Konfiguration, Fehlerkorrektur und Erweiterungsmöglichkeiten. Im Abschnitt Testing werden die Testfälle und deren Aufbau erläutert, gefolgt von einer umfassenden Auswertung der Testergebnisse. Abschließend erfolgt eine Bewertung, die Klärung der Forschungsfragen und eine Diskussion zukünftiger Entwicklungen.

Diese Arbeit trägt dazu bei, ein tieferes Verständnis der Potenziale und Grenzen von ViperGPT und ähnlichen Technologien zu entwickeln, und bietet Einblicke in die zukünftige Rolle von Softwareentwicklern.

Das komplette Projekt ist auf Github finden, bis zum Ende vom Oktober:
https://github.com/GMFelixfex/Thesis-Felix-Poenitzsch-ViperGPT

# Inhaltsverzeichnis

# 1. Einführung

Die vorliegende Arbeit befasst sich mit der Untersuchung der Fähigkeiten und Grenzen von großen Sprachmodellen (Large Language Models, LLMs) in Kombination mit Computer Vision (CV)-Modellen zur automatisierten Analyse und Interpretation von Bildern und Videos. Im Zentrum der Untersuchung steht das Werkzeug "ViperGPT", welches die Code-Generierung durch LLMs mit der Bildverarbeitung durch spezialisierte CV-Modelle verbindet.

Ziel dieser Arbeit ist es, die Effektivität und Zuverlässigkeit von ViperGPT bei der Bewältigung verschiedener Bildanalyseaufgaben zu evaluieren und dessen Ergebnisse mit den Leistungen menschlicher Teilnehmer zu vergleichen. Dabei wird ein besonderer Fokus auf die Frage gelegt, inwieweit moderne KI-Modelle bereits in der Lage sind, komplexe visuelle Daten zu verarbeiten und inwiefern sie den Anforderungen in realen Anwendungsszenarien gerecht werden können.

Im Laufe der Arbeit werden verschiedene Testfälle, die unterschiedliche Fähigkeiten der Modelle herausfordern, analysiert. Dies umfasst grundlegende Aufgaben wie die Zählung von Objekten und die Erkennung von Farben sowie komplexere Herausforderungen wie die Interpretation von Bildinhalten, das Verständnis von Beziehungen zwischen Objekten und die Nutzung externen Wissens zur Beantwortung von Fragen.

Die Ergebnisse dieser Untersuchungen sollen nicht nur Aufschluss über den aktuellen Stand der Technik im Bereich der Bildverarbeitung durch KI geben, sondern auch eine fundierte Einschätzung darüber ermöglichen, welche Weiterentwicklungen erforderlich sind, um die Leistungsfähigkeit solcher Systeme in der Praxis zu verbessern. Darüber hinaus wird die Frage beleuchtet, ob und in welchem Maße menschliche Expertise durch den Einsatz von KI ersetzt werden kann und welche Rolle spezialisierte Ausbildung in der grafischen Datenverarbeitung weiterhin spielt.

Durch die umfassende Analyse der Funktionsweise und der Leistungsfähigkeit von ViperGPT trägt diese Arbeit dazu bei, das Verständnis für die Interaktion von Sprachmodellen und Bildverarbeitungssystemen zu vertiefen und deren Potenzial sowie Grenzen in der Anwendung zu bewerten.

## 1.1. Forschungsfragen

Um die in dieser Arbeit verfolgten Ziele zu erreichen und ein tieferes Verständnis der Leistungsfähigkeit von ViperGPT und ähnlichen Systemen zu erlangen, wurden spezifische Forschungsfragen formuliert. Diese Fragen zielen darauf ab, die Funktionsweise des KI-Werkzeugs, seine Auswirkungen auf den Beruf des Softwareentwicklers sowie die Relevanz einer spezialisierten Ausbildung in der grafischen Datenverarbeitung zu beleuchten.

Im Folgenden werden die zentralen Forschungsfragen vorgestellt, die den Rahmen für die Untersuchung bilden und deren Beantwortung essenziell für das Verständnis der Potenziale und Herausforderungen moderner KI-Systeme ist.

### 1.1.1. Wie funktioniert ein solches KI-Werkzeug?

Diese Frage wurde aufgestellt, um die technische Funktionsweise von ViperGPT zu verstehen und zu erklären, wie verschiedene Komponenten – insbesondere Sprachmodelle (LLMs) und Computer Vision (CV)-Modelle – miteinander interagieren, um visuelle Daten zu verarbeiten und zu interpretieren. Das Verständnis dieser Funktionsweise ist entscheidend, um die Leistungsfähigkeit des Systems beurteilen zu können und seine Stärken sowie Schwächen zu identifizieren. Nur durch eine detaillierte Analyse der Arbeitsweise des Werkzeugs können fundierte Aussagen über seine Eignung für unterschiedliche Anwendungsbereiche getroffen werden.

### 1.1.2. Werden Softwareentwickler durch die Inkorporation von KI obsolet?

Diese Frage richtet den Fokus auf die potenziellen Auswirkungen der fortschreitenden Automatisierung durch KI auf den Arbeitsmarkt, insbesondere auf den Beruf des Softwareentwicklers. Mit der zunehmenden Fähigkeit von KIs, Code zu generieren und komplexe Aufgaben zu lösen, stellt sich die Frage, ob menschliche Softwareentwickler in Zukunft durch solche Systeme ersetzt werden könnten. Die Beantwortung dieser Frage ist nicht nur aus technischer, sondern auch aus gesellschaftlicher Perspektive von Bedeutung, da sie Aufschluss darüber gibt, welche neuen Anforderungen und Qualifikationen in der Softwareentwicklung erforderlich sein könnten.

### 1.1.3. Ist eine Ausbildung in der grafischen Datenverarbeitung weiterhin notwendig?

Die dritte Forschungsfrage wurde formuliert, um zu untersuchen, ob spezialisierte Kenntnisse in der grafischen Datenverarbeitung angesichts der Fortschritte in der KI-gestützten Bildverarbeitung weiterhin relevant sind. Da moderne CV-Modelle zunehmend in der Lage sind, komplexe visuelle Aufgaben zu bewältigen, stellt sich die Frage, ob eine tiefgehende Ausbildung in diesem Bereich noch erforderlich ist oder ob die Technologie bereits so weit entwickelt ist, dass sie diesen Bedarf überflüssig macht. Die Antwort auf diese Frage ist entscheidend für die Gestaltung zukünftiger Bildungsprogramme und für die Vorbereitung der nächsten Generation von Fachkräften in der Informatik und Grafik.

## 1.2. Zeitstrahl der Entwicklung von Large Language Models

Dieser Text bietet eine Übersicht über die Geschichte der Künstlichen Intelligenz (KI) und konzentriert sich insbesondere auf die Entwicklung großer Sprachmodelle. Die Darstellung orientiert sich an der "History of Artificial Intelligence" von S. Russell und P. Norvig sowie an weiteren Quellen. Eine Kurzfassung der Liste von Turing-Preisträgern könnten die bedeutendsten Meilensteine in der KI Geschichte hervorheben, wie etwa Marvin Minsky (1969), John McCarthy (1971), Ed Feigenbaum und Raj Reddy (1994), Judea Pearl (2011) sowie Yoshua Bengio, Geoffrey Hinton und Yann LeCun (2019). [1] Dennoch beginnt die ausführliche Darstellung mit der Entstehung der Künstlichen Intelligenz.

Die Entwicklung von KI nahm 1943 ihren Anfang, als Warren McCulloch und Walter Pitts, inspiriert von den Arbeiten von Nicolas Rashevsky (1936, 1938), ein Modell künstlicher Neuronen vorschlugen. Sie zeigten, dass jede berechenbare Funktion durch ein Netzwerk verbundener Neuronen berechnet werden kann. Zudem legten McCulloch und Pitts nahe, dass solche Netzwerke lernen könnten. [1]

1947 hielt Alan Turing Vorträge über Künstliche Intelligenz vor der London Mathematical Society. Zwei Jahre später, 1949, entwickelte Donald Hebb eine einfache Regel zur Anpassung der Verbindungsstärken zwischen Neuronen, die als Hebbsches Lernen bekannt wurde. [1]

1950 bauten Marvin Minsky und Dean Edmonds den ersten neuronalen Netzwerkcomputer namens SNARC. Im selben Jahr veröffentlichte Alan Turing seine bahnbrechende Arbeit „Computing Machinery and Intelligence“, in der er den Turing-Test, maschinelles Lernen, genetische Algorithmen und Verstärkungslernen einführte. Er argumentierte, dass es einfacher sei, menschliche Intelligenz durch die Entwicklung von Lernalgorithmen zu erreichen, anstatt sie direkt zu programmieren. [1]

1956 fand die Dartmouth College Sommerkonferenz zur Künstlichen Intelligenz statt, bei der der Begriff "Künstliche Intelligenz" geprägt wurde. In diesem Jahr präsentierten Newell und Simon das "Logic Theorist" (LT), ein mathematisches Theorem-Beweissystem, das 38 von 52 Theoremen von Russell und Whitehead bewies. Gleichzeitig entwickelte Arthur Samuel ein Dame-Programm unter Verwendung von Methoden des Verstärkungslernens, das als Vorläufer vieler Spielesysteme, wie TD-Gammon (Backgammon) und AlphaGo (Go), gilt. [1]

1958 schuf John McCarthy die Programmiersprache Lisp und schlug in seiner Arbeit „Programs with Common Sense“ ein fortschrittliches KI-System namens „The Advice Taker“ vor. Frank Rosenblatt

entwickelte im selben Jahr tiefere mehrschichtige Perzeptronen (MLP), die jedoch noch keine tiefen neuronalen Netze darstellten, sondern mehrschichtige "Feedforward Neural Networks" (FNN). [2]

1959 entwickelten Newell, Shaw und Simon den "General Problem Solver" (GPS) an der Carnegie Mellon University (CMU), ein Programm, das menschliche Problemlösungsprotokolle imitieren sollte. McCarthy und Minsky gründeten das MIT AI Lab. [1]

1962 veröffentlichte Rosenblatt den Konvergenzsatz des Perzeptrons, der Hebbs Lernmethode verbesserte. [2] 1963 gründete McCarthy ein KI-Labor an der Stanford University, wo er zusammen mit seinen Studenten das Analogieprogramm von Tom Evans und James Slagles SAINT-Programm entwickelte. Leonard Uhr und Charles Vossler veröffentlichten im selben Jahr ein Mustererkennungsprogramm, das in der Lage war, adaptiv Merkmale zu erwerben und anzupassen, wodurch die Einschränkungen der von Rosenblatt entwickelten Perzeptronen überwunden wurden. [3]

1965 führten Alexey Ivakhnenko und Valentin Lapa die ersten funktionierenden Lernalgorithmen für tiefe MLPs ein, was als der Beginn des Deep Learning gilt. [2] 1966 entwarf Joseph Weizenbaum von der MIT das ELIZA-Programm, das jedoch die Grenzen der damaligen Computerleistung offenbarte, da die Verarbeitung von natürlicher Sprache ins Stocken geriet. Dies führte zu einem drastischen Rückgang der Finanzierung für neuronale Netzwerke,[1] was teilweise in einem Bericht des Automatic Language Processing Advisory Committee (ALPAC) dokumentiert wurde.[4]

Von hier an bis in die 1980er Jahre lag der Schwerpunkt auf der Entwicklung von Algorithmen und domänenspezifischen Systemen.[1] 1969 veröffentlichten Marvin Minsky und Seymour Papert das Buch „Perceptrons“, das die Grenzen der zweischichtigen Feedforward-Struktur aufzeigte.[5] Im selben Jahr wurde DENDRAL von Feigenbaum entwickelt, das erste erfolgreiche wissensintensive System, dessen Expertise auf einer großen Anzahl spezieller Regeln basierte und als eines der ersten Expertensysteme gilt.[1]

1970 veröffentlichte Seppo Linnainmaa eine Methode, die später als "Backpropagation" bekannt wurde und bis heute intensiv in künstlichen neuronalen Netzwerken verwendet wird. 1972 wurde das MYCIN-System an der Stanford University entwickelt, das mit 450 Regeln in der Lage war, auf dem Niveau einiger Experten zu agieren.[1] 1973 führte der Lighthill-Bericht zu einer negativen Einschätzung der KI-Forschung, was dazu führte, dass Großbritannien den Großteil seiner Finanzierung zurückzog.[6]

1976 schlugen Newell und Simon die „Physical Symbol System Hypothesis“ vor, und 1978 erhielt Simon den Nobelpreis für Wirtschaftswissenschaften für seine Arbeiten über „satisficing“, eine Theorie, die zeigt, dass Entscheidungen, die „gut genug“ sind, eine bessere Beschreibung tatsächlichen menschlichen Verhaltens liefern als optimierende Entscheidungen.[1]

In den späten 1970er Jahren entwickelten Schank und Abelson sowie Wilensky und Riesbeck eine Reihe von Programmen, die die Aufgabe hatten, natürliche Sprache zu verstehen.[1] 1979 wurde der Stanford Cart zum ersten computergesteuerten, autonomen Fahrzeug. [7]

Die 1980er Jahre markierten eine Wiederbelebung der Forschung zu neuronalen Netzen. 1981 kündigte die japanische Regierung das „Fifth Generation“-Projekt an, ein zehnjähriges Programm zur Entwicklung massiv paralleler, intelligenter Computer, die Prolog ausführen sollten. In den 1980er Jahren boomte die KI-Industrie, von einigen Millionen Dollar im Jahr 1980 auf Milliarden Dollar im Jahr 1988, und es entstanden Hunderte von Unternehmen, die Expertensysteme, Visionsysteme, Roboter sowie spezialisierte Software und Hardware entwickelten. [1]

1984 prägten Roger Schank und Marvin Minsky den Begriff „AI Winter“, der eine Phase zwischen 1974 und 1980 sowie zwischen 1987 und 2000 beschreibt, in der das Interesse und die Finanzierung für KI stark zurückgingen. [1]

1986 fand die Rückpropagation breite Anwendung auf viele Lernprobleme in der Informatik und Psychologie, und die Ergebnisse wurden in der Sammlung „Parallel Distributed Processing“ veröffentlicht.[8] Diese sogenannten konnektionistischen Modelle wurden von einigen als direkte Konkurrenz zu den symbolischen Modellen angesehen, die von Newell und Simon propagiert wurden, sowie zum logizistischen Ansatz von McCarthy und anderen. Die symbolische Natur wurde von Geoff Hinton, einer führenden Figur der Wiederbelebung der neuronalen Netze, als „luminiferous aether of AI“ bezeichnet, ein Konzept, das modernen Standards nicht mehr entspricht. Die konnektionistischen Modelle bilden interne Konzepte in einer flüssigeren und ungenaueren Weise, die besser zur Komplexität der realen Welt passt. [1]

1988 führte Judea Pearls Werk „Probabilistic Reasoning in Intelligent Systems“ zu einer neuen Akzeptanz von Wahrscheinlichkeitstheorie und Entscheidungstheorie in der KI. Im selben Jahr verband Rich Sutton das Verstärkungslernen mit der Theorie der Markow-Entscheidungsprozesse (MDPs), die im Bereich Operations Research entwickelt wurde. Diese Entwicklungen führten zu einer Flut von Arbeiten, die die KI-Planungsforschung mit MDPs verbanden, und das Feld des Verstärkungslernens fand Anwendungen in Robotik und Prozesssteuerung sowie tiefere theoretische Grundlagen. [1]

Diese Wertschätzung für Daten, statistische Modellierung, Optimierung und maschinelles Lernen führte zur allmählichen Wiedervereinigung der Teilbereiche der KI (wie Computer Vision, Robotik, Spracherkennung, Multiagentensysteme und Verarbeitung natürlicher Sprache) zurück zum Kern der Künstlichen Intelligenz. [1]

1997 wurde die Deep-Learning-Methode Long Short-Term Memory (LSTM) von Sepp Hochreiter und Jürgen Schmidhuber in „Neural Computation“ veröffentlicht. [9] LSTM wurde zum meistzitierten neuronalen Netzwerk des 20. Jahrhunderts. [2]

Das Phänomen Big Data bezeichnet Datensätze, die Billionen von Wörtern, Milliarden von Bildern und Milliarden von Stunden Sprache und Video umfassen, eine Entwicklung, die durch das World Wide Web ermöglicht wurde. [1]

2001 argumentierten Banko und Brill, dass die Leistungsverbesserung, die durch die Vergrößerung des Datensatzes um zwei oder drei Größenordnungen erreicht wird, jede Verbesserung übersteigt, die durch Optimierung des Algorithmus erzielt werden kann, also: Mehr Daten = bessere KI. [1]

2007 entwickelten Hays und Efros eine clevere Methode, indem sie Pixel aus ähnlichen Bildern mischten. Sie stellten fest, dass die Technik mit einer Datenbank von nur Tausenden von Bildern schlecht funktionierte, aber eine Qualitätsschwelle überschritt, als Millionen von Bildern verwendet wurden. [1]

2009 wurde ImageNet veröffentlicht, der bis dato größte Datensatz von Bildern und ihren Klassifikationen. Die Verfügbarkeit von Big Data und die Hinwendung zum maschinellen Lernen halfen der KI, ihre kommerzielle Attraktivität wiederzugewinnen. [10]

2011 triumphierte IBMs Watson-System über menschliche Champions in der Quizshow "Jeopardy!". [1]

2012 wurde AlexNet, ein von Alex Krizhevsky entwickeltes Deep-Learning-Modell, veröffentlicht und gewann den ImageNet Large Scale Visual Recognition Challenge. [10]

Seitdem haben Deep-Learning-Systeme in einigen visuellen Aufgaben die menschliche Leistung übertroffen (und hinken in anderen Aufgaben hinterher). Ähnliche Fortschritte wurden in den Bereichen Spracherkennung, maschinelle Übersetzung, medizinische Diagnostik und Spielentwicklung erzielt. Diese bemerkenswerten Erfolge haben zu einem Wiederaufleben des Interesses an KI bei

Studierenden, Unternehmen, Investoren, Regierungen, Medien und der allgemeinen Öffentlichkeit geführt. Es scheint, dass jede Woche eine neue KI-Anwendung vorgestellt wird, die menschliche Leistungen erreicht oder übertrifft, oft begleitet von Spekulationen über beschleunigten Erfolg oder einen neuen KI-Winter. [1]

2015 unterzeichneten Stephen Hawking, Elon Musk und Dutzende von KI-Experten einen offenen Brief zur Künstlichen Intelligenz, in dem sie zu Forschungen über die gesellschaftlichen Auswirkungen von KI aufriefen. [11]

2017 entwickelte Google die TRANSFORMER-Architektur für Deep Learning, die in ihrem Papier „Attention Is All You Need“ vorgestellt wurde. Dies markierte den Beginn der Nutzung der Transformer-Architektur für viele Systeme weltweit, wie die GPT-LLMs oder BERT-LLMs. [12]

2018 wurden GPT-1 und BERT veröffentlicht, 2019 folgte GPT-2, 2020 schließlich GPT-3. Zwischen 2021 und Oktober 2022 wurden Modelle wie LaMDA, XLNet, Chinchilla, CodeGen, InCoder, MGPT, PaLM, OPT-IML und Minerva veröffentlicht. [13] Im November 2022 wurde ChatGPT veröffentlicht, das einen Rekord für das schnellste Nutzerwachstum aufstellte, und im Dezember 2022 erschien GPT-3.5. [14]

2023 wurden im Januar WebGPT, im Februar Google Bard und LLaMA sowie im März GPT-4 veröffentlicht. [13] In dieser Zeit wurde auch ViperGPT, das Programm, auf dem diese Arbeit basiert, vorgestellt. In den folgenden Monaten wurden viele verschiedene offene Sprachmodelle veröffentlicht, von denen einige für Tests in dieser Arbeit verwendet wurden. [15]

# 2. Grundlagen

Die zuvor dargestellte historische Entwicklung der Künstlichen Intelligenz (KI) zeigt eindrucksvoll, wie vielfältig und komplex die Fortschritte in diesem Feld sind. Um diese Entwicklungen umfassend zu verstehen, ist es entscheidend, grundlegende Begriffe und Konzepte zu klären, da das fundierte Verständnis dieser Grundlagen eine unerlässliche Basis für die weitergehende Auseinandersetzung mit der Materie darstellt. Es gibt eine Fülle an Grundwissen, das geklärt werden muss, um die technologische Entwicklung, insbesondere die modernen Ansätze wie Deep Learning und die Transformer-Architektur, vollständig zu erfassen. Im Folgenden werden daher die wesentlichen Fachbegriffe und Konzepte, die für das Verständnis von Large Language Models (LLMs) und anderen KI-Technologien zentral sind, erläutert.

## 2.1. Fachbegrifserklärung

AI – Artificial Intelligence

Künstliche Intelligenz (AI) ist ein umgangssprachlicher Begriff, der verschiedene Systeme, Modelle und Konzepte beschreibt. In diesem Zusammenhang wird AI verwendet, um eine Maschine oder ein Modell zu bezeichnen, das in der Lage ist, einen Teil der menschlichen Intelligenz zu repräsentieren. Da dies eine sehr allgemeine Definition ist, werden spezifischere Begriffe wie ML (Machine Learning), ANN/CNN (Artificial Neural Networks/Convolutional Neural Networks), NLP (Natural Language Processing) und CV (Computer Vision) für die einzelnen Konzepte innerhalb der AI verwendet.

CV – Computer Vision

Computer Vision befasst sich mit einem breiten Spektrum an Anwendungen, die von der Erfassung roher Daten bis zur Extraktion von Bildmustern und der Interpretation von Informationen reichen. Es kombiniert Konzepte, Techniken und Ideen aus der digitalen Bildverarbeitung, Mustererkennung, künstlichen Intelligenz und Computergrafik. Die meisten Aufgaben in der Computer Vision beziehen sich auf die Informationsgewinnung aus Eingabeszenen (digitale Bilder) und die Merkmalsextraktion.

Die Methoden zur Problemlösung in der Computer Vision hängen vom Anwendungsbereich und der Natur der analysierten Daten ab.

Computer Vision kombiniert Bildverarbeitung und Mustererkennung, wobei das Ziel die Bildverarbeitung ist. Die Entwicklung dieses Fachgebiets orientiert sich an der menschlichen Sehfähigkeit zur Informationsaufnahme. Im Gegensatz zur Computergrafik konzentriert sich Computer Vision auf die Extraktion von Informationen aus Bildern. Die Weiterentwicklung dieses Gebiets hängt von der technologischen Entwicklung der Computersysteme ab, sei es im Hinblick auf die Verbesserung der Bildqualität oder der Bilderkennung. Es gibt Überschneidungen mit der Bildverarbeitung in den grundlegenden Techniken, und einige Autoren verwenden beide Begriffe synonym. [16]

ML – Machine Learning

Maschinelles Lernen zielt darauf ab, „Wissen“ aus „Erfahrung“ zu generieren, indem Lernalgorithmen aus Beispielen ein komplexes Modell entwickeln. Dieses Modell und die damit einhergehende automatisch erworbene Wissensrepräsentation können anschließend auf neue, potenziell unbekannte Daten derselben Art angewendet werden. [17]

NLP – Natural Language Processing

Natural Language Processing (NLP) umfasst eine Sammlung von rechnergestützten Techniken zur automatischen Analyse und Repräsentation menschlicher Sprachen, die von theoretischen Überlegungen motiviert sind. Um jedoch Texte auf einem Niveau zu analysieren, das dem menschlichen Verständnis entspricht, ist ein tiefgehendes Verständnis der natürlichen Sprache durch Maschinen erforderlich. Dies stellt eine große Herausforderung dar und ist Gegenstand umfangreicher Forschung. NLP ist Teilgebiet der Computerlinguistik, die sich mit der Entwicklung von Computersystemen zur Verarbeitung und Generierung natürlicher Sprache beschäftigt. Die in dieser Arbeit verwendeten Modelle sind das Ergebnis dieser NLP-Forschung. [18]

ANN – Artificial Neural Networks

Künstliche neuronale Netze (ANN) sind Netzwerke aus künstlichen Neuronen, die von den biologischen Neuronen im Gehirn inspiriert wurden. Ein KNN besteht aus künstlichen Neuronen, die miteinander verbunden und in der Regel in Schichten organisiert sind. ANNs werden im maschinellen Lernen eingesetzt, um Probleme zu lösen, die zu komplex sind, um sie durch Regeln zu beschreiben, aber für die viele Daten als Beispiele für die gewünschte Lösung vorliegen. [17] Dazu zählen Convolutional Neural Networks (CNN), Feedforward Neural Networks (FNN), Recurrent Neural Networks (RNN) und Deep Neural Networks (DNN). Für diese Arbeit sind Deep Neural Networks von besonderer Bedeutung, da alle verwendeten Modelle DNNs sind.

Deep Learning

Deep Learning ermöglicht es, Repräsentationen von Daten auf mehreren Abstraktionsebenen durch Modelle mit mehreren Verarbeitungsschichten zu lernen. Es entdeckt komplexe Strukturen in großen Datensätzen mithilfe des Backpropagation-Algorithmus, der angibt, wie eine Maschine ihre internen Parameter anpassen sollte, um die Repräsentationen in jeder Schicht aus der vorhergehenden Schicht zu berechnen. Tiefe neuronale Netze (insbesondere tiefe Convolutional Netze) haben Durchbrüche in der Verarbeitung von Bildern, Videos, Sprache und Audio erzielt, während rekurrente Netze sich bei der Verarbeitung von sequentiellen Daten wie Text und Sprache bewährt haben. [19]

Backpropagation

Backpropagation ist ein Lernverfahren für künstliche neuronale Netze, bei dem das Netzwerk wiederholt die Gewichte der Verbindungen anpasst, um eine Maßgröße für den Unterschied zwischen dem tatsächlichen und dem gewünschten Ausgang des Netzes zu minimieren. Durch die Gewichtsanpassungen repräsentieren die internen, „versteckten" Einheiten, die weder Teil des Eingangs noch des Ausgangs sind, wichtige Merkmale der Aufgabe, und die Regelmäßigkeiten der Aufgabe werden durch die Interaktionen dieser Einheiten erfasst. [20]

Transformer

Der Transformer ist eine Deep-Learning-Modellarchitektur, die auf Rekurrenz verzichtet und stattdessen vollständig auf einem Aufmerksamkeitsmechanismus beruht, um globale Abhängigkeiten zwischen Eingabe und Ausgabe herzustellen. Diese Architektur wurde von Google entwickelt und ist derzeit führend bei der Erstellung großer Sprachmodelle. Innerhalb dieser Architektur werden Wörter und Texte in numerische Repräsentationen umgewandelt, sogenannte Tokens, die dann in hochdimensionale Vektoren konvertiert werden. Die Transformer-Architektur ist effizienter als die zuvor verwendeten Long-Short-Term-Memory-Architekturen (LSTM). Die aktuellen GPT-Modelle (Generative Pretrained Transformer) verwenden die Transformer-Architektur und sind daher für diese Arbeit von entscheidender Bedeutung. [12]

LLM – Large Language Model

Ein Large Language Model (LLM) ist ein rechnergestütztes Modell, das in der Lage ist, Sprachgenerierung oder andere Aufgaben der natürlichen Sprachverarbeitung zu übernehmen. Diese Modelle erlernen ihre Fähigkeiten durch das Lernen statistischer Zusammenhänge aus großen Textmengen während eines selbstüberwachten und halbüberwachten Trainingsprozesses. LLMs werden im Tool ViperGPT verwendet, um Code zu generieren und Folgefragen zu beantworten. [21]

## 2.2. Konzeptgrundlagen

Um das System hinter ViperGPT verstehen zu können, ist es notwendig, einige grundlegende Konzepte zu erläutern, die die fundamentalen Bestandteile des Prozesses ermöglichen. Dabei stehen die Funktionalität von Large Language Models (LLMs) sowie das Verständnis der Visual Question Answering (VQA) als Kern des Werkzeugs im Mittelpunkt.

### 2.2.1. Visual Question Answering (VQA)

Visual Question Answering (VQA) ist ein System, das ein Bild sowie eine freiformulierte, offene, natürlichsprachliche Frage zu diesem Bild entgegennimmt und als Ausgabe eine natürlichsprachliche Antwort generiert. Dieses Verfahren wird unter anderem für sehbehinderte Nutzer oder für Geheimdienstanalysten genutzt, um visuelle Informationen zu extrahieren.

Die Beantwortung offener Fragen erfordert eine Vielzahl von KI-Fähigkeiten, darunter:

- Detaillierte Erkennung
- Objekterkennung
- Aktivitätserkennung
- Wissensbasierte Schlussfolgerung
- Alltagswissen und logisches Denken

Obwohl VQA häufig kurze Antworten oder Ja/Nein-Antworten liefert, erfordert die Aufgabe dennoch ein komplexes Set an Fähigkeiten, was den Prozess kompliziert und nicht trivial macht.

Die Grundlage für VQA bildet das textbasierte Frage-Antwort-System, ein gut untersuchtes Problem in den Bereichen NLP und Textverarbeitung. Dabei werden textbasierte Fragen von NLP-Modellen

entweder über Multiple-Choice-Formate oder freie Formate beantwortet. Ein weiterer verwandter Bereich ist das Tagging von Bildern, die Bildunterschriftenerstellung und die Video-Untertitelung, bei denen Wörter oder Sätze generiert werden, um visuelle Inhalte zu beschreiben. [22]

Ein zentrales Problem von VQA, das auch für die Funktionalität dieses Werkzeugs von Bedeutung ist, besteht darin, dass die Sprachverarbeitung den Eindruck einer starken Leistung erwecken kann, selbst wenn die Modelle den visuellen Inhalt nicht vollständig verstehen. [23] Dieses Problem spielt eine Rolle bei der Überbrückung der Lücke zwischen Code und Bild, die in diesem Projekt eine zentrale Herausforderung darstellt und später im Zusammenhang mit der Funktionsweise von ViperGPT näher erläutert wird.

Das Konzept von VQA ist umfangreich genug, um eine eigene Dissertation zu rechtfertigen, weshalb in dieser Arbeit nur die notwendigsten Details behandelt werden.

### 2.2.2. Visuelles Schlussfolgern (Visual Reasoning)

Visuelles Schlussfolgern ist ein Teilbereich von VQA und umfasst Fähigkeiten wie Zählen, Vergleichen, logisches Denken, Informationsspeicherung im Gedächtnis und Kompositionsbewusstsein. Das ursprüngliche CLEVR-Datensatz (Compositional Language and Elementary Visual Reasoning diagnostics) besteht aus Fragen und Bildern, die komplexes Schlussfolgern erfordern, um beantwortet zu werden. Dabei werden Alltagswissen und wissensbasierte Schlussfolgerungen bewusst ausgeschlossen, um externe Informationsquellen nicht zu nutzen und die Fragen ausschließlich auf Basis der visuellen Inhalte zu beantworten.

Visuelles Schlussfolgern trägt auch zur Überbrückung der zuvor genannten Lücke zwischen Sprachverarbeitung und visuellen Inhalten bei. [24]

### 2.2.3. Statistische Modelle

Alle in dieser Arbeit verwendeten Modelle basieren auf demselben System: der statistischen Analyse mit einem Training zur Schätzung des nächsten Wortes, Satzes oder Absatzes. Dabei wird ein Zufallsfaktor (Temperatur) eingeführt, um den Output zu variieren, Voreingenommenheiten zu minimieren und Wiederholungen zu vermeiden. Wie bereits bei LLMs erläutert, funktioniert dies über Tokens. Das Modell versucht, basierend auf dem gegebenen Kontext, der aus etwa 3900 Tokens besteht, den Eingabetext fortzuführen. Dies ist die einzige Aufgabe des Modells. Ein oder mehrere Tokens werden je nach Wahrscheinlichkeit an den Kontext angefügt, und dieser Prozess wird so lange wiederholt, bis ein Stopp-Signal auftritt oder das Modell errechnet, dass der Output abgeschlossen ist. Es ist daher wichtig zu betonen, dass ein LLM, wie es hier verwendet wird, nicht denken kann. [25] und [26]

## 2.3. Genutzte Werkzeuge

Um ViperGPT erfolgreich nutzen zu können, ist es nicht nur notwendig, den theoretischen Hintergrund zu verstehen, sondern auch die richtigen Werkzeuge und Programme einzusetzen. Für diese Arbeit wurden mehrere spezifische Werkzeuge verwendet.

LM-Studio

LM-Studio ist ein Programm, das es ermöglicht, Large Language Models (LLMs) lokal auf dem eigenen Computer auszuführen, ohne auf externe Server zugreifen zu müssen. Die Modelle können über eine API angesteuert werden, zum Beispiel durch ein Python-Programm. Die Modelle selbst können in LM-Studio aus den Huggingface-Repositories heruntergeladen werden. Die Benutzeroberfläche erlaubt es, Einstellungen der Modelle wie die Kontextgröße, die GPU-Auslastung, System-Prompts und weitere Parameter zu ändern. [27]

Visual Studio Code (VS Code)

VS Code ist ein kostenloser Editor bzw. eine Entwicklungsumgebung (IDE), die eine Vielzahl an Entwicklerwerkzeugen und Erweiterungen bietet. Entwickelt von Microsoft, ist VS Code bei Entwicklern sehr beliebt, da es eine breite Unterstützung für verschiedene Programmiersprachen bietet und eine benutzerfreundliche Oberfläche hat. Der Editor ist leichtgewichtig, aber dennoch leistungsstark, mit Funktionen wie IntelliSense (intelligente Codevervollständigung), Debugging, Git-Integration und einem integrierten Terminal. Für diese Arbeit wurde die Anaconda-Integration in VS Code genutzt, um ViperGPT im Hintergrund auszuführen. Zudem wurde VS Code für alle Programmierarbeiten bei der Erweiterung und Verbesserung des Projekts verwendet. [28]

Python 3.10

Die Programmiersprache, die das gesamte Projekt antreibt, ist Python, speziell die Version 3.10. Python wird häufig in wissenschaftlichen und industriellen Projekten eingesetzt, da es eine hohe Lesbarkeit sowie eine breite Unterstützung durch Bibliotheken und Frameworks bietet. Es findet oft Anwendung in den Bereichen Computer Vision, Machine Learning und bei LLMs. Auch bei ViperGPT wird alles in Python programmiert und für die Zwecke dieser Arbeit erweitert. [29]

Anaconda

Anaconda ist eine Distribution, die hauptsächlich für Datenwissenschaftler, Entwickler und Forscher entwickelt wurde, um die Arbeit mit Python und R zu erleichtern. Sie bietet eine Sammlung von mehr als 1.500 Open-Source-Paketen, die für Datenanalyse, maschinelles Lernen, wissenschaftliches Rechnen und vieles mehr verwendet werden können. Die Anaconda-Distribution enthält eine integrierte Entwicklungsumgebung (IDE) namens Spyder sowie das Jupyter Notebook, das sich besonders für interaktive Datenanalysen eignet. Darüber hinaus umfasst Anaconda das Paket- und Umgebungsverwaltungstool Conda, mit dem Benutzer leicht neue virtuelle Umgebungen erstellen und verwalten können, um Pakete und Abhängigkeiten zu isolieren. Anaconda wurde für die Installation von ViperGPT genutzt, da es viele der Standardpakete enthält und eine einfache Transfermöglichkeit bietet. [30]

# 3. ViperGPT

Kurzbeschreibung: Das Werkzeug Visual Inference via Python Execution for Reasoning with GPT (ViperGPT) wurde entwickelt, um Fragen zu Bildern und Videos zu beantworten.

Ausführliche Beschreibung: ViperGPT ist ein Framework, das Code-Generierungsmodelle nutzt, um Vision-and-Language-Modelle in Unterroutinen zu integrieren, die Ergebnisse für beliebige Anfragen erzeugen. ViperGPT verwendet eine bereitgestellte API, um auf die verfügbaren Module zuzugreifen, und komponiert diese, indem es Python-Code generiert, der anschließend ausgeführt wird. Dieser einfache Ansatz erfordert kein weiteres Training und erzielt dennoch state-of-the-art Ergebnisse bei verschiedenen komplexen visuellen Aufgaben. [31]

## 3.1. Funktionserklärung von ViperGPT

1. Das Gesamtkonzept von ViperGPT lässt sich in einem Ablaufdiagramm darstellen:
2. Eingabe der Anfrage (Query): Der Benutzer gibt eine Frage zu einem Bild oder Video ein.
3. Übermittlung der Anfrage: Die Anfrage wird über die API an ein Large Language Model (LLM) gesendet.
4. Code-Generierung: Das LLM generiert den entsprechenden Python-Code und sendet diesen zurück.
5. Laden der visuellen Daten: Das Bild oder die Bilder werden geladen.

6. Ausführung des Codes: Der generierte Code wird mit den geladenen Bildern als Input ausgeführt.
7. Analyse und Verarbeitung:
    a. Bildanalyse: Das Bild wird analysiert.
    b. Rückfragen: Es werden gegebenenfalls Rückfragen gestellt, um weitere Informationen zu erhalten.
8. Antwortformatierung: Die Antworten werden in ein geeignetes Format gebracht.
9. Ausgabe des Ergebnisses: Das finale Ergebnis wird dem Benutzer präsentiert.
10. Optionale Visualisierung: Der gesamte Prozess kann visuell dargestellt werden.

Der modifizierte Ablauf wird später in Abschnitt 4.1 bei der Beschreibung der Tests erläutert, da dort spezifische Änderungen genutzt werden.

Visualisierung

Die Visualisierung war ein praktischer Teil des Projekts, bei dem jeder Schritt der Codeausführung einzeln durchlaufen wurde. Dabei wurden beispielsweise Bildausschnitte (Image Patches) angezeigt, die einzelnen Ausgaben der Funktionen dargestellt sowie das Endergebnis in ansprechender Form präsentiert. Leider war die Visualisierung bei der Testinstallation für diese Thesis nicht möglich, da ein

Fehler aufgrund eines Paketinkompatibilitätsproblems und der Aktualität des Projekts nicht behoben werden konnte. Visualisierte Ergebnisse können jedoch auf der Projektwebsite eingesehen werden. [32]

Batch-System

Im Originalprojekt gab es ein Batch-System, das den Ablauf für die Analyse von Videos und zahlreichen Bildern unterstützen sollte. Dieses System wurde im Paper vorgestellt und ermöglichte die Analyse von Videos Bild für Bild. Bei meiner Installation war das Batch-System jedoch veraltet, was zu Problemen bei der Ansteuerung verschiedener Modelle sowie bei der Aufbereitung der Anfragen führte.

Modularität

ViperGPT ist äußerst modular programmiert, sodass die interne Steuerung der Modelle vollständig flexibel gestaltet ist. Jedes in der Konfigurationsdatei (Config) eingestellte Modell kann im Code über die forward()-Funktion genutzt werden. Diese Funktion leitet die Parameter an das jeweilige Modell weiter und verarbeitet die Ergebnisse. Diese Modularität ermöglicht es, das Tool einfach zu erweitern und neue Modelle schnell zu integrieren. Weitere Details zur Erweiterung finden sich in Abschnitt 3.4.

Integrierte Modelle

Durch die hohe Modularität von ViperGPT ist es möglich, viele verschiedene und spezialisierte Modelle zu nutzen, die jeweils bestimmte Aufgaben erfüllen. Jedes in der Konfigurationsdatei aufgelistete Modell kann während des Ausführungsschritts vom Tool genutzt werden, mit Ausnahme der Code-Generatoren. Im Folgenden werden die verschiedenen Modelle und ihre Leistungen innerhalb des Tools beschrieben:

- Mask R-CNN: Wird für die Objekterkennung und Segmentierung eingesetzt.
- CLIP: Verbindet Text und Bilder und klassifiziert Bilder in natürlicher Sprache.
- GLIP (Grounded Language-Image Pre-training): Wird in diesem Projekt zur Erkennung von Personen verwendet; alternativ kann auch Mask R-CNN für die Objekterkennung und Segmentierung genutzt werden.

- OWL-ViT (Vision Transformer for Open-World Localization): Ein mögliches Modell für die Objekterkennung, das jedoch in dieser Thesis nicht verwendet wird.
- TCL (Vision-Language Pre-Training with Triple Contrastive Learning): Wird für die Bildklassifizierung eingesetzt und könnte CLIP ersetzen, wird jedoch in dieser Thesis nicht verwendet.
- X-VLM (Learning Multi-Grained Vision Language Alignments): Kann ebenfalls als Ersatz für CLIP bei der Bildklassifizierung verwendet werden, wird jedoch wie TCL in dieser Thesis nicht genutzt.
- Depth (Depth Estimation Model): Wird zur Tiefenschätzung von Objekten im Bild eingesetzt und kann eines der drei Modelle MiDaS_small, DPT_Hybrid oder DPT_Large verwenden. Das Standardmodell ist DPT_Large.
- BLIP (Bootstrapping Language-Image Pre-training): Dieses Modell wird verwendet, um einfache Fragen zum Bild oder zu Bildausschnitten zu beantworten, d.h. Fragen, die keine externen Wissensquellen oder komplexe Schlussfolgerungen erfordern. Diese Fragen reichen von der Erfassung von Eigenschaften bis hin zu Positionsfragen im Bild. Dieses Modell ist auch der Grund, warum ViperGPT viel GPU-VRAM benötigt. Es kann als weniger komplexe Version des gesamten Tools betrachtet werden.
- Saliency (InSPyReNet: Image Pyramid Structure for High Resolution Salient Object Detection): Dies ist ein auf Aufmerksamkeit basierendes Objekterkennungsmodell, das jedoch im aktuellen Projekt und in dieser Thesis nicht verwendet wird.
- Codex (OpenAI Codex): Das ursprüngliche ViperGPT-Projekt von 2023 nutzte die ältere OpenAI Codex API zur Codegenerierung. Diese wurde später durch GPT-3 ersetzt, das Projekt bezeichnet das Code-generierende Modell jedoch weiterhin mit der Codex-Bezeichnung.
- CodeLlama: Eine ältere Implementierung des CodeLlama LLM, die veraltet und für die Thesis ungenutzt ist. Stattdessen wird LM-Studio verwendet.
- GPT-3: Das GPT-3-Modell wird für Fragen verwendet, die mehr Wissen und Schlussfolgerungen erfordern. Die Fragen werden über die OpenAI API und das Modell GPT-3.5-turbo beantwortet, unabhängig davon, ob LM-Studio verwendet wird. Die Bezeichnungen „qa“ und „general“ kennzeichnen die Größe der Frage bzw. den Umfang des an GPT-3 gesendeten Prompts.
- LM-Studio: Dies ist meine eigene Implementierung der LM-Studio API, die ausschließlich zur Codegenerierung genutzt wird. Weitere Details zur Implementierung finden sich in Abschnitt 3.4 und zusätzliche Informationen zu den LLMs in Abschnitt 4.1.1.

## 3.2. Installation und Konfiguration

Die GitHub-Seite des Projekts enthält eine Installationsanleitung, die jedoch nicht vollständig ist. Es fehlen wichtige Informationen wie:

- Welches Betriebssystem genutzt werden soll.
- Ob Miniconda oder Anaconda verwendet werden sollte.
- Die Standardeinstellungen der Konfigurationsdateien.
- Welche Python-Version verwendet werden soll.

Während der Installation konnten die bereitgestellten Bash-Dateien nicht für das Setup genutzt werden; jeder Befehl musste einzeln ausgeführt werden. Außerdem waren vier der benötigten Pakete veraltet, wobei drei davon glücklicherweise in einer neueren Version verfügbar waren. Das einzige Paket, das nicht installiert werden konnte, war „qd“, das jedoch für die Nutzung des Tools nicht notwendig war.

Die Paketinstallation wurde entsprechend angepasst, sodass sie jetzt reibungslos funktioniert, zumindest bis ein weiteres Paket nicht mehr verfügbar ist. Der letzte Installationsstand war der 04.06.2024.

1. Die Installation ist abgesehen von den genannten Problemen sehr einfach:
2. Anaconda installieren.
3. Ein neues Environment namens „vipergpt“ erstellen.
    a. Oder benutze eine Kopie des Environments
    b. conda install --name vipergpt --file spec-file.txt
    c. danach bei Schritt 8b weiter
4. Das Environment aktivieren.
5. Falls noch nicht installiert, folgende Pakete installieren: gdown, wget, pip.
6. Die Dateien des Projekts in einen leicht zugänglichen Ordner verschieben.
7. Über die Konsole in das Verzeichnis wechseln.
8. Folgende Befehlskette ausführen:
    a. conda activate vipergpt
    b. conda install pytorch==1.13.1 torchvision==0.14.1 torchaudio==0.13.1 pytorch-cuda=11.6 -c pytorch -c nvidia
    c. pip install -r requirements.txt
9. Dies sollte alle benötigten Pakete und Grundlagen installieren.
10. Die integrierten Modelle werden über die Datei downloadmodell.bat installiert.
11. Anschließend wird GLIP installiert:
    a. cd glip
    b. python setup.py clean --all build develop --user
    c. cd ..
12. Nun muss der OpenAI API-Schlüssel registriert werden:
    a. echo DEIN_API_KEY > api.key
13. Damit sollte das Standardprojekt installiert sein.

Man kann jetzt in der Konsole mit dem Befehl jupyter notebook das Projekt durchsuchen. In der Datei main_simple.ipynb befindet sich der ausführbare Code, mit dem das Tool getestet werden kann, so wie es für diese Thesis durchgeführt wurde.

Falls man sich mit Environments in VS Code auskennt, kann man auch VS Code für das Testen nutzen. Dies ist jedoch für das grundlegende Verständnis hier nicht notwendig.

## 3.3. Fehlerkorrektur

Der ursprüngliche Code von ViperGPT war veraltet und musste in mehreren Bereichen angepasst werden:

- Die API von OpenAI musste auf das neue System aktualisiert werden.
- Die Funktion GetCode() wurde angepasst, um die Verwendung verschiedener Modelle zu ermöglichen.
- Die Funktion execute_code() wurde vollständig ersetzt durch eine neue Funktion ohne Visualisierung.
- Das Batch-System, das ursprünglich automatisch in der vision_model.py organisiert war, wurde nun in der main_simple.ipynb automatisiert.
- Die replace-Funktion wurde repariert, um Code-Segmente und Anfragen an das LLM korrekt zu ersetzen.

## 3.4. Erweiterungen

Folgende Erweiterungen wurden am ursprünglichen Code vorgenommen:

Integration von LM-Studio in der vision_model.py:

- In der vision_model.py wird die jeweilige forward()-Funktion pro Modell platziert. Jedes Modell hat einen eigenen Ablauf, da jedes Modell spezifische API-Spezifikationen besitzt und anders aufgerufen wird. Die Erweiterung für LM-Studio basiert auf der OpenAI-Bibliothek, die von LM-Studio als API genutzt wird, um mit den jeweiligen Modellen zu kommunizieren. Der Ablauf ähnelt dem der GPT-Modelle.

Außerhalb des Originalcodes wurden folgende Erweiterungen implementiert:

- Automatisierung der Codegenerierung durch LLMs.
- Automatisierung der Codeausführung über verschiedene Computer-Vision-Modelle.
- Ein Speichersystem für verschiedene Code-Segmente, Einstellungen und Ergebnisse.
- Funktionen zur Wiederherstellung von Variablen aus den gespeicherten Dateien.
- Ein Timing-System, das die Bildanalyse abbricht, falls diese zu lange dauert.
- Erweiterte Konfigurationsmöglichkeiten speziell für die Nutzung von LM-Studio.

## 3.5. Nutzung des Tools

Um das Tool nutzen zu können, benötigt der Testcomputer eine gewisse Rechenleistung, insbesondere eine leistungsstarke GPU oder mehrere schwächere GPUs. Mindestens 12 GB VRAM und idealerweise 16 GB Arbeitsspeicher sind erforderlich. Für größere LLMs wird zusätzlicher Arbeitsspeicher benötigt. Dies war auch der Grund, warum keine hoch quantisierten LLMs verwendet wurden, da der Testcomputer nur über 16 GB VRAM und 64 GB RAM verfügt.

Um das Tool zu nutzen, wenden wir uns der Datei main_simple.ipynb zu, in der zunächst die richtigen Einstellungen vorgenommen werden müssen:

- imageFolderPath: Pfad zum Ordner mit den Bildern (z. B. .testing/images).
- queryTextPath: Pfad zur Datei, in der die API-Anfrage gespeichert ist, die an das LLM gesendet wird.
- logFolderPath: Pfad zum Ordner, in dem die Log-Dateien mit den Ergebnissen gespeichert werden.
- logNRFolderPath: Pfad zum Ordner, der nur Log-Dateien mit Code-Segmenten enthält.
- UseImageDisplay: Aktivierung der Bildanzeige während der Codeausführung.
    - getCodeFromLog: Auswahl der Quelle des Codes:
    - 0 für eine einzelne Log-Datei,
    - 1 für mehrere Log-Dateien,
- jede andere Einstellung für keine Log-Datei, was zur Codegenerierung führt.
- codeLogFilePath: Pfad zur einzelnen Log-Datei.
- multipleLogsFolderPath: Pfad zum Ordner mit mehreren Log-Dateien.
- codeLogPartName: Identifikation der mehreren Log-Dateien (Teil des Namens).
- codeGeneratorAi: Auswahl der KI für die Codegenerierung:
    - lm_studio für die LM-Studio-KIs,
    - codex für Online-KIs wie GPT-3-Turbo oder GPT-4.

In der my_config.yaml im configs-Ordner sind die Hintergrund-Einstellungen des ursprünglichen ViperGPTs und der LM-Studio-KI-Erweiterung enthalten. Diese sind für die Tests vorkonfiguriert, wobei

die wichtigste Einstellung die Wahl des Modells im Abschnitt codex oder lm_studio ist. Dort wird die genaue Bezeichnung der KI festgelegt.

# 4. Testing

Das Testen ist die Hauptleistung, die in dieser Thesis erbracht wurde. Ziel war es, die Fähigkeiten aktueller Large Language Models (LLMs) und Computer-Vision-Modelle (CV) zu prüfen und insbesondere deren Grenzen aufzuzeigen. Die Tests umfassten:

- Vergleiche zwischen verschiedenen KIs.
- Tests verschiedener Fähigkeiten wie Zählen, Farberkennung, Texterkennung, Erkennung bekannter und unbekannter Objekte, Beschreibung, Wissensbeschaffung und der Umgang mit Fiktionalem.
- Tests mit variierter Wortinterpretation.
- Tests mit bedeutungsähnlichen Anfragen.
- Vergleichstests zwischen KIs und Menschen.

## 4.1. Aufbau der Tests

Die Tests bestanden aus 51 Anfragen (Queries) und 24 Bildern. Jede Query ist in der Datei Testing/query.txt zu finden, und alle Bilder befinden sich im Ordner Testing/Images. Es wurden zehn LLMs verwendet, um den Code zu generieren, was den ersten Schritt darstellt. Nacheinander wird der Code für alle LLMs generiert und in Logs gespeichert. Jeder Durchlauf, in dem ein Modell den Code für alle 51 Queries generiert, wird als „Generation Loop“ bezeichnet.

Es gibt vier Generation Loops pro Modell, was hochgerechnet auf 51 * 10 * 4 = 2040 Code-Segmente führt. Um bei einer solch großen Anzahl an Ergebnissen eine klare Struktur zu gewährleisten, wurden bestimmte Vorkehrungen getroffen.

Bei der Generierung wurden die LLMs, sofern möglich, wie folgt konfiguriert:

- Temperatur auf 0: Dies reduziert die Zufälligkeit der Modelle, sodass bei mehreren Durchläufen derselbe Code produziert wird. Dies dient der Reproduzierbarkeit der Tests.
- Maximale Tokens: 512: Dies stellt sicher, dass die Codegenerierung abbricht, falls zu viel generiert wird.
- System Prompt: „Only answer with a function starting def execute_command“. Diese Anweisung zwingt das Modell dazu, nur den Codeblock zu generieren, der zur Ausführung des Codes benötigt wird, und begrenzt somit die Menge an zusätzlichem Text, den das Modell produziert, wie z. B. Erklärungen oder zufällige Textfragmente.
- frequency_penalty: 0 und presence_penalty: 0: Diese Einstellungen sorgen dafür, dass Tokens nicht durch ihre Häufigkeit im Code beeinflusst werden. Dies ist nützlich, da Code häufig dieselbe Syntax verwendet (z. B. if, else, for). Das Standardverhalten (penalty = 0,01) bestraft Modelle dafür, dass sie Tokens häufig wiederholen, was jedoch bei Code kontraproduktiv sein kann.
- Prompt: Der Prompt für die Modelle ist in prompts/chatapi.prompt zu finden. Er enthält alle Funktionen der API sowie deren Beschreibungen und einige Anwendungsbeispiele. Der Prompt endet mit der allgemeinen Anweisung an das LLM:
  ““Write a function using Python and the ImagePatch class (above) that could be executed to provide an answer to the query.
  Consider the following guidelines:
  - Use base Python (comparison, sorting) for basic logical operations, left/right/up/down, math, etc.

- Use the llm_query function to access external information and answer informational questions not concerning the image.
  Query: INSERT_QUERY_HERE““

Dieser Prompt zwingt das Modell dazu, nur einen Codeblock zu liefern, der die angegebenen Funktionen und Standard-Python verwendet, anstatt auf andere Frameworks oder Bibliotheken zurückzugreifen. Der Platzhalter „INSERT_QUERY_HERE“ wird vor dem Senden an das Modell durch die entsprechende Query aus der query.txt-Datei ersetzt.

Wie bereits erwähnt, erfolgt der gesamte Codegenerierungsprozess ohne Bezugnahme auf die Bilder. Dies bedeutet jedoch auch, dass der Code nach seiner Generierung nicht mehr verändert werden kann, selbst wenn er nach einer ersten Bildausführung verbessert werden könnte.

Im nächsten Schritt des Testprozesses werden alle Ergebnisse jedes Modells gespeichert. Zu diesem Zweck wurde die Funktion log_everything erstellt, die alle Basisinformationen wie Konfiguration, Zeit und Modell sowie die Code-Segmente, zugehörigen Queries und Bilder speichert. Für jeden Generation Loop wird ein Log gespeichert, das alle 51 Queries, Bilder und Codes enthält. Diese Logs befinden sich im Ordner testing/LogsNoResult.

Dieser Schritt kann so oft wie nötig wiederholt werden; jeder Generation Loop generiert 51 weitere mögliche Code-Segmente.

Im folgenden Schritt gibt es verschiedene Optionen: Entweder werden die zuletzt generierten Codes verwendet, eine einzelne Log-Datei wird geladen und die Codes extrahiert, oder mehrere Log-Dateien werden geladen, extrahiert und verglichen. Die ersten beiden Optionen sind unkompliziert, die letzte hingegen ist komplexer. Da die Modelle versuchen, mit einer Temperatur von 0 zu generieren, ist es möglich, dass derselbe Code mit derselben Query mehrfach generiert wird. Da es unnötig wäre, denselben Code mehrmals auszuführen, werden die Code-Segmente desselben Modells und derselben Query verglichen und nur ein eindeutiges Code-Segment beibehalten. Dies kann mit jeder Anzahl von Log-Dateien durchgeführt werden, solange sie dieselben Queries in derselben Reihenfolge enthalten.

Sobald eine geeignete Auswahl an Code-Segmenten vorhanden ist, kann die Ausführungsphase beginnen. In dieser Phase wird für jede Query und jedes Bildpaar der entsprechende Code ausgeführt. Während der Code ausgeführt wird, werden Funktionen aus der API aufgerufen und genutzt, um das Bild zu analysieren oder Fragen an GPT-3 zu stellen. Treten Fehler auf, stoppt der Code, und der Fehler wird als Ergebnis gespeichert. Erfolgt keine Fehler, wird der Rückgabewert als Ergebnis gespeichert.

Nach Abschluss der Ausführungsphase werden alle Ergebnisse, Queries, Bildpfade und Code-Segmente in einer neuen Log-Datei im Ordner testing/logs gespeichert. Damit ist der Testprozess für die LLMs abgeschlossen.

Für menschliche Tester ist der Prozess einfacher. Jede Person wird individuell getestet, um Gruppenwissen zu vermeiden, das die Ergebnisse verfälschen könnte. Die Regeln für den Test sind wie folgt:

- Der Tester stellt jede Frage/Query höchstens dreimal.
- Der Tester wird keine weiteren Erklärungen zu den Fragen geben.
- Jeder Testteilnehmer erhält das entsprechende Bild, das zur Query passt.
- Der Testteilnehmer hat so viel Zeit, wie er benötigt, und gibt dann eine Antwort, die als solche gekennzeichnet sein sollte, falls Unklarheiten bestehen.
- Falls eine einleitende Frage eines Frageblocks bereits beantwortet werden kann, wird die nachfolgende Frage mit anderer Formulierung übersprungen.
- Es werden mindestens 34 Fragen/Queries gestellt und maximal 51.

### 4.1.1. Genutzte LLMs

Um eine breite Varietät abzudecken, wurden zehn verschiedene Modelle genutzt. Diese Modelle unterscheiden sich in ihrer Größe, ihrem Training und ihrer Zensur. Die verwendete Quantisierung, also wie viele Bits pro Parameter genutzt werden, beträgt 4 Bit, bekannt als Q4. Die Quantisierungsmethode ist k (quant), und die Modellgröße ist M (medium), zusammengefasst als Q4_K_M. Diese Konfiguration wird bevorzugt, da sie ein ausgewogenes Verhältnis zwischen Qualität, Größe und Geschwindigkeit bietet. Eine höhere Quantisierung könnte zwar eine bessere Qualität liefern, würde jedoch erheblich mehr Zeit erfordern.

Online-Modelle:

Es gibt drei Modelle, die nicht lokal ausgeführt werden konnten. Diese stammen alle von OpenAI und sind nur über deren API zugänglich: [33]

- GPT-3.5-Turbo-0125: Dieses Modell steht hinter ChatGPT und wurde am 25. Januar 2024 veröffentlicht. Es ist aufgrund seiner Integration in ChatGPT das am häufigsten genutzte Modell. Es wurde bis September 2021 trainiert und hat einen maximalen Kontext von 16.385 Tokens.
- GPT-3.5-Turbo-Instruct: Dieses Modell ist identisch mit dem vorherigen, jedoch als Instruct-Modell konzipiert. Es ist darauf ausgelegt, spezifische Benutzeranweisungen präzise und effektiv auszuführen, im Gegensatz zum „Chat"-Modus, der sich auf die Aufrechterhaltung eines Gesprächsflusses konzentriert.
- GPT-4o: Dies ist das neueste Modell von OpenAI und verfügt über erweiterte Fähigkeiten im Vergleich zu den Vorgängermodellen. Es kann zusätzlich zu Text auch Video, Audio und Bilder als Eingabe verarbeiten. Das Modell wurde am 13. Mai 2024 veröffentlicht und hat einen Kontext von 128.000 Tokens.

Lokale Modelle von Huggingface:

Diese sieben Modelle wurden auf Huggingface veröffentlicht und sind frei verfügbar. Alle Modelle sind Varianten von Llama 2 oder 3 und unterliegen der Nutzungslizenz von Meta:

- CodeLlama 7B – 70B: Die CodeLlama-Modelle wurden durch Feinabstimmung von Llama 2 mit einer höheren Samplingrate von Code entwickelt. Wie bei Llama 2 wurden auch bei den feinabgestimmten Versionen des Modells umfangreiche Sicherheitsmaßnahmen ergriffen. Sie wurden am 24. August 2023 veröffentlicht. Für diese Thesis wurden vier Größen verwendet: 7B, [34] 13B, [35] 34B [36] und 70B, [37] sowie die Instruct-Variante des 34B-Modells. [38] Der Kontext beträgt 16.384 Tokens.
- WizardLM 1.0 (CodeLlama 34B) Uncensored: [39] Dieses Modell wurde als eine weitere Version des CodeLlama 34B-Modells erstellt, jedoch ohne die Sicherheitsregulierungen und -beschränkungen. Diese Entfernung von Sicherheitsmaßnahmen, bekannt als Uncensoring, kann dazu führen, dass das Modell jeden vom Benutzer gewünschten Text ohne moralische oder rechtliche Einschränkungen generiert. In dieser Thesis wurde das Modell verwendet, um zu prüfen, ob Sicherheitsregeln die Ergebnisse selbst bei legalen und moralischen Aufgaben wie dem Schreiben von CV-Code-Snippets beeinträchtigen.
- Meta Llama 3 70B - Instruct: [40] Dieses Modell ist Teil des Llama 3-Kontingents und wurde am 18. April 2024 veröffentlicht. Es handelt sich um eine neuere Version des vorherigen Open-Source-Modells Llama 2, das entwickelt wurde, um mit den zu diesem Zeitpunkt verfügbaren Modellen gleichzuziehen oder diese zu übertreffen. Das Modell wurde mit neueren Daten im Vergleich zu den Llama 2-Modellen trainiert und hat einen Kontext von 8.192 Tokens.

### 4.1.2. Befragte Menschen

Für die Thesis wurden zehn Personen unterschiedlicher Altersgruppen und Hintergründe befragt, alle freiwillig. Das Alter der Teilnehmer reicht von 10 bis 62 Jahren. Beide Geschlechter sind vertreten, ebenso wie unterschiedliche Bildungs- und Lebenserfahrungen. Sechs der zehn Personen sind Mitglieder der Pfadfinder.

## 4.2 Erklärung der Testfälle

Die Testfälle decken eine Vielzahl von Fähigkeiten ab. Der erste Fall basiert direkt auf dem im Paper beschriebenen Szenario: Es geht um das Zählen von Objekten und einfache mathematische Operationen. Das Bild zeigt zwei Kinder, die insgesamt acht Muffins essen, und die Frage zielt darauf ab, eine gerechte Aufteilung der Muffins unter den Kindern zu berechnen.

Das zweite Bild stammt nicht aus dem Paper, jedoch die dazugehörige Anfrage. Das Bild soll eine klarere Darstellung bieten als das ursprüngliche Bild. Dieser Test umfasst die Erkennung bekannter Objekte, einfache Interpretation und möglicherweise Rückfragen an ein LLM. Das Bild zeigt mehrere Getränke, und die Anfrage bezieht sich auf die alkoholfreien Getränke.

Die Anfragen 3 bis 8 betreffen dasselbe Bild eines möglichen Schulausflugs oder Kindergartenausflugs mit vier Kindern und einem Lehrer. Die erste Anfrage fragt einfach, wie viele Kinder auf dem Bild zu sehen sind, während die zweite nach der Farbe des Hemdes des letzten Kindes fragt. Da der Begriff „letztes“ zwei Kinder meinen könnte, wurden vier weitere Fragen gestellt, um zu klären, welches Kind genau gemeint ist. Dieser Test dient der Erkennung von Personen, der Interpretation und der Eigenschaftsbewertung. Das Bild und die Anfragen stammen nicht aus dem Paper.

Der neunte und zehnte Fall enthält insgesamt 20 Katzen in verschiedenen Positionen und mit unterschiedlicher Klarheit. Diese Bilder testen die variable Objekterkennung, um zu prüfen, ob die CV-Modelle alle Katzen erkennen und ihre Positionen (liegend, sitzend oder stehend) korrekt bestimmen können. Auch dieser Test war nicht Teil des Papers.

Der elfte Fall beinhaltet eine eingeschränkte Erkennung unbekannter Objekte. Hier wird jedoch nach einem Objekt gesucht, das im Hintergrund und nicht im Fokus steht, zudem ist das Objekt in einem Winkel dargestellt. Dieser Fall ist ebenfalls nicht Teil des Papers.

Die Fälle 12 bis 15 betreffen Objekterkennung und einfache Anfragen an das LLM. Die Bilder zeigen Blumen in den Farben Blau, Rot und Weiß, und es wird nach den Farben gefragt, wenn diese kombiniert oder gemischt werden. Aufgrund der Besonderheit in der Wortwahl müssen beide Anfragen getestet werden, ebenso wie zwei verschiedene Bilder. Die erste Anfrage stammt aus dem Paper, das zweite Bild ist dem im Paper verwendeten sehr ähnlich.

Der 16. Fall ist eine weitere Aufgabe zur Personenerkennung mit Interpretation, Eigenschaftsbewertung und mathematischen Berechnungen. Auf dem Bild sind acht Personen zu sehen: sieben Kinder und ein Lehrer. Hier soll das Verhältnis von Kindern zu Lehrern bestimmt werden.

Der 17. Fall testet die Fähigkeiten zur Erkennung unbekannter und bekannter Objekte sowie die Orientierung im Bild. Das Bild zeigt eine Katze auf einem Papp-Laptop mit Stickern auf der Rückseite, darunter ein Sticker mit einer Bombe. Die Anfrage lautet: „Was ist auf dem Sticker in der unteren rechten Ecke des Laptops zu sehen?“ Dieser Test war nicht Teil des Papers.

Der 18. Fall testet ähnliche Fähigkeiten wie der 17. Fall, aber mit einer anderen Bildkomposition. Das Bild zeigt mehrere Pizzen mit unterschiedlichen Belägen, und die Anfrage lautet: „Welche Beläge hat die Pizza in der oberen rechten Ecke?“ Auch dieser Test war nicht Teil des Papers.

Die Fälle 18 und 19 testen die Erkennung bekannter Objekte und die Interpretation sowie Bewertung von Objekten. Die Bilder zeigen 13 Tiere, davon 7 oder 8 verschiedene Arten, je nach Interpretation. Die Anfragen betreffen die Anzahl der Tiere sowie die Anzahl der verschiedenen Tierarten. Dieser Test war nicht Teil des Papers.

Der 20. Fall ist einer der komplexesten Testfälle. Er prüft die Erkennung bekannter Objekte, die Interpretation von Objekten, die Bewertung von Eigenschaften und die Befragung eines LLM. Das Bild zeigt zwei Wolkenkratzer (das Empire State Building und das Chrysler Building), und die Anfrage lautet: „Erzähle mir etwas über den Wettbewerb zwischen den beiden Wolkenkratzern auf dem Bild." Diese Anfrage war Teil des Papers, und das Bild ist eine sehr ähnliche Kopie des im Paper verwendeten Bildes. Dieser Fall war auch der erste, der zwingend eine Anfrage an eine externe Wissensdatenbank wie ein LLM erforderte.

Der 22. Fall ist ebenfalls sehr komplex und ähnelt dem 20. Fall. Hier werden die Erkennung bekannter Objekte, die Interpretation von Objekten, die Bewertung von Eigenschaften, die Befragung eines LLM und die Erkennung von Details getestet. Das Bild zeigt drei Autos (in der Reihenfolge Bentley, Mercedes, Ferrari), und die Anfrage lautet: „Was würde der Gründer der Marke des Autos links den Gründern der anderen Autos sagen?" Diese Anfrage war Teil des Papers, das Bild jedoch ist neu und zeigt andere Autos und Marken. Auch dieser Fall erfordert den Zugang zu einem LLM sowie eine genaue Detailerkennung, um die Marken der drei Autos zu identifizieren.

Die Fälle 23 bis 25 testen die Erkennung bekannter Objekte, die Interpretation von Objekten, die Bewertung von Eigenschaften und die Befragung eines LLM. Das Bild zeigt drei Tiere (in der Reihenfolge Löwe, Jaguar/Leopard, Tiger), und die Anfragen zielen darauf ab, welches Tier das schnellste ist. Diese Tests waren nicht Teil des Papers, und das Ergebnis ist aufgrund widersprüchlicher Quellen debattierbar, da das mittlere Tier entweder ein Jaguar oder ein Leopard sein könnte, was den Jaguar oder den Löwen zum schnellsten Tier machen würde.

Die Fälle 26 und 27 testen die Erkennung unbekannter Objekte, die Interpretation von Objekten und die Befragung eines LLM. Das Bild zeigt einen Teddybären, und die Anfragen zielen darauf ab, was das echte Tier in der Winterzeit macht. Diese Anfrage war Teil des Papers, das Bild jedoch nicht.

Der 28. Fall testet die Erkennung bekannter Objekte, die Interpretation von Objekten und die Befragung eines LLM. Das Bild zeigt einen Box-/Wrestling-Ring, und die Anfrage lautet: „Welche Sportarten werden typischerweise in der Arena auf dem Bild ausgeübt?" Auch dieser Test war nicht Teil des Papers.

Die folgenden Fälle (28 bis 36) erfordern Texterkennung / Zeichenerkennung. Alle Fälle wurden ohne konkrete Erwartungen an die Ergebnisse durchgeführt, da das Paper die Zeichenerkennung nicht erwähnt.

Der 29. Fall testet die Texterkennung / Zeichenerkennung. Das Bild zeigt das Wort „Apple", und die Anfrage lautet: „Was ist das Wort auf dem Bild?" Dies ist ein einfacher und direkter Test zur Zeichenerkennung, der nicht Teil des Papers war. Er soll prüfen, wie die Modelle mit Text umgehen.

Die Fälle 30 bis 33 testen die Erkennung unbekannter und bekannter Objekte, die Zeichenerkennung, die Textinterpretation und einfache Anfragen an ein LLM. Die Bilder zeigen eine Fußballszene mit einem Werbebanner von Turkish Airlines, und die Anfragen betreffen die auf dem Banner angezeigte Firma, wobei zunehmend mehr Informationen in der Anfrage gegeben werden. Diese Tests waren nicht Teil des Papers und wurden verwendet, um zu bewerten, wie die CV-Modelle mit störenden und irreführenden Texten umgehen.

Die Fälle 34 und 35 testen die Erkennung unbekannter und bekannter Objekte, die Zeichenerkennung, die Textinterpretation und einfache Anfragen an ein LLM. Das Bild zeigt ein Zimmer mit vielen

Gegenständen sowie einen Rahmen, der die Worte „hoppy easter“ enthält. Die Anfragen zielen darauf ab, den Fehler / Rechtschreibfehler im Bild zu identifizieren. Diese Tests waren nicht Teil des Papers.

Die Fälle 36 und 37 testen die Erkennung bekannter Objekte, die Zeichenerkennung, die Textinterpretation und Anfragen an ein LLM. Das Bild zeigt mehrere Bücher in einem Regal, darunter mehrere Serien. Die Anfragen betreffen die Namen der Bücher im Regal sowie die Frage, welche die längste Buchserie ist. Diese Tests waren nicht Teil des Papers, und das Bild wurde von mir erstellt.

Die letzten Tests sind äußerst vage und daher eher unwahrscheinlich, zufriedenstellende Antworten zu liefern.

Die Fälle 38 bis 42 testen die Erkennung unbekannter und bekannter Objekte, die Texterkennung, die Textinterpretation sowie einfache und komplexe Anfragen an ein LLM. Die Bilder zeigen ein Klassenzimmer mit Schülern und einem Lehrer, der Papiere austeilt, während an der Tafel geometrische Formen zu sehen sind. Die zweite Szene zeigt einen Schüler, der an die Tafel schreibt, wobei Begriffe und Gleichungen zu sehen sind. Die Anfrage lautet: „Was lernen/schreiben die Schüler auf diesem Bild?“ mit zunehmend mehr Informationen im Prompt. Dieser Test war nicht Teil des Papers und aufgrund der Mehrdeutigkeit wurde nicht erwartet, dass er erfolgreich sein würde.

Die Fälle 43 bis 47 testen die Erkennung unbekannter und bekannter Objekte, die Erkennung fiktiver Charaktere sowie einfache und komplexe Anfragen an ein LLM. Das Bild zeigt eine Werbegrafik für das Spiel League of Legends mit vier ihrer prominentesten Charaktere: Lux, Jinx, Yasuo und Blitzcrank. Die Anfragen betreffen zunächst die Beschreibung des Bildes, dann die Frage, wie viele Charaktere auf dem Bild zu sehen sind, und schließlich, wer der älteste Charakter ist, wobei zunehmend mehr Informationen im Prompt gegeben werden. Dieser Test war nicht Teil des Papers und wurde verwendet, um zu bewerten, wie die Modelle mit fiktiven Charakteren umgehen.

Die Fälle 48 bis 51 testen die Erkennung unbekannter und bekannter Objekte, die Erkennung dargestellter Charaktere, die Interpretation von Objekten/Personen, die Bewertung von Eigenschaften und einfache sowie komplexe Anfragen an ein LLM. Das Bild zeigt Jesus am Kreuz, und die Anfrage lautet: „Welcher Feiertag ist auf dem Bild dargestellt/mit dem Bild verbunden?“ mit zunehmend mehr Informationen im Prompt. Dieser Test war nicht Teil des Papers und wurde verwendet, um ein komplexes thematisches Verständnis des Bildes zu testen.

Diese Testfälle wurden alle genutzt, um ViperGPT zu testen. Einige sind bekannt dafür, dass sie funktionieren, während andere weit außerhalb des normalen Anwendungsbereichs liegen.

## 4.3. Erwartungen im Hinblick auf das ViperGPT-Paper

Das ViperGPT-Paper [31] lieferte viele Beispiele und Erklärungen zu den Fähigkeiten des Projekts. Diese umfassten:

- Logisches Denken
- Räumliches Verständnis
- Wissen von LLMs
- Konsistenz in den Antworten
- Mathematische Fähigkeiten
- Objekterkennung und -attribute
- Relationales Denken

Auf Grundlage der im Paper beschriebenen Anfragen sollte ViperGPT in der Lage sein, mindestens die folgenden Fälle zu beantworten: 1, 2, 12, 13, 14, 15, 21, 26 und 27. Angesichts der versprochenen Funktionalität und der im Paper gegebenen Beispiele sollte es außerdem in der Lage sein, die Fälle 3,

4, 5, 6, 7, 8, 11, 16, 17, 19, 20, 22, 23, 24, 25, 48, 49, 50 und 51 zu beantworten. Schließlich sollte es aufgrund der im Code und in der API gezeigten Fähigkeiten auch in der Lage sein, die Fälle 29, 32, 33, 34, 35, 38, 39, 40, 41 und 42 zu beantworten.

Dies würde zu einer Erfolgsquote von 38/51, also 74,5 % führen. Persönlich erwartete ich jedoch eine niedrigere Erfolgsquote, näher bei 40 %. Diese Erfolgsrate berücksichtigt die Ausgaben aller verwendeten LLMs.

Ein weiterer wichtiger Aspekt ist das Modell, das zur Codegenerierung verwendet wurde. Zum Zeitpunkt der Veröffentlichung war dies das Codex-Modell von OpenAI. Da dieses Modell mittlerweile eingestellt wurde, können wir nicht genau mit denselben Parametern testen, was zu niedrigeren Erfolgsraten führen könnte.

Leider konnte aufgrund der Unmöglichkeit, das Blip-xxl-Modell zu verwenden, die Genauigkeit aller Ergebnisse geringer ausfallen, da das xl-Modell signifikant kleiner ist. Dasselbe gilt für alle lokalen LLMs, da diese kleiner sind als das ursprüngliche Codex-Modell, was ebenfalls zu einer Verringerung der Erfolgsrate führen könnte.

# 5. Auswertung

In der Auswertung wurden viele auffällige Ergebnisse deutlich. Um diese Ergebnisse zu erläutern, werden zunächst die allgemeinen Ergebnisse besprochen, bevor jedes LLM im Detail analysiert wird. Hierzu werden die Statistiken verwendet, die in der Datei „Ergebnisse AI.xlsx“ aufbereitet wurden. Die Daten in der Datei wurden mithilfe des Skripts answertesting.py generiert und nachträglich manuell überprüft, insbesondere bei Antworten mit mehreren Sätzen oder Ergebnissen. Bei der Generierung werden alle Ergebnisse eines LLMs zusammengeführt und gemeinsam ausgewertet. Solange eine der vier pro Query generierten Antworten korrekt ist, wird die Antwort als richtig markiert (mit einer „1“).

## 5.1. Allgemeine Ergebnisse

Zunächst können die allgemeinen Statistiken betrachtet werden. Diese zeigen, dass es insgesamt 96 richtige Ergebnisse gab. Bei einer Gesamtzahl von 51 * 10 = 510 Ergebnissen entspricht dies einer Erfolgsrate von 18,82 %. Die Tabelle zeigt nicht nur die richtigen oder falschen Antworten, sondern auch die Anzahl der generierten Fehler (Fehler im Code und fehlender Code). Wenn etwas als Fehler markiert ist, bedeutet dies, dass die Generierung in irgendeiner Weise fehlgeschlagen ist und daher nicht ausgeführt werden konnte. Die Statistik zeigt 172 bei 510 Ergebnissen, was einer Fehlerquote von 33,72 % entspricht. Dies ergibt 222 falsche Ergebnisse, was einer Rate von 43,53 % entspricht. Diese Ergebnisse können in Abbildung 1 eingesehen werden.

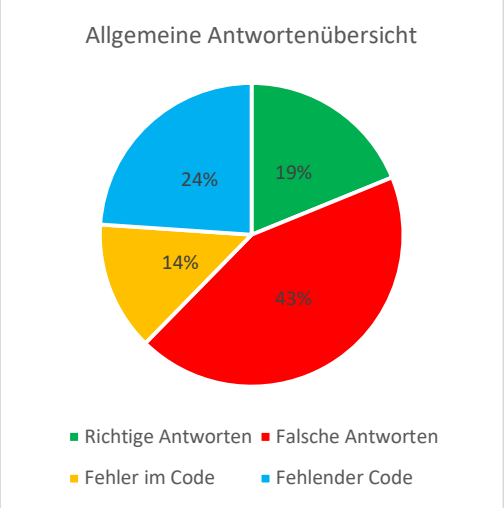


*Abbildung 1: Allgemeine Antwortenübersicht*

Wichtige Information zu den Ergebnissen: Am Ende der Thesis-Phase, vier Tage vor der Abgabe, fiel mir auf, dass Python die zuvor generierte Funktion def execute_command nach Abschluss der Ausführung nicht löschte. Dies beeinträchtigte die Ergebnisse, da aufeinanderfolgende Fälle mit derselben Funktion beantwortet wurden, was in der Realität die Anzahl der Fehler und möglicherweise der richtigen Antworten erhöht hat. Dies wurde mir leider erst bei der Endüberprüfung bewusst. Ich habe den Ursprung des Problems untersucht und gefunden. Die

Antworten wurden nach dieser Erkenntnis erneut in answerTesting.py gefiltert und sollten jetzt korrekt sein. Dies verursacht jedoch eine Diskrepanz zwischen den Logs und der Datei „Ergebnisse AI.xlsx“.

## 5.2. Vergleich der LLMs

Um über alle LLMs hinweg eine angemessene Länge des Textes zu wahren, wird nur selten auf einzelne Ergebnisse eingegangen. Falls spezifische Ergebnisse gewünscht sind, können diese im Anhang unter den Logs sowie in der Datei „Ergebnisse AI.xlsx“ eingesehen werden. Jedes der Modelle wurde auf Auffälligkeiten im Code sowie auf die Qualität der Antworten überprüft. Hier wird nicht auf jeden einzelnen Testfall eingegangen, aber unter Abschnitt 5.2.2 werden die wichtigsten Fälle besprochen.

### 5.2.1. Ergebnisse und Auffälligkeiten der einzelnen LLMs

Modell: CodeLlama-7B
Dieses Modell hatte folgende Statistik nach Abbildung 2:

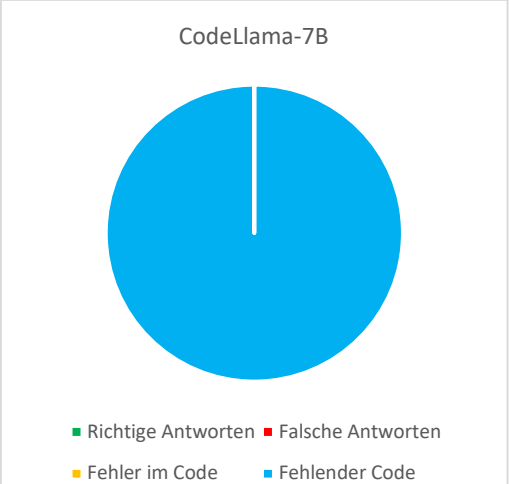


*Abbildung 2: Statistik CodeLlama 7B*

- Richtige Antworten: 0
- Falsche Antworten: 0
- Fehler im Code: 0
- Fehlender Code: 51

Dies waren die Auffälligkeiten:

- Das Modell beginnt immer mit Import-Anweisungen.
- Je nach Länge der Import-Anweisungen wird der Rest des Codes mit der API-Dokumentation des Prompts gefüllt.
- Die Funktion „def execute_command“ wird nur zweimal verwendet, und in beiden Fällen wird die Funktion lediglich mit der API befüllt.
- Testfälle 18-25, 27-29, 30-36, 38-43, 45-51 resultieren in extrem langen Einzeilercodeblöcken, die ebenfalls mit dem oben beschriebenen Code gefüllt sind.
- Alle Codeblöcke wurden bis zum Tokenlimit von 512 Tokens gefüllt, wodurch die Generierung abgebrochen wurde.
- Das Modell zeigte keine Fähigkeit, den Instruktionen zu folgen.
- Es war das einzige Modell, das keine Antworten liefern konnte.

Zusammenfassende Aussage:
Das Modell war höchstwahrscheinlich zu klein, um in irgendeiner Weise den richtigen Code zu generieren.

Modell: CodeLlama-13B
Dieses Modell hatte folgende Statistik nach Abbildung 3:

- Richtige Antworten: 1
- Falsche Antworten: 5
- Fehler im Code: 4
- Fehlender Code: 41

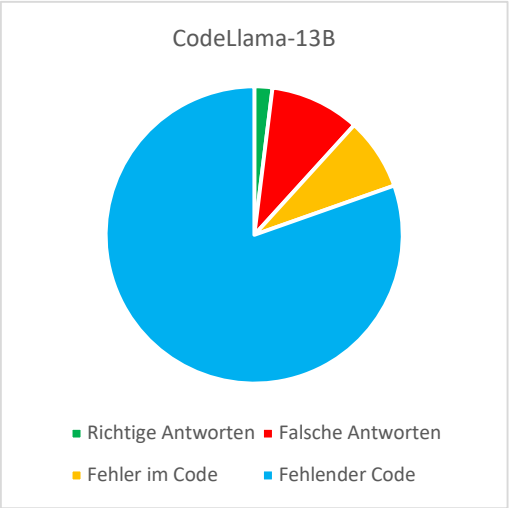

*Abbildung 3: Statistik CodeLlama 13B*

Dies waren die Auffälligkeiten:

- Es wurden 10 Codeblöcke mit „def execute_command“ generiert, von denen 6 ausgeführt werden konnten. Die übrigen 4 wurden mit der API befüllt.
- Der Rest der Codeblöcke war praktisch identisch und begann alle mit „### Solution“, gefolgt von der API.
- Der richtige Code für Fall 42 war gültig und die Ausführung reproduzierbar, was eine der wenigen richtigen Lösungen darstellt.
- Das Modell kann den Instruktionen folgen, neigt jedoch, ähnlich wie das 7B-Modell, dazu, falsche Antworten zu liefern.
- Es konnte einen der am häufigsten richtig beantworteten Fälle lösen, sonst jedoch nichts.

Zusammenfassende Aussage:

Das Modell war wahrscheinlich immer noch zu klein, um den richtigen Code zu generieren. Mit einer höheren Temperatur in den Einstellungen hätte es möglicherweise mehr richtige Codeblöcke generiert. Auch durch gezieltes Prompting könnten die anderen Probleme möglicherweise gelöst werden.

Modell: CodeLlama-34B
Dieses Modell hatte folgende Statistik nach Abbildung 4:

- Richtige Antworten: 5
- Falsche Antworten: 19
- Fehler im Code: 6
- Fehlender Code: 21

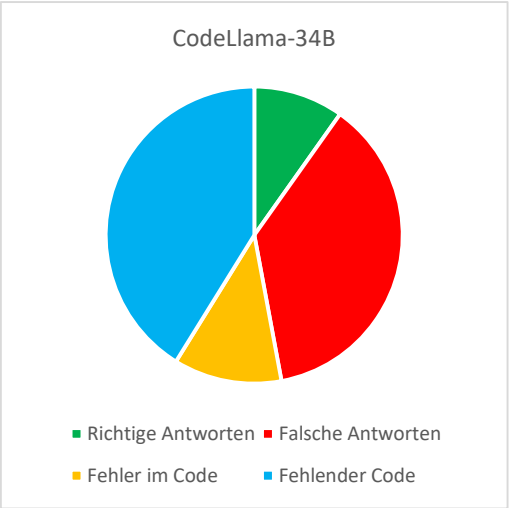

*Abbildung 4: Statistik CodeLlama 34B*

Dies waren die Auffälligkeiten:

- Bei 30 der 51 Fälle wurde die Funktion „def execute_command“ korrekt generiert.
- In den übrigen 21 Fällen wurde entweder ein textbasierter Code auf Grundlage der API generiert oder es traten 6 fehlerhafte Codeblöcke auf, die durch falsche Nutzung der API oder falsche Interpretation der Query entstanden.
- Das Modell nutzte Listen von Strings, anstatt beispielsweise nach Farben zu fragen.
- Es gab logische Probleme, selbst bei einfachen Fragen wie in den Fällen 1 und 10, da die Reihenfolge der Ausführung oft keinen Sinn ergab.
- Das Modell nutzte die Funktionalität der Texterkennung.

Zusammenfassende Aussage:
Das Modell hatte Probleme mit den logischen Abläufen und der Interpretation der Query. Die Größe allein reicht leider nicht aus, um den Instruktionen des Prompts und der API zu folgen. Da einige Codeblöcke richtig generiert wurden, wäre es mit einer höheren Temperatur möglicherweise möglich, die fehlenden und fehlerhaften Codeblöcke korrekt zu generieren.

Modell: WizardLM-1.0
Dieses Modell hatte folgende Statistik nach Abbildung 5:

- Richtige Antworten: 5
- Falsche Antworten: 35
- Fehler im Code: 11
- Fehlender Code: 0

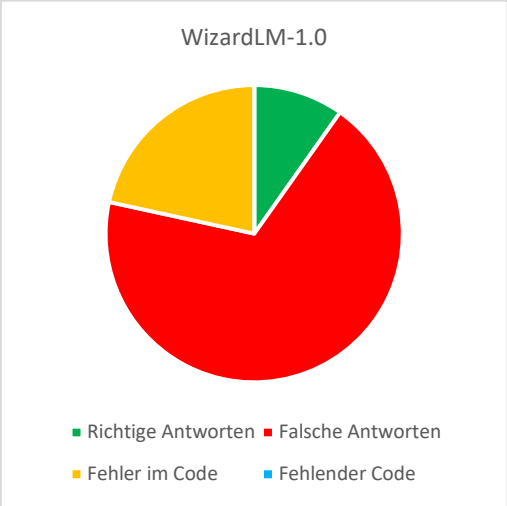

*Abbildung 5: Statistik WizardLM-1.0*

Dies waren die Auffälligkeiten:

- Die Fehler im Code resultierten oft aus der falschen Nutzung der API, insbesondere der Funktion simple_query.
- Das Modell nutzte Listen von Strings, anstatt beispielsweise nach Farben zu fragen.
- Der fehlerhafte Code war oft logisch korrekt, scheiterte jedoch an der Nutzung der simple_query-Funktion.
- Es war das einzige Modell der 34B-Klasse, das keine fehlenden Codeblöcke generierte.
- Es gab logische Probleme, selbst bei einfachen Fragen wie in den Fällen 1 und 13-15, bei denen die Reihenfolge der Ausführung manchmal keinen Sinn ergab. Antworten wurden aus Texten zusammengesetzt, anstatt ein LLM nach der Antwort zu fragen.
- Von den richtigen Antworten waren 3 von 5 nur zufällig korrekt und nicht durch korrekten Code.

Zusammenfassende Aussage:
Das Modell hat definitiv noch logische Probleme, kann aber den Instruktionen folgen. Die Nutzung der API lässt jedoch zu wünschen übrig, da viele Fehler auftraten. Möglicherweise könnte das Modell mit genaueren Instruktionen mehr Probleme korrekt lösen. Die Unzensiertheit scheint dazu beizutragen, dass das Modell genauere Ergebnisse liefert.

Modell: CodeLlama-34B-Instruct
Dieses Modell hatte folgende Statistik nach Abbildung 6:

- Richtige Antworten: 4
- Falsche Antworten: 17
- Fehler im Code: 21
- Fehlender Code: 9

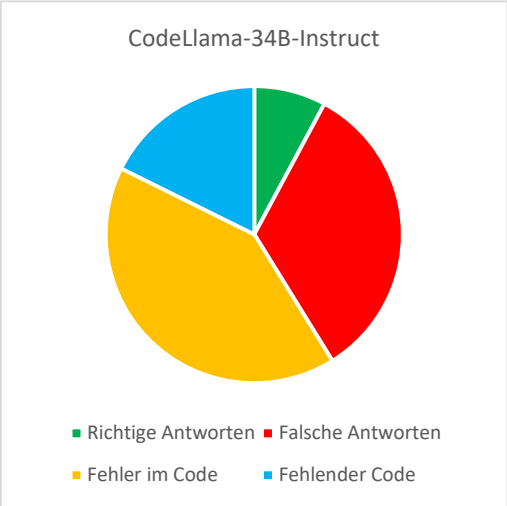

*Abbildung 6: Statistik CodeLlama 34B Instruct*

Dies waren die Auffälligkeiten:

- Das Modell schreibt oft Testfälle wie in einem Produktionscode.
- Viele der fehlerhaften Codestücke enthalten keinen Code, sondern lediglich die Aussage „Your Code here“.

- Die generierten Codestücke folgen fast immer einem logischen Ablauf, scheitern jedoch an der richtigen Nutzung der API oder an der Syntax.
- Bei ignorierten Anweisungen wird oft der Prompt mehrfach kopiert und angegeben.
- Das Modell hat die Tendenz, wiederholt die Anweisungen zu ignorieren.
- Die richtigen Codestücke sind logisch nachvollziehbar und nicht dem Zufall überlassen.

Zusammenfassende Aussage:
Das Instruct-Modell hat, entgegen den Erwartungen, die Anweisungen nicht gut befolgt, ist jedoch bei der Generierung von Codestücken relativ logisch vorgegangen. Es mangelt an der Nutzung der API sowie teilweise an der richtigen Syntax. Die richtigen Codestücke sind jedoch praktisch einsetzbar.

Modell: CodeLlama-70B
Dieses Modell hatte folgende Statistik nach Abbildung 7:

- Richtige Antworten: 9
- Falsche Antworten: 32
- Fehler im Code: 10
- Fehlender Code: 0

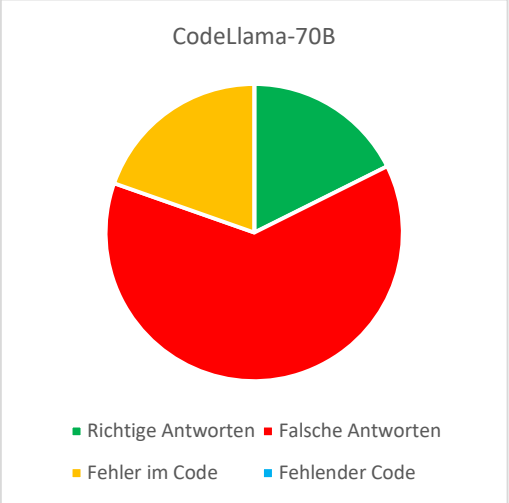

*Abbildung 7: Statistik CodeLlama 70B*

Dies waren die Auffälligkeiten:

- Das Modell hat keine fehlenden Codeblöcke, aber dafür 17 generierte Codeblöcke ohne tatsächlichen Code. Jeder dieser Blöcke hat einen zufällig generierten Rückgabewert, der die Ergebnistabelle verzerrte und zu 3 zufällig richtigen Antworten führte.
- Die richtigen Codestücke sind logisch und vom Ablauf her sehr praktisch.
- Das Modell erwartet zu viel von der API und generiert dadurch relativ unlogischen Code.
- Das Modell hat Probleme mit der richtigen Syntax. Fast jeder Fehler resultierte aus einem Syntaxproblem, außer 2, die durch falsche API-Nutzung verursacht wurden.
- Es folgt der Anweisung zur „def execute_command“-Funktion.

Zusammenfassende Aussage:
Das Modell kann den Anweisungen folgen, nutzt jedoch die API zu frei, um korrekte Ergebnisse zu generieren. Es generiert viele unbrauchbare Codeblöcke, und selbst den richtigen Ergebnissen kann man nicht vertrauen, da es viele zufällige Antworten ausgibt. Die Größe des Modells hat in diesen Fällen wenig gebracht.

Modell: Meta-Llama-3-70B
Dieses Modell hatte folgende Statistik nach  Abbildung 8:

- Richtige Antworten: 17
- Falsche Antworten: 27
- Fehler im Code: 7
- Fehlender Code: 0

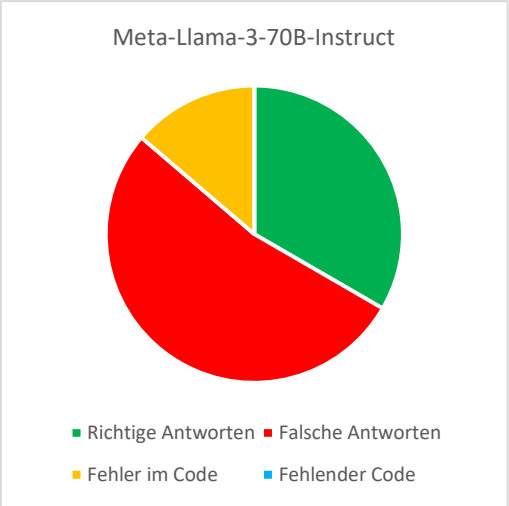

*Abbildung 8: Statistik Meta Llama 3 70B instruct*

Dies waren die Auffälligkeiten:

- Das Modell hat keine fehlenden Codestücke.
- Jedes der Codestücke enthält eine Erklärung am Ende.
- Es gab wenige Fehler in den Codestücken, 5 resultierten aus API-Problemen und 2 durch Syntaxfehler.
- Fast alle Codestücke sind logisch generiert.
- Es gab oft eine falsche Nutzung von simple_query und llm_query.
- Die Einschränkungen resultierten meist aus den CV-Modellen und nicht aus dem Code selbst.

Zusammenfassende Aussage:
Das Modell kann logischen und sinnvollen Code sehr gut generieren. Es lässt nichts dem Zufall überlassen, und nur sehr wenige der Codestücke sind nicht praktisch einsetzbar. Die Hauptlimitierung liegt in den CV-Modellen. Die Größe des Modells ist ausreichend, und selbst die Instruct-Version schränkt das Modell wenig ein.

Modell: GPT-3.5-Turbo-0125
Dieses Modell hatte folgende Statistik nach  Abbildung 9:

- Richtige Antworten: 17
- Falsche Antworten: 32
- Fehler im Code: 2
- Fehlender Code: 0

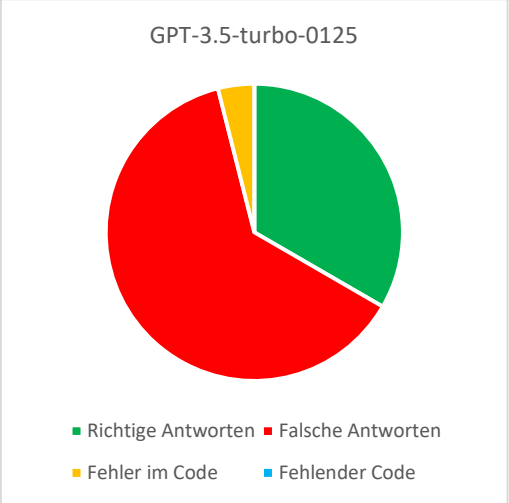

*Abbildung 9: Statistik GPT 3.5 turbo 0125*

Dies waren die Auffälligkeiten:

- Das Modell hat keine fehlenden oder leeren Codeblöcke generiert.
- Selbst mit einer Temperatur von 0 generiert das Modell noch variierte Codeblöcke.
- Der erste Fehler im Code resultierte aus einem fehlerhaften Ergebnis des CV-Modells.
- Der zweite Fehler im Code wurde durch die verschiedenen Varianten der Codeblöcke negiert, resultierte jedoch ursprünglich aus einem Problem bei der Erstellung von Listen anstatt nach Namen zu fragen, was in diesem Fall das Tokenlimit erschöpfte.
- Die Codeblöcke sind logisch aufgebaut und sollten praktisch nutzbar sein.
- Die Einschränkungen resultieren meist aus den CV-Modellen und nicht aus dem Code selbst.

Zusammenfassende Aussage:
Das Modell kann logischen und sinnvollen Code generieren. Nahezu jedes der Codestücke kann mit den richtigen CV-Modellen das richtige Ergebnis liefern. Die Größe erlaubt es, alle Instruktionen zu befolgen, und auch die API wird korrekt genutzt.

Modell: GPT-3.5-Turbo-Instruct
Dieses Modell hatte folgende Statistik nach Abbildung 10:

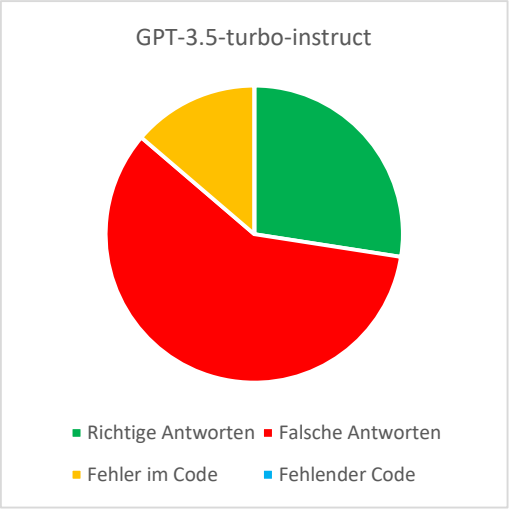

*Abbildung 10: Statistik GPT 3.5 turbo instruct*

- Richtige Antworten: 14
- Falsche Antworten: 30
- Fehler im Code: 7
- Fehlender Code: 0

Dies waren die Auffälligkeiten:

- Das Modell hat keine fehlenden oder leeren Codeblöcke generiert.
- Es gibt manchmal Probleme mit der Nutzung der API.
- Es gibt oft Probleme mit dem logischen Ablauf des Codes, insbesondere bei Fall 1, der das Logikproblem deutlich zeigt.
- Die Einschränkungen resultieren oft aus den CV-Modellen und nicht aus dem Code selbst.
- Es gibt eine gewisse Variation, 58 verschiedene Codeblöcke anstelle von 51, obwohl die Temperatur auf 0 gesetzt ist.
- Die fehlerhaften Codeblöcke resultieren aus Problemen mit der Syntax.

Zusammenfassende Aussage:
Das Modell kann die Instruktionen befolgen, nutzt jedoch oft die API nicht richtig oder generiert Syntaxfehler. Das Instruct-Modell ist groß genug, wird aber wahrscheinlich durch die Instruct-Eigenschaft geschwächt. Die größte Einschränkung liegt auch hier in den CV-Modellen.

Modell: GPT-4o
Dieses Modell hatte folgende Statistik nach Abbildung 11:

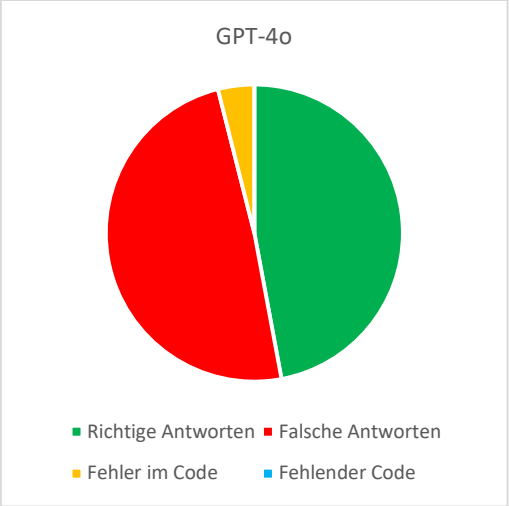

*Abbildung 11: Statistik GPT 4o*

- Richtige Antworten: 24
- Falsche Antworten: 25
- Fehler im Code: 2
- Fehlender Code: 0

Dies waren die Auffälligkeiten:

- Das Modell hat keine fehlenden oder leeren Codeblöcke generiert.
- Es generierte die meisten Variationen, 106 Codeblöcke anstelle von 51.
- Diese Variation sorgte auch dafür, dass das Modell einem der beiden Fehler im Code entgehen konnte und dennoch Ergebnisse lieferte.
- Es ist das einzige Modell, das versucht hat, Unterfunktionen zu generieren, was den ersten Fehler verursachte.
- Das Modell nutzt die API richtig und es gibt keine Syntaxfehler.
- Fast alle Codeblöcke sind logisch und praktisch einsetzbar, und jeder der 51 Fälle hat einen funktionsfähigen Codeblock.
- Die einzige große Einschränkung resultiert aus den CV-Modellen und nicht aus dem Code selbst.

Zusammenfassende Aussage:
Dieses Modell kann den Code für das Projekt korrekt generieren. Die Variation hilft dabei, Fehler zu beseitigen und erlaubt viele weitere richtige Antworten. Die einzige auffällige Einschränkung kommt von den CV-Modellen.

### 5.2.2. Ergebnisse und Auffälligkeiten im Modellvergleich

Zunächst vergleichen wir die Anzahl der richtigen Antworten, unsere wichtigste Metrik. Siehe dazu Abbildung 12, die Folgendes zeigt:

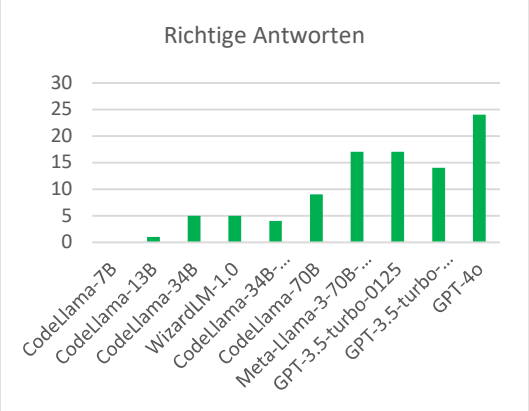


*Abbildung 12: Statistik Richtige Antworten*

Je größer das Modell, desto mehr richtige Ergebnisse gibt es. Die Neuheit der Modelle und höchstwahrscheinlich deren Trainingsmethoden haben großen Einfluss auf die Richtigkeit des Codes. Kein Modell erreicht eine 50 %-Richtigkeitsrate. Das beste Modell ist GPT-4o mit 24 richtigen Antworten, und das schlechteste ist CodeLlama-7B ohne richtige Antwort.

Als Nächstes betrachten wir die fehlenden Codeblöcke, die in Abbildung 13 zu sehen sind. Dieses Diagramm zeigt:

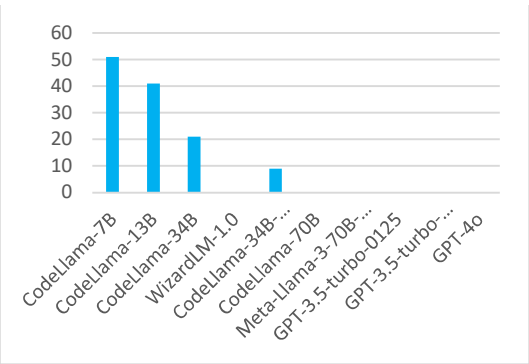


*Abbildung 13: Statistik Fehlender Code*

Kleinere Modelle generieren häufiger keine korrekten Codeblöcke. Auf der 34B-Größe ist das WizardLM-Modell das einzige, das keine fehlenden Codeblöcke aufweist. Große Modelle vermeiden es vollständig, den Instruktionen nicht zu folgen.

Als Nächstes betrachten wir das Diagramm für die Fehler im Code, das in Abbildung 14 genutzt wird:

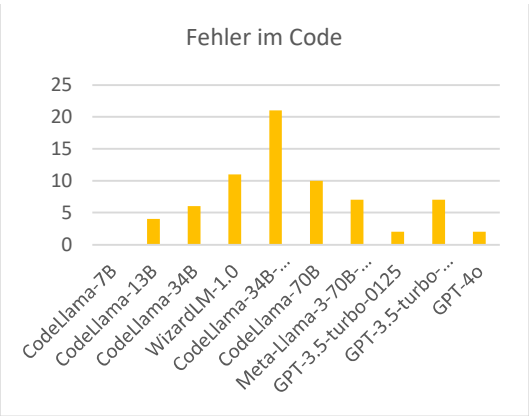


*Abbildung 14: Statistik Fehler im Code*

Isoliert betrachtet verursachen die 34B-Modelle die meisten Fehler. Das 34B-Instruct-Modell hat die höchste Fehleranzahl.

Wenn man die fehlenden Codeblöcke und die Fehler zusammenrechnet, siehe Abbildung 15, liegen die kleineren Modelle immer noch vorne. Es ist jedoch zu beachten, dass das Instruct-Modell auf 34B-Level sowie das GPT-3.5-Instruct-Modell eine höhere Fehleranzahl aufweisen als ihre vergleichbaren Chat-Modelle.

Um zu sehen, wie genau die Modelle mit ihrem Code sind, vergleichen wir die falschen Antworten mit allen fehlerfreien Antworten (richtige und falsche Antworten). Siehe Abbildung 16, dort sieht man:

Kleinere Modelle haben mehr falsche Antworten. Instruct-Modelle derselben Version haben mehr falsche Antworten. Neuere Modelle haben weniger falsche Antworten. Das beste Modell ist GPT-4o, bei dem nur 51 % der Antworten falsch sind.

Zum Schluss betrachten wir noch den Vergleich aller Antworten pro Modell, siehe Abbildung 17.

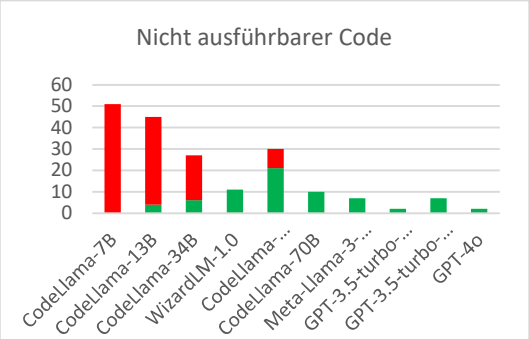

*Abbildung 15: Statistik nicht ausführbarer Code*

Es gibt eine klare Verbesserung in Abhängigkeit von der Modellgröße. Das WizardLM-Modell sticht unter den unteren fünf Modellen hervor. Das Meta-Llama-3-Modell hat ein ähnliches Ergebnis wie das GPT-3.5-Modell, macht jedoch mehr Fehler im Code. Ein wichtiger Vergleich ist der zwischen den 34B-Modellen. Sie haben alle nur eine geringe Menge richtiger Antworten, aber die Aufteilung der Fehlergründe ist stark unterschiedlich. Während das WizardLM-Modell den meisten ausführbaren Code generiert hat, hat das Instruct-Modell die wenigsten ausführbaren Codestücke generiert. Wenn man das Standard-34B-Modell als Basis nimmt, ist der Unterschied deutlich erkennbar.

## 5.3. Vergleich der Fälle

Um die Fähigkeiten der Modelle genauer zu analysieren, können einzelne Testfälle betrachtet werden:

Der am häufigsten richtig beantwortete Testfall ist Fall 9 mit 8 von 10 richtigen Antworten. Auf dem zweiten Platz stehen die Fälle 35, 42 und 51 mit jeweils 6 von 10 richtigen Antworten, gefolgt von den Fällen 1 und 44 mit jeweils 5 von 10 richtigen Antworten. Alle anderen Fälle haben weniger als 5 von 10 richtigen Antworten. Die Fälle 3, 4, 6, 10, 16, 17, 19, 20, 30, 31, 32, 33, 34, 36, 38, 43, 45, 46 und 48 haben keine richtige Antwort geliefert. Das bedeutet, dass 19 der Queries unbeantwortet blieben, was 37 % der Queries entspricht. Der Fall mit dem wenigsten Code und damit möglicherweise der einfachste war Fall 9, bei dem lediglich Katzen gezählt wurden.

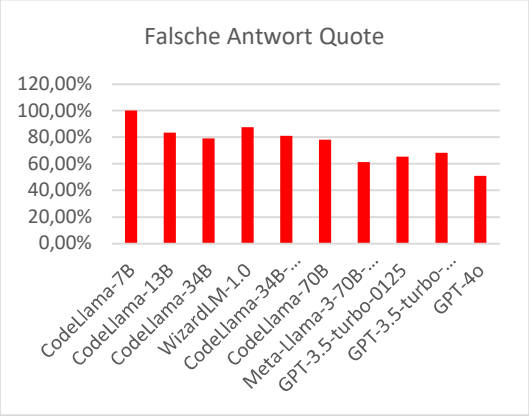

*Abbildung 16: Statistik Falsche Antwort Quote*

Es gab dazu noch auffällige und besondere Fälle:

Fall 1: Die Antworten sind inkonsistent, abhängig von der Codequalität.

Fall 2: Jede Antwort, außer von GPT-4o, war leer.

Fälle 3-8: Die CV-Modelle konnten in diesem Bild nicht zwischen Kindern und Lehrern unterscheiden. Daher gaben alle Modelle die gleiche falsche Antwort von 6

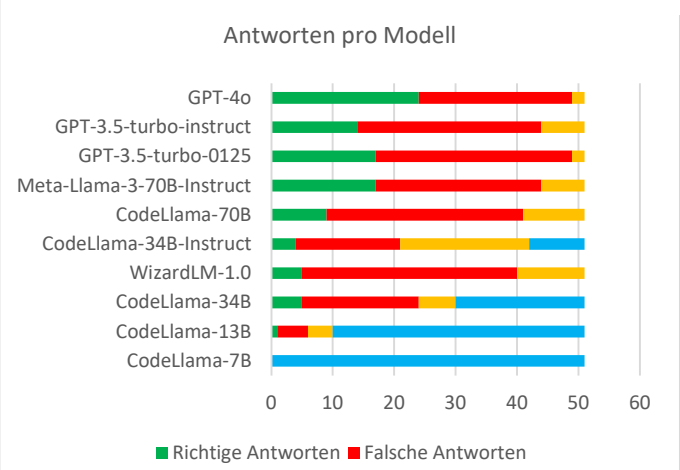

*Abbildung 17: Antworten pro Modell*

Personen, da 6 Personen auf dem Bild zu sehen waren. Dies wirkte sich auch auf die folgenden 5 Fälle aus, bei denen die Anzahl der Kinder wichtig war. Deshalb konnte der Einfluss der Wortwahl für diese Fälle leider nicht analysiert werden.

Fall 9: Dies war der Fall mit den meisten richtigen Antworten. Jedes Modell, außer dem 7B- und 13B-Modell, lieferte hier eine korrekte Antwort.

Fall 12: Dieser Testfall war der am wenigsten ausgeführte. Nur drei Modelle hatten bei diesem Fall keine Fehler. Die nachfolgenden drei Fälle bieten Einblick, ob die Wortwahl einen Einfluss hat, und ja, Fall 15 ist der fehlerfreiste Fall, zusammen mit Fall 43.

Fall 16: Die Personenerkennung war auch hier problematisch, da zwei Lehrer erkannt wurden, obwohl nur einer vorhanden war.

Fall 18: Bei den Pizzen wurden häufiger Oliven als jeder andere Belag erkannt, möglicherweise aufgrund der grünen Farbe.

Fälle 19 und 20: Katzen konnten im Fall 9 sehr gut erkannt werden, aber hier scheiterten die Modelle an den verschiedenen Tieren, was größtenteils auf die CV-Modelle zurückzuführen ist.

Fall 21: Es gab zwei Modelle, die eine Antwort geben konnten, aber nur GPT-4o lieferte eine korrekte Antwort.

Fall 22: Der Autovergleich gelang nur den größeren Modellen, was immer an der Nutzung der API oder Syntaxproblemen lag.

Fälle 23-25: Die Spezialisierung auf die Tiergruppe scheint hier keinen Einfluss gehabt zu haben, da alle drei theoretisch eine richtige Antwort hatten. Interessanterweise erkannten die CV-Modelle die Tiere oft als „Wallpaper“, was zu einigen falschen Antworten führte.

Fall 29: Die Worterkennung funktionierte bei korrekt generiertem Code sehr gut.

Fälle 30-33: Keiner dieser Fälle konnte beantwortet werden, und die Ergebnisse waren oft sehr zufällig. Möglicherweise verursachte die Textunterbrechung Probleme.

Fälle 34 und 35: Nur nach einem Fehler zu fragen war zu vage. Die Codestücke waren interessant, da sie eher mit falschen Objekten gerechnet hatten. Sobald jedoch klar war, dass es sich um einen Rechtschreibfehler handelte, konnten 6 Modelle die richtige Antwort liefern.

Fälle 36 und 37: Dies war der Test, der die Texterkennung herausfordern sollte, und das Ergebnis war erwartungsgemäß: Kein Modell konnte diese Fälle lösen, und nur durch das zufällige richtige Schreiben eines Buchtitels lieferte GPT-3.5 die korrekte Antwort. Das Problem hier lag, wie zuvor, an den Einschränkungen der CV-Modelle.

Fälle 38-42: Die Frage nach dem Fach war beim ersten Bild für die Modelle nicht beantwortbar, beim zweiten Bild war das besser. Die Hinzufügung des Wortes „Schule“ führte zu mehr richtigen Antworten.

Fall 43: Eine Anfrage, das Bild zu beschreiben, führte zu sehr variierenden Antworten. Entweder gab es API-Probleme oder es wurden erfundene Begriffe verwendet.

Fall 44: Das Abfragen der Anzahl von Charakteren funktionierte hier wiederum sehr gut.

Fälle 45-47: Nur GPT-4o konnte, nachdem die Charaktere genannt wurden, die richtige Antwort liefern.

Fälle 48-51: Anfangs konnte kein richtiger Code generiert werden. Erst als Jesus erwähnt wurde, wurde die Frage richtig beantwortet. Die Ausnahme war GPT-4o, das bereits bei der zweiten Frage eine korrekte Antwort liefern konnte.

## 5.4. Vergleich mit dem Paper

Die verschiedenen Testfälle haben gezeigt, dass nicht alle im Paper aufgestellten Erwartungen erfüllt werden konnten.

Von den erwarteten Fällen (1, 2, 12, 13, 14, 15, 21, 26, 27) wurde jeder mindestens einmal richtig beantwortet, wobei fast alle vom besten Modell, GPT-4o, korrekt beantwortet wurden.

Von der zweiten Kategorie der Fälle (3, 4, 5, 6, 7, 8, 11, 16, 17, 19, 20, 22, 23, 24, 25, 48, 49, 50, 51) wurden die Fälle 3, 4, 6, 16, 17, 19, 20 und 48 nicht einmal richtig beantwortet, während die anderen Fälle mindestens einmal korrekt beantwortet wurden.

In der letzten Kategorie der Fälle (29, 32, 33, 34, 35, 38, 39, 40, 41, 42) wurden die Fälle 32, 33, 34 und 38 nicht richtig beantwortet. Die Gründe dafür wurden im vorherigen Abschnitt erläutert.

Betrachten wir die Fähigkeiten des Modells und ob es diese erreicht hat:

- Logisches Denken: Definitiv. Die größeren LLMs haben logischen und gut strukturierten Code generiert.
- Räumliches Verständnis: Fälle mit dieser Fähigkeit konnten gelöst werden, allerdings waren die Einschränkungen der Objekterkennung manchmal problematisch.
- Wissen von LLMs: Wissensbasierte Fragen konnten von den großen Modellen sehr gut beantwortet werden, allerdings nicht immer zu 100 %.
- Konsistenz in den Antworten: Für denselben Code gab es immer dasselbe Ergebnis, aber bei mehreren Codegenerierungsanfragen resultierten unterschiedliche Antworten.
- Mathematische Fähigkeiten: Mathematische Fragen wurden nachvollziehbar beantwortet, abgesehen von den Einschränkungen im Code.
- Objektinterpretation / Objekattribute: Hier gab es die meisten Probleme. Die CV-Modelle haben das gesamte Projekt beeinträchtigt, insbesondere bei der Personen- oder Tiererkennung, was möglicherweise an der Größe des Blip-Modells lag.
- Relationales Denken: Die Fälle, die sich damit befassten, konnten von kleineren Modellen weniger gut, aber von den größeren Modellen meist richtig gelöst werden.

## 5.5. Vergleich mit menschlichen Testpersonen

Der Vergleich mit den menschlichen Testpersonen hat ebenfalls viele interessante Erkenntnisse geliefert. Zunächst betrachten wir die richtigen Antworten (siehe Abbildung 18). Diese zeigt, dass 86 % der Antworten korrekt gegeben wurden, was 67 % mehr richtige Antworten sind als die der KI-Modelle und 37 % mehr als das beste LLM des Projekts.

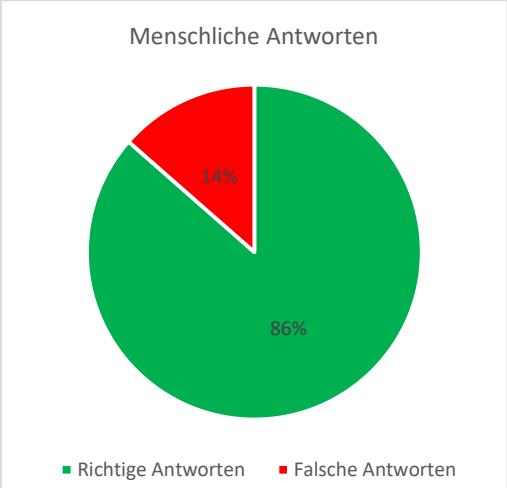


*Abbildung 18: Statistik Menschline Antworten*

Die Abbildung 19 zeigt, dass keiner der Testfälle unbeantwortet blieb und sogar 25 der Testfälle von allen Personen korrekt beantwortet wurden. Dies war den LLMs nicht gelungen. Der einzige Fall, der nur von einer Person beantwortet werden konnte, war Fall 21 über den Wettbewerb zwischen dem Empire State und dem Chrysler Building.

Weitere interessante Fälle:

Fall 10: Alle waren sich unsicher, ob eine Katze steht oder sitzt.

Fall 16: Das Verhältnis von Kindern zu Lehrern war nicht für alle ganz schlüssig, und einige haben sich verzählt.

Fälle 19 und 20: Einige Personen waren unsicher, ob die Vögel als eigene Art gelten und ob ein Schmetterling als Tier zählt. Einige haben vergessen, den Hasen zu zählen.

Fall 21: GPT-4o lieferte hier eine bessere Antwort als jede der getesteten Personen.

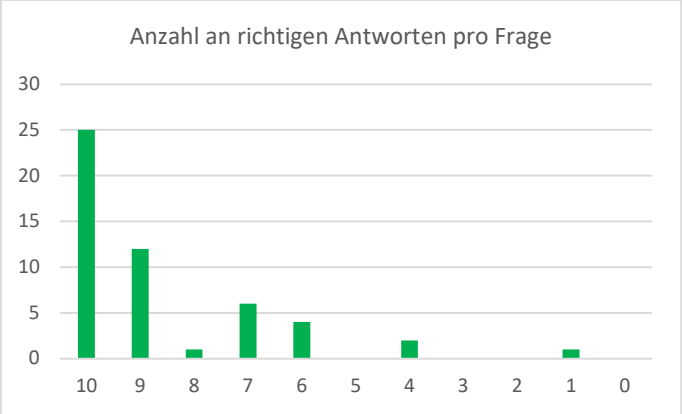


*Abbildung 19: Statistik Anzahl an richtigen Antworten pro Frage*

Fall 23: Hier handelte es sich oft um eine Schätzung, und das Tier wurde oft falsch erkannt. Da es schwer zu sagen war, ob das mittlere Tier ein Jaguar, Leopard oder Gepard ist, wurden alle drei Antworten akzeptiert (selbst die Bildquelle war sich nicht sicher).

Fälle 30-33: Alle Personen konnten den Text erkennen, auch wenn nicht jeder Turkish Airlines kannte, was die Erkennung jedoch nicht behinderte.

Fälle 34 und 35: Ohne die Information, dass es sich um einen Rechtschreibfehler handelt, waren die Antworten weniger korrekt.

Fälle 36 und 37: Es war nicht immer offensichtlich, ob es sich um die Länge des Textes oder die Anzahl der Bücher handelte. Außerdem gab es Probleme bei der Erkennung der Titel.

Fälle 43-45: Die Bildbeschreibung war bei den jüngeren Personen sehr detailliert, während die älteren Personen sich kürzer fassten und manches nicht richtig erkannten. Es gab eine klare Trennung zwischen Personen, die die Charaktere kannten, und solchen, die sie nicht kannten. Nur 2 Personen hatten Wissen über das Alter der Charaktere, der Rest schätzte.

Fälle 48-51: Die Antworten wurden sehr schnell gegeben, was bei den Jüngeren wahrscheinlich daran lag, dass sie Pfadfinder waren.

Die Personen selbst zeigten keine großen Unterschiede (Abbildung 20), nur Person 9 schnitt etwas schlechter ab, während die Personen 2, 4, 5 und 6 gleichauf vorne lagen. Einige auffällige Unterschiede:

Person 1 verfügte über weniger Wissen auf Grund des alters. Person 9 hatte Probleme mit Farbtheorie. Person 5 drückte sich detaillierter aus.

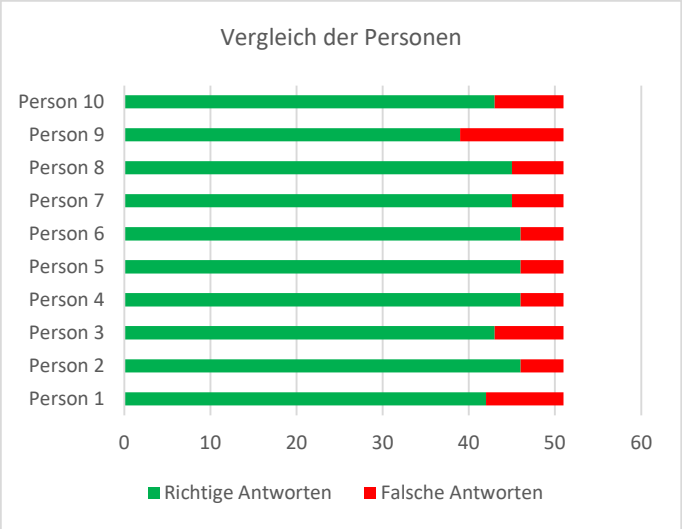


*Abbildung 20: Statistik Vergleich der Personen*

# 6. Bewertung der Ergebnisse

Das ViperGPT-Projekt hat einige interessante Erkenntnisse hervorgebracht, die im Vergleich zwischen Menschen und KI-Modellen deutlich wurden. Menschen übertreffen KIs im Allgemeinen, insbesondere in ihrer Fähigkeit, komplexe Aufgaben zu bewältigen. Sprachmodelle (LLMs) sind jedoch sehr gut darin, Wissensfragen präzise zu beantworten, während menschliche Testpersonen oft Schwierigkeiten bei der Interpretation von Fragen hatten. Es zeigte sich, dass die Wortwahl der Anfragen einen erheblichen Einfluss auf die Ergebnisse der LLMs hat – je detaillierter die Anfragen formuliert sind, desto besser fallen die Ergebnisse aus.

Obwohl KIs viele Fähigkeiten beherrschen, stoßen sie bei detaillierten und spezialisierten Anfragen weiterhin an ihre Grenzen. Besonders problematisch bleiben Anfragen in Nischenthemen, bei denen die LLMs oft nicht zufriedenstellend reagieren. Computer Vision-Modelle (CV-Modelle), die im ViperGPT-Projekt eingesetzt wurden, hatten Schwierigkeiten, fiktive Charaktere korrekt zu identifizieren und stellten somit eine generelle Einschränkung für das Projekt dar. Es wurde auch deutlich, dass neuere LLMs signifikant bessere Leistungen erbringen als ältere Versionen. Instruct-Modelle hingegen hatten oft größere Probleme, den gegebenen Anweisungen genau zu folgen. Insgesamt wurden die Erwartungen an das ViperGPT-Projekt leider nicht erfüllt, da die tatsächliche Erfolgsquote von 19 % deutlich unter der erwarteten Quote von 40 % lag und somit weniger als die Hälfte des angestrebten Ergebnisses erreicht wurde.

# 7. Klärung der Forschungsfragen

## 7.1. Wie funktioniert ein solches KI-Werkzeug?

Diese Frage lässt sich anhand des technischen Überblicks aus Abschnitt 3 beantworten, hier jedoch noch einmal zusammengefasst: ViperGPT nutzt eine vorprogrammierte API, die verschiedene Funktionen zur Ansteuerung mehrerer CV-Modelle sowie LLMs bereitstellt, um Bilder zu analysieren und Fragen zu beantworten. Die Dokumentation dieser API wird zusammen mit der Anfrage (Query) an ein LLM gesendet, das daraufhin den ausführbaren Code generiert, der für die Bildanalyse genutzt wird. Während der Ausführung werden die API-Funktionen und somit die CV-Modelle angesteuert. Das Endergebnis sollte die gestellte Frage beantworten, abhängig davon, wie gut der generierte Code und die verwendeten CV-Modelle sind.

Im gesamten Prozess sind alle Modelle austauschbar, sodass bei Verfügbarkeit neuer Modelle diese einfach integriert und genutzt werden können. Diese Modularität ist die wichtigste Funktion des ViperGPT-Projekts, da sie es ermöglicht, auch zukünftige Fortschritte in der KI-Entwicklung zu nutzen.

### 7.2. Werden Softwareentwickler durch die Inkorporation von KI obsolet?

Kurz gesagt: nein. Das Projekt und die damit genutzten KIs sind derzeit noch unzuverlässig. Obwohl neuere KIs immer leistungsfähiger werden, kann niemals garantiert werden, dass alles korrekt ist. Daher bleibt die Rolle eines Softwareentwicklers unerlässlich, um die Ausgaben der KI zu überprüfen. Allgemein sind die KIs noch nicht in der Lage, jedes Problem zu lösen, was bedeutet, dass in komplexen und anspruchsvollen Fällen weiterhin Softwareentwickler benötigt werden. Die Ausbildung sollte sich daher weniger auf das Schreiben von Code konzentrieren, sondern vielmehr auf das Verständnis der zugrunde liegenden Konzepte. Debugging, Unit-Testing und Strukturierung werden zu den wichtigsten Aufgaben gehören.

### 7.3. Ist eine Ausbildung in der grafischen Datenverarbeitung weiterhin notwendig?

Aus meiner Perspektive wäre es ohne die Ausbildung im Fach Grafische Datenverarbeitung nicht möglich gewesen, die Fähigkeiten der Modelle adäquat zu analysieren. Viele der Erkenntnisse, die aus den Ergebnissen gewonnen wurden, stützen sich auf das Wissen aus diesem Fachgebiet. Insbesondere um die Fähigkeiten von CV-Modellen zu verstehen, ist eine entsprechende Ausbildung unerlässlich.

## 8. Fazit und Zukunftsaussicht

Das Projekt war in vielerlei Hinsicht sehr interessant und hat großen Spaß gemacht, insbesondere bei der Analyse und Erweiterung der Modelle. Anfangs wurden höhere Erwartungen an die Fähigkeiten des Projekts gestellt, doch es wurde deutlich, dass noch einige Limitationen bestehen. Besonders die CV-Modelle müssen erheblich verbessert werden, da sie derzeit die größte Einschränkung für das Projekt darstellen. Trotz dieser Herausforderungen ist das Konzept zukunftsfähig, und Menschen übertreffen KIs weiterhin in vielen Bereichen. Insgesamt war dieses Projekt äußerst wertvoll, um ein besseres Verständnis dafür zu entwickeln, wie moderne KIs mit Bildern und Videos umgehen.

Künstliche Intelligenz wird sich kontinuierlich weiterentwickeln und wird meiner Einschätzung nach irgendwann die Fähigkeiten des Menschen übertreffen. Das Projekt könnte in Zukunft weitergeführt werden, auch wenn einige Modelle bereits integrierte Bildanalysefunktionen bieten. Es werden immer größere Modelle trainiert und ihre Fähigkeiten weiter ausgebaut. Die Nachfrage nach unzensierten großen Modellen wird voraussichtlich weiter bestehen, jedoch dürfte dies vor allem im Open-Source-Bereich relevant bleiben. Der Beruf des Softwareentwicklers bleibt weiterhin sinnvoll, auch wenn der Bedarf an Entwicklern möglicherweise geringer sein wird als zuvor.

## 9. Anhang

### 9.1. KI-Nutzung und Prompts

Im Verlauf der Thesis wurden zahlreiche Modelle für die Tests verwendet, aber auch beim Schreiben und Korrigieren kamen KIs zum Einsatz.

QuillBot wurde genutzt, um die Grammatik nachträglich zu überprüfen.

ChatGPT (GPT-4o) wurde verwendet, um den endgültigen Text zu formatieren und zu korrigieren. Dies bedeutet, dass jedes Prüfverfahren diesen Text möglicherweise als durch eine KI verfasst erkennt. Nach Rücksprache mit meinem Betreuer war dies jedoch legitim. Insbesondere wurde ChatGPT genutzt, um Stichpunkte in zusammenhängende Texte zu konvertieren und Übergänge zwischen einzelnen Textpassagen zu generieren. Das Abstract und die Einleitung wurden weitgehend früher erstellt als die anderen Textpassagen; leider sind diese in meinem Verlauf nicht mehr auffindbar und konnten daher nicht gelistet werden. Alle Texte basieren auf dem zuvor erstellten Dokument „Draft/Zusammengesetzte_Thesis.docx".

Die Formatierungsprompts, die für die Korrekturen verwendet wurden, lauteten wie folgt:

Erster Prompt:
„Handeln Sie als erfahrener akademischer Schreiber. Überprüfen Sie den bereitgestellten Text sorgfältig auf Grammatik-, Interpunktions- und Syntaxfehler. Korrigieren Sie diese Fehler, während Sie die ursprüngliche Bedeutung beibehalten. Konvertieren Sie Stichpunkte, Auflistungen und freistehende Sätze in zusammenhängende Sätze und in einen fortlaufenden Text. Stellen Sie sicher, dass der Text akademisch, klar, präzise und gut strukturiert ist. Stellen Sie sicher, dass die endgültige Version poliert und fehlerfrei ist.
[Text][Text]"

Zweiter Prompt:
„Handeln Sie als erfahrener akademischer Schreiber. Ihnen werden zwei Textstellen übergeben, und Sie müssen einen Übergang zwischen diesen schreiben. Für diesen Übergang werden, mit [Übergang] markiert, Stichpunkte gegeben, die in den Text integriert werden müssen. Stellen Sie sicher, dass der Text akademisch, klar, präzise und gut strukturiert ist. Stellen Sie sicher, dass die endgültige Version poliert und fehlerfrei ist. Die beiden Textstellen beginnen jeweils mit [Text1] und [Text2].
[Übergang][Übergang][Text1][Text1][Text2][Text2]"

Der endgültige Formatierungsverlauf ist unter folgenden Links zu finden:

https://chatgpt.com/share/5b6c1286-7ccc-48f3-9314-90bea81794ba

https://chatgpt.com/share/dc321bdd-a1ce-4e56-bc6f-de779f97fd94

https://chatgpt.com/share/c64298cf-1d89-4231-9142-4bf6f078cbc9

## 9.2. Literatur- und Quellenverzeichnis


[1] S. Russell und P. Norvig, „Artificial Intelligence, Global Edition". Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://elibrary.pearson.de/book/99.150005/9781292401171

[2] J. Schmidhuber, „Annotated History of Modern AI and Deep Learning", 29. Dezember 2022, *arXiv*: arXiv:2212.11279. doi: 10.48550/arXiv.2212.11279.

[3] E. A. Feigenbaum und J. Feldman, *Computers and thought*. New York, McGraw-Hill, 1963. Zugegriffen: 31. Juli 2024. [Online]. Verfügbar unter: http://archive.org/details/computersthought00feig

[4] D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*. 1993, S. 386.

[5] M. Minsky und S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 2017. doi: 10.7551/mitpress/11301.001.0001.

[6] „Chilton::INF::Lighthill Report". Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://www.chilton-computing.org.uk/inf/literature/reports/lighthill_report/p001.htm

[7] L. Earnest, „Stanfort Cart", Stanford Cart. Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://web.stanford.edu/~learnest/sail/oldcart.html

[8] D. E. Rumelhart, J. L. McClelland, und S. D. P. R. G. University of California, *Parallel distributed processing : explorations in the microstructure of cognition*. Cambridge, Mass. : MIT Press, 1986. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: http://archive.org/details/paralleldistribu00rume
[9] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory“, *Neural Comput.*, Bd. 9, Nr. 8, S. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
[10] A. Krizhevsky, I. Sutskever, und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
[11] „Top scientists call for caution over artificial intelligence“, The Telegraph. Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://www.telegraph.co.uk/technology/news/11342200/Top-scientists-call-for-caution-over-artificial-intelligence.html
[12] A. Vaswani *u. a.*, „Attention Is All You Need“, 1. August 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
[13] A. N. Ph.D, „The brief history of Large Language Models: A Journey from ELIZA to GPT-4 and Google Bard“, Medium. Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://levelup.gitconnected.com/the-brief-history-of-large-language-models-a-journey-from-eliza-to-gpt-4-and-google-bard-167c614af5af
[14] K. Hu und K. Hu, „ChatGPT sets record for fastest-growing user base - analyst note“, *Reuters*, 2. Februar 2023. Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/
[15] E. Yan, *eugeneyan/open-llms*. (1. August 2024). Zugegriffen: 1. August 2024. [Online]. Verfügbar unter: https://github.com/eugeneyan/open-llms
[16] V. Wiley und T. Lucas, „Computer Vision and Image Processing: A Paper Review“, *Int. J. Artif. Intell. Res.*, Bd. 2, S. 22, Feb. 2018, doi: 10.29099/ijair.v2i1.42.
[17] „Maschinelles Lernen: Kompetenzen, Forschung, Anwendung“, Fraunhofer-Allianz Big Data<br>und Künstliche Intelligenz. Zugegriffen: 20. August 2024. [Online]. Verfügbar unter: https://www.bigdata-ai.fraunhofer.de/de/publikationen/ml-studie.html
[18] K. R. Chowdhary, „Natural Language Processing“, in *Fundamentals of Artificial Intelligence*, K. R. Chowdhary, Hrsg., New Delhi: Springer India, 2020, S. 603–649. doi: 10.1007/978-81-322-3972-7_19.
[19] Y. LeCun, Y. Bengio, und G. Hinton, „Deep learning“, *Nature*, Bd. 521, Nr. 7553, S. 436–444, Mai 2015, doi: 10.1038/nature14539.
[20] D. E. Rumelhart, G. E. Hinton, und R. J. Williams, „Learning representations by back-propagating errors“, *Nature*, Bd. 323, Nr. 6088, S. 533–536, Okt. 1986, doi: 10.1038/323533a0.
[21] „Better language models and their implications“. Zugegriffen: 20. August 2024. [Online]. Verfügbar unter: https://openai.com/index/better-language-models/
[22] A. Agrawal *u. a.*, „VQA: Visual Question Answering“, 26. Oktober 2016, *arXiv*: arXiv:1505.00468. doi: 10.48550/arXiv.1505.00468.
[23] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, und D. Parikh, „Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering“, 15. Mai 2017, *arXiv*: arXiv:1612.00837. doi: 10.48550/arXiv.1612.00837.
[24] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, und R. Girshick, „CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning“, 20. Dezember 2016, *arXiv*: arXiv:1612.06890. doi: 10.48550/arXiv.1612.06890.
[25] „What Is ChatGPT Doing … and Why Does It Work?“ Zugegriffen: 21. August 2024. [Online]. Verfügbar unter: https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

[26] T. B. Brown *u. a.*, „Language Models are Few-Shot Learners“, 22. Juli 2020, *arXiv*: arXiv:2005.14165. doi: 10.48550/arXiv.2005.14165.
[27] „ LM Studio - Discover and run local LLMs“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://lmstudio.ai
[28] „Visual Studio Code - Code Editing. Redefined“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://code.visualstudio.com/
[29] „Welcome to Python.org“, Python.org. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://www.python.org/
[30] „Anaconda | The Operating System for AI“, Anaconda. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://www.anaconda.com/
[31] D. Surís, S. Menon, und C. Vondrick, „ViperGPT: Visual Inference via Python Execution for Reasoning“, 14. März 2023, *arXiv*: arXiv:2303.08128. doi: 10.48550/arXiv.2303.08128.
[32] „ViperGPT: Visual Inference via Python Execution for Reasoning“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://viper.cs.columbia.edu/
[33] „Pricing“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://openai.com/api/pricing/
[34] „TheBloke/CodeLlama-7B-GGUF · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/TheBloke/CodeLlama-7B-GGUF
[35] „TheBloke/CodeLlama-13B-GGUF · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/TheBloke/CodeLlama-13B-GGUF
[36] „TheBloke/CodeLlama-34B-GGUF · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/TheBloke/CodeLlama-34B-GGUF
[37] „codellama/CodeLlama-70b-hf · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/codellama/CodeLlama-70b-hf
[38] „TheBloke/CodeLlama-34B-Instruct-GGUF · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/TheBloke/CodeLlama-34B-Instruct-GGUF
[39] „TheBloke/WizardLM-1.0-Uncensored-CodeLlama-34B-GGUF · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/TheBloke/WizardLM-1.0-Uncensored-CodeLlama-34B-GGUF
[40] „meta-llama/CodeLlama-70b-Instruct-hf · Hugging Face“. Zugegriffen: 30. August 2024. [Online]. Verfügbar unter: https://huggingface.co/meta-llama/CodeLlama-70b-Instruct-hf

## 9.3. Bildquellen

Alle Statistiken wurden aus den Ergebnissen der Tests erstellt und sind in der „Ergebnisse AI.xlsx“ sowie der „Ergebnisse Mensch.xlsx“ zu finden. Alle Testbilder wurden mit links in der QuelleBilder.txt festgehalten. Die Bilder des orginalem und verändertem Ablaufs wurden mit draw.io erstellt.

## 9.4. Hilfsmittelverzeichnis

Zotero: Für Zitierung und Quellenmanagement
https://www.zotero.org/

ChatGPT (GPT 4o): Für Korrektur, Formatierung und Generierung von Text
https://chatgpt.com/

QuillBot: Zur Rechtschreibprüfung
https://www.zotero.org/

Draw.io: Für Flussdiagramme
https://app.diagrams.net/

Excel: Für Statistik und Auswertung
https://www.microsoft.com/de-de/microsoft-365/excel?market=de

VS-Code: Als IDE
https://code.visualstudio.com/

Python: Zur automatischen Ergebnis Evaluierung
https://www.python.org/

## 9.5. Eigenständigkeitserklärung

“Versicherung über redliches wissenschaftliches Arbeiten“

Hiermit versichere ich, [Vorname Nachname], dass ich die vorliegende Arbeit selbstständig verfasst und erstellt habe. Ich versichere, dass ich nur zugelassene Hilfsmittel und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ferner versichere ich, dass ich alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gemäß gängiger wissenschaftlicher Zitierregeln korrekt zitiert und als solche gekennzeichnet habe. Darüber hinaus versichere ich, dass alle verwendeten Hilfsmittel, wie KI-basierte Chatbots (bspw. ChatGPT), Übersetzungs- (bspw. Deepl), Paraphrasier- (bspw. Quillbot) oder Programmier-Applikationen (bspw. Github Copilot) vollumfänglich deklariert und ihre Verwendung an den entsprechenden Stellen angegeben und gekennzeichnet habe.

Ich bin mir bewusst, dass die Nutzung maschinell generierter Texte keine Garantie für die Qualität von Inhalten und Text gewährleistet. Ich versichere, dass ich mich textgenerierender KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Textpassagen vollumfänglich selbst.

Auch versichere ich, die „Satzung der Hochschule Furtwangen (HFU) zur Sicherung guter wissenschaftlicher Praxis“ vom 27. Oktober 2022 zur Kenntnis genommen zu haben und mich an den dortigen Ausführungen zu orientieren.

Mir ist bewusst, dass meine Arbeit auf die Benutzung nicht zugelassener Hilfsmittel oder Plagiate überprüft werden kann. Auch habe ich zur Kenntnis genommen, dass ein Verstoß gegen § 10 bzw. § 11 Absatz 4 und 5 der Allgemeinen Teile der HFU-SPOen zu einer Bewertung der betroffenen Arbeit mit der Note 5 oder mit «nicht ausreichend» und/oder zum Ausschluss von der Erbringung aller weiteren Prüfungsleistungen führen kann.

Ort, Datum Unterschrift