Projektarbeit

im Fach künstliche Intelligenz

zur Erlangung des Grades eines

Bachelor of Science (B.Sc.)

über das Thema

Wissensbasierte Klassifikation von ITSM-Tickets mittels Deep-Learning

von

Tom Schich und Michael Ruhrig

Datum der Abgabe: 30.01.2026

Anzahl Wörter: 8.785 Wörter

Inhaltsverzeichnis

Seite

Abbildungsverzeichnis

Seite

Tabellenverzeichnis

Seite

Abkürzungsverzeichnis

[CLS]-Token	Classification-Token
CNN	Convolutional Neural Network
CMMI-SVC	Capability Maturity Model Integration Services
COBIT	Control Objectives for Information and Related Technologies
DNN	Deep Neural Networks
FitSM	Federated IT Service Management
GPQA	Graduate -Level Google-Proof Question and Anwser
HDI	Haftpflichtverband der Deutschen Industrie
HumanEval	Handwritten Evaluationset
IEC	International Electrotechnical Commission
ISO	International Organisation for Standardisation
ITIL	Information Technology Infrastructure Libraries
ITSM	Information Technology Service Management
KI	Künstliche Intelligenz
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Maschinelles Lernen
MMLU	Massive Multitask Language Understanding
NLM	Natural Language Model
NLP	Natural Language Processing
RegEx	Regular Expressions
RNN	Recurrent Neural Networks
SLA	Service Level Agreements

SVM	Support Vector Machines

1. Einleitung

In einem sich stetig wandelnden technologischen Umfeld sind die Anforderungen an Support-Strukturen heute komplexer denn je. Vor diesem Hintergrund wird die aktuelle Situation der Technik beleuchtet, um die Relevanz und den Fokus der vorliegenden Arbeit einzuordnen.

1.1. Problemstellung

In großen Unternehmen entstehen täglich hunderte bis tausende IT-Supportanfragen, deren schnelle und präzise Bearbeitung den reibungslosen Ablauf von Geschäftsprozessen ermöglicht. Damit diese Vielzahl von Anfragen effektiv und effizient verarbeitet werden können, müssen Störungen („Incidents“) und Dienstleistungsanfragen („Service Requests“) zuverlässig klassifiziert und priorisiert werden. Im Frühjahr 2024 berichtet der HDI in einer Studie bei einer Umfrage der IT-Branche, dass viele Gefragte angaben ein steigendes Ticketaufkommen bemerkt zu haben. [1] Die Klassifizierung und Zuweisung von Tickets an die zuständigen Support-Teams (z.B. Service Desk, technischer Support oder Lieferanten) ist zeitaufwendig, fehleranfällig und angesichts wachsender Ticketvolumina zunehmend schwererer skalierbar. [2] Daraus lässt sich schließen, dass in der Praxis die Bewältigung der Menge und Vielfalt eingehender Tickets bei begrenzten Ressourcen ein zentraler Engpass im IT-Support ist.

Anfragen über Tickets reichen von einfachen Anwenderfragen bis hin zu schwerwiegenden Serviceunterbrechungen/Störungen. Wenn gleichzeitig nur eine begrenzte Anzahl an Bearbeitern mit umfangreichem Domänenwissen zur Verfügung steht, ist eine systematische und nachvollziehbare Strategie zur Abarbeitung erforderlich. [3]

Um vorhandene Ressourcen effizient einzusetzen, müssen eingehende Tickets nach objektiven Kriterien klassifiziert werden. Diese Klassifizierung dient dazu, Art und Umfang einer Anfrage zu bestimmen und ihre geschäftliche Relevanz einzuschätzen. So können zeit- oder geschäftskritische Störungen priorisiert und schneller bearbeitet werden, während weniger dringliche Anliegen nachrangig behandelt werden.

---

1 Vgl. *HDI*, State of Technical Support, 2024, *Zugriff am 28.12.2025*
2 Vgl. *Yu, Q.*, Artificial Intelligence to enhance IT Operations, 2024, S. 2.
3 Vgl. *Pfitzinger, B.*, IT-Betrieb, 2016, S. 452–454.

Best Practices-Frameworks wie ITIL 4, CMMI-SVC oder ISO/IEC 20000 bieten strukturierte Vorgehensweisen für die Klassifizierung und Priorisierung solcher Tickets. Diese Ansätze verfolgen das gleiche Ziel, begrenzte Ressourcen effizient einzusetzen, geschäftskritische Störungen frühzeitig zu erkennen und die Servicequalität durch konsistente Prozesse sicherzustellen. Aus einem wissenschaftlichen Fachartikel des Journal of Network and Computer Applications geht hervor, dass bislang kein automatisiertes Incident-Management-Framework existiert, das den gesamten Prozess von Ticket-Generierung bis Incident-Resolution abdeckt. Dies verdeutlicht die Forschungslücke, die moderne KI-Methoden, insbesondere Deep-Learning-basierte Klassifizierungsmodelle, schließen sollen. [4] Eine solche Lücke stellt das automatisierte Herleiten von Lösungen für Incidents dar.

Eine beispielhafte Bearbeitung eines Incidents könnte sein, anhand der Recherche vergangener gelöster Vorgänge eine Störung, oder um Ressourcen zu schonen, einen darauf passenden Workaround zu identifizieren. Incidents mit größeren oder bisher unbekannten Auswirkungen erfordern ein komplexeres Management und mehr Personal. [5]

Um eine Differenzierung für das ressourcenschonende Zuweisen an die korrekten Bearbeiter zu erleichtern, ist eine korrekte Priorisierung unerlässlich. Incidents werden auf Basis einer vereinbarten Klassifizierung priorisiert, um sicherzustellen, dass jene Störungen mit den höchsten geschäftlichen Auswirkungen zuerst gelöst werden. [6]

Moderne ITSM-Tools bieten hierfür bereits Ansätze zur Optimierung durch KI-Unterstützung, diese basieren jedoch meist auf generativer KI und können domänenspezifische Incidents daher nur generisch beantworten. [7]

Genau an dieser Schnittstelle setzt die vorliegende Arbeit an, um durch Deep Learning, eine Form des maschinellen Lernens, die Klassifizierung zu automatisieren und die Effizienz der Diagnose und Wiederherstellung durch Unternehmensspezifisches KI-Training zu steigern.

---

4 Vgl. *Yu, Q.*, Artificial Intelligence to enhance IT Operations, 2024, S. 19.
5 Vgl. *Gupta, R.*, Incident classification, 2008, S. 142.
6 Vgl. *Chen, Z.*, Software Engineering Conference, 2020, S. 1487.
7 Vgl. *Intercom*, Service Trends Report, 2024, *Zugriff am 10.01.2026*

1.2. Zielsetzung und Forschungsfragen

Die Zielsetzung dieser Arbeit ist die experimentelle Programmentwicklung eines (auf dem DistilBERT-Modell basierenden) Encoder-Modells, welches durch Training mit ITSM-Ticketdaten für den Zweck der ITSM-Ticketklassifizierung spezialisiert werden soll.

Daraus ergibt sich die erste und primäre Forschungsfrage, mit der sich diese Seminararbeit beschäftigt:

- Ist es möglich mit einem solchen DistilBERT-Modell ein – nach qualitativen Metriken – besseres Ergebnis in der Klassifizierungs- und Priorisierungsgüte für ITSM-Tickets zu erreichen als größere nicht spezifisch auf diesen Anwendungsfall trainierte LLMs im Zero-Shot Verfahren?

Bei der Betrachtung dieser Forschungsfrage ist zu beachten, dass die Wahl der zum Vergleich herangezogenen LLMs mit begrenzten wirtschaftlich/ökonomischen Mitteln getroffen wurde. Deshalb kamen bei dieser Forschungsfrage nicht die ursprünglich geplanten LLMs wie Gemini oder ChatGPT zum Einsatz, sondern kleinere Decoder-Modelle. Auf die exakten verwendeten Modelle wird in Kapitel 3.2 genauer eingegangen.

Auf dem Weg zur Erreichung des primären Forschungsziels stellten sich auch einige technische Fragen, aus welchen sich die sekundäre Forschungsfrage ergab:

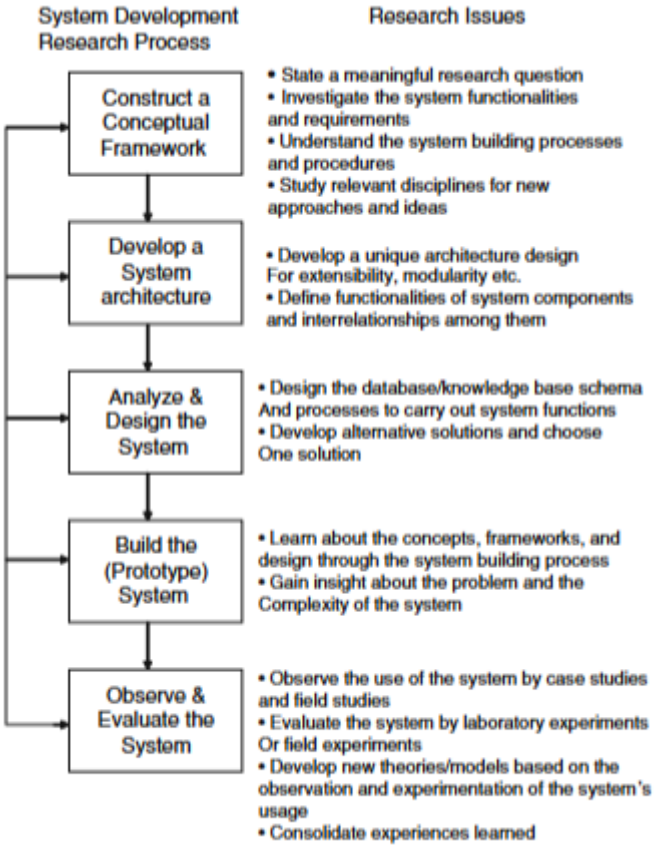
- Welcher Umfang an Trainingsdaten und welche Anzahl an Trainingsiterationen (Epochen) sind erforderlich, um unter Berücksichtigung der sogenannten Hyperparameter-Konfiguration das bestmögliche Ergebnis im Sinne des Forschungsziels zu erzielen?

1.3. Forschungsmethodik und Vorgehensweise

Diese Arbeit verfolgt einen kombinierten Ansatz, der die experimentelle Entwicklung eines feintrainierten Modells mit dem konstruktionswissenschaftlichen Paradigma („Design Science“) verbindet. [8] [9]

[8] Vgl. *Österle, H.*, Gestaltungsorientierte Wirtschaftsinformatik, 2010, S. 667 ff.
[9] Vgl. *Hevner, A.*, Design Science Research Methodology, 2010, S. 28–30.

Abbildung 1: Design Science Research Methodology



Quelle: *Hevner, A.*, Design Science Research Methodology, 2010, S. 26.

Der systematische Vergleich von zwei Modellen unter kontrollierten Bedingungen ermöglicht es, die Forschungsfragen hinreichend zu beantworten und gleichzeitig ein Artefakt (das mit den ITSM-Datensätzen trainierte DistilBERT-Modell) zu erstellen und zu evaluieren.

Um die Entwicklung eines funktionsfähigen spezialisierten Modells zu ermöglichen, wird zunächst in dem Grundlagenteil dieser Arbeit die zu diesem Thema gefundene Literatur zusammengefasst, um den derzeitigen Stand der Technik zu reflektieren.

Neben den Grundlagen werden Parameter bestimmt, welche für einen Vergleich nach Laborbedingungen erforderlich sind.

Der Experimentalablauf und die damit verbundenen Erkenntnisse werden zusammengefasst und in dem Versuchsaufbau beschrieben.

Ein systematischer Vergleich der beiden Modelle unter kontrollierten Bedingungen wird durchgeführt und beschrieben.

Das trainierte DistilBERT-Modell wird optimiert, um seine Leistung zu verbessern. Hierbei wird insbesondere auf das Thema des Overfittings eingegangen.

Bei der Entwicklung des trainierten Modells wird als Hilfsmittel für die Programmierung auf das aktuelle generative KI-Programmierwerkzeug *Gemini Pro* zurückgegriffen, um das Debugging von Code zu erleichtern, Funktionen für das KI-Training und einen Chat mit dem Modell zu implementieren. Da im Rahmen der Entwicklung sehr umfangreiche iterative Prompt-Sequenzen verwendet wurden, wird dieser Projektarbeit im Anhang ein ausführliches KI-Verzeichnis beigefügt, welches eine vollständige Übersicht über alle vorgenommenen iterativen Schritte enthält. [10] In dieser Arbeit selbst wird jedoch nur an wenigen inhaltlich relevanten Stellen daraus zitiert.

2. Grundlagen

Um den Inhalten von späteren Kapiteln folgen zu können, müssen zuerst einige Begriffsdefinitionen eingeführt und erklärt werden. Dabei geht es hauptsächlich um technisch-organisatorische Hintergründe im Bereich des ITSM sowie rein technische Grundlagen aus dem Bereich der neuronalen Netze und Transformer Architektur.

2.1. Fachliche Grundlage: ITSM-Ticket-Klassifizierung

Dieses Kapitel legt das theoretische Fundament für das Verständnis der Ticket-klassifizierung im Kontext des IT-Service Managements. Es beleuchtet die relevanten Standards, insbesondere ITIL 4, definiert die unterschiedlichen Ticket-Typen und erläutert die Mechanismen der Priorisierung, welche durch den in dieser Arbeit entwickelten Deep-Learning-Ansatz automatisiert werden sollen.

2.1.1. IT-Service-Management und Frameworks

IT-Service-Management umfasst alle Praktiken, Maßnahmen und Methoden, die nötig sind, um die bestmögliche Unterstützung von Geschäftsprozessen durch die IT-Organisation zu erreichen. Der Fokus liegt dabei auf der Kundenorientierung, Wertschöpfung

[10] Anhang: KI-Verzeichnis.

und Servicequalität. [11] Wie in der Einleitung erwähnt, ist die IT Infrastructure Library (ITIL) der de-facto Standard in der IT-Branche. [12]

Nach ITIL 4 wird ein Incident definiert als „eine nicht geplante Unterbrechung eines Service oder eine Qualitätsminderung eines Service“. Die Incident Management Practice verfolgt das Ziel, die negativen Auswirkungen solcher Vorfälle zu minimieren und den normalen Servicebetrieb so schnell wie möglich wiederherzustellen. [13]

In der aktuellen Version, ITIL 4, wird der Fokus von reinen Prozessen hin zu „Practices“ (Praktiken) verschoben. Für die Ticket- Klassifizierung ist primär die Incidents Management Practice sowie die Service Request Management Practice relevant. Neben diesen beschreibt das Framework weitere Practices wie zum Beispiel das Change-Enablement, auf welches wir in dieser Arbeit jedoch nicht näher eingehen werden, da der Prozess der Klassifizierung innerhalb eines ITSM-Tools („Ticketsystem“) bei jedem eingehenden Ticket ähnlich abläuft (Annahme → Klassifizierung → Weiterleitung). [14]

Ein Ticketsystem dient hierbei als Datendrehscheibe. Sie bündelt durch das Zusammenfassen mehrerer Kommunikationskanäle auf einer zentralen Plattform die Kommunikation zwischen den Serviceverbrauchern („End-Users“) und den Mitarbeitern der IT-Abteilung („Agents“). Hierbei beschreibt man ein Ticketsystem sinngemäß als „Anfrage- und Vorgangsverwaltungssystem“ oder „Service- und Störungsmanagementsystem“, worin ein Ticket ein einzelner kontextbezogener Vorgang (Thread) ist. [15] Mit dem Eröffnen eines neuen Kontext wird immer auch ein neues Ticket mit einer eindeutigen Identifikationsnummer („Ticketnummer“) erzeugt. Dies dient dazu verschiedene Störungen, Anfragen und Vorgänge leichter voneinander zu trennen, um sie dennoch thematisch und nach weiteren Kriterien, oft einem übergeordneten Problem, zuzuordnen. Die Qualität der Daten in diesen Tickets ist entscheidend für die nachgelagerte Bearbeitung. [16]

Auch für die Durchführung des Experimentes ist eine hohe Qualität von Tickets durch hohen Informationsgehalt wichtig. Nur Tickets mit einem ausreichend hohen Informationsgehalt, der eine Qualifizierung einer Anfrage ermöglicht, sind für eine maschinelle

---

11 Vgl. *Axelos Ltd.*, ITIL 4 Edition, 2019, S. 10–16.
12 Vgl. *Pilogret, L.*, Managing IT, 2025, S. 38.
13 Vgl. *Agutter, C.*, Incident Management, 2020, S. 67.
14 Vgl. *Pilogret, L.*, Managing IT, 2025, S. 189.
15 Vgl. *Eichstädt, S.*, Support, 2024, S. 238–239.
16 Vgl. *Pilogret, L.*, Managing IT, 2025, S. 33.

Verarbeitung geeignet. Kontextlose Tickets wie „Ich habe ein Problem“ oder „Bitte um Rückruf“ haben einen zu geringen Informationsgehalt und können nicht ohne Rückfrage an den Ticketersteller korrekt qualifiziert werden. Diese Art von kontextlosen Tickets eignet sich nicht für das Training eines nicht interaktiven oder zustandslosen Modells wie in dieser Arbeit verwendet. [17]

2.1.2. Differenzierung von Ticket-Typen

Eine fundamentale Anforderung an ein Klassifizierungssystem ist die korrekte Unterscheidung der Anfrageart, da Störungen oft eine höhere Dringlichkeit besitzen als Serviceanfragen. Eine Fehlklassifizierung würde zu einer ineffizienten Ressourcenallokation führen. [18]

Die folgenden zwei Ticket-Typen sind daher von besonderer Relevanz.

- Incident (Störung):
  Ist eine „nicht geplante Unterbrechung eines Service“. Das Ziel ist die schnellstmögliche Wiederherstellung. Ein Beispiel wäre ein ausgefallener Server oder die Fehlermeldung eines Druckers.
- Service Request (Serviceanfrage):
  Hierbei handelt es sich um eine formale Anfrage eines Anwenders nach etwas das bereitgestellt werden soll. Zum Beispiel Informationen, Zugriffsberechtigungen oder neue Hardware. Es liegt keine Störung des Betriebs vor. [19]

In der Praxis kann man diese Vorklassifizierung bereits durch den Anfragenden durchführen lassen, indem man zum Beispiel ein „Self-Service“-Portal für Service Requests und Incidents zur Verfügung stellt oder auf zwei verschiedene Emailadressen für diese Tickets zurückgreift.

Ungeachtet dieser Vorklassifizierungsmöglichkeiten sind oft weiterführende Informationen unerlässlich, um ein erfolgreiches Ticket-Routing zu gewährleisten. Service-Agenten müssen häufig die initialen Angaben validieren, fehlende Details ergänzen und ihr Domänenwissen nutzen, um den genauen Umfang (Scope) des Problems zu erfassen. Eine

---

[17] Vgl. *Yu, Q.*, Artificial Intelligence to enhance IT Operations, 2024, S. 2.
[18] Vgl. *Topalovic, D.*, Advisera, 2025, *Zugriff am 25.01.2026*
[19] Vgl. *Agutter, C.*, Incident Management, 2020, S. 68 ff.

besondere Herausforderung liegt hierbei in der Bestimmung von Dringlichkeit und Auswirkung. Da diese Einschätzung in der Praxis für End-Anwender oft schwierig ist, erfolgt die finale Priorisierung in der Regel durch die Agenten auf Basis spezifischer Service Level Agreements (SLA).

2.1.3. Der Prozess der Klassifizierung und Priorisierung

Sobald ein Ticket erstellt wurde, muss es klassifiziert werden. Dieser Prozess besteht aus zwei Dimensionen, die für das Training des Encoder-Modells (DistilBERT) in dieser Arbeit von zentraler Bedeutung sind:

1. Kategorisierung (Thematische Zuordnung):
   Das Ticket wird einem Sachgebiet zugeordnet. Zum Beispiel „Hardware → Drucker → Papierstau“, „Software → E-Mail → Passwortrücksetzung“, einem einzelnen übergeordneten Sachgebiet „Onboarding“ oder weiteren. Diese Hierarchie hilft dabei, das Ticket einem „Tier“ (einer Stufe), der korrekten Support-Einheit zuzuweisen.
2. Priorisierung (Dringlichkeitsbewertung):
   Die Priorisierung bestimmt die Reihenfolge der Bearbeitung und steuert die strukturierte Warteschlange („Ticket-Queue“). Die Priorität ergibt sich aus einer Matrix, die zwei Faktoren kombiniert:
   i. Impact (Auswirkung):
      Ob es nur einen Benutzer, mehrere oder das gesamte Unternehmen betrifft.
   ii. Urgency (Dringlichkeit):
      Wie zeitkritisch eine Lösung notwendig ist, also ob ein geschäftskritischer Prozess blockiert ist, die Arbeit eingeschränkt jedoch fortführbar ist oder es einen Workaround gibt.

Zur Bestimmung der Priorität existieren in der Praxis verschiedene Berechnungsmodelle.

Für die in diesem Projekt verwendeten Datensätze kommt das Additionsverfahren zum Einsatz. Hierbei ergibt sich die Priorität aus der mathematischen Summe von Auswirkung und Dringlichkeit. Dieses Vorgehen führt zu einer linearen Verteilung, die in fünf handlungsrelevante Prioritätsstufen konsolidiert wird.

Wie in Abbildung 2 dargestellt, folgt daraus die Logik:

Abbildung 2: Prioritätsformel

$$Priorität = Auswirkung + Dringlichkeit$$

Quelle: Eigene Darstellung

Eine hohe Auswirkung gepaart mit hoher Dringlichkeit führt zur höchsten Priorität („Critical“), während geringe Auswirkung und geringe Dringlichkeit zu einer niedrigen Priorität führen.

Tabelle 1: Prioritätsmatrix D_A

		Impact		
		1	2	3
Urgency	1	2 (very low)	3 (low)	4 (medium)
	2	3	4	5 (high)
	3	4	5	6 (critical)

Quelle: Eigene Darstellung in Anlehnung an *Topalovic, D.,* Advisera, 2026

Daraus resultierend basiert das KI-Training dieser Arbeit auf einem Datensatz mit fünf Prioritätsstufen, was eine präzisere Klassifizierung und Unterscheidung gegenüber anderer in weiteren öffentlichen Datensätzen gefundener Bewertungsschemata (z.B. low, medium, high) ermöglicht. [20]

Die Entscheidung für einen Klassifikationsansatz anstelle eines Regressionsmodells resultiert aus der diskreten Struktur der ITSM-Daten. Prioritäten und Dringlichkeiten sind ihrer Natur nach diskrete, kategoriale Werte. Abstände zwischen den verschiedenen Prioritätsstufen sind nicht linear gleichmäßig. Da diese auch in den Datensätzen nicht als stetige Variablen vorliegen, bildet ein Klassifikationsmodell mit Zuordnungskästen für die nicht linear zugewiesenen Prioritäten die zugrundeliegenden ITSM-Prozesse methodisch korrekt ab.

[20] Vgl. *Bueck, T.,* Alternative Dataset, 2025, *Zugriff am 15.01.2026.*

2.1.4. Herausforderung der manuellen Klassifizierung

Obwohl die oben genannten Regeln klar definiert scheinen, ist die praktische Umsetzung durch Menschen oft fehlerbehaftet.

- Subjektivität:
  Anwender tendieren dazu, ihre eigenen Probleme stets als „dringend“ einzustufen, was die Matrix verzerrt. Auch soziale Sympathien (Freundschaften oder soziale Bevorzugungen) führen zu Ressourcenallokationen.
- Unstrukturierte Daten:
  Tickets bestehen häufig aus Freitextfeldern („Beschreibung“) oder werden durch Emails erzeugt, die Qualität der Texte variiert stark, sie können Umgangssprache, Tippfehler, Stichworte oder irrelevante Informationen enthalten.
- Komplexität:
  In großen IT-Landschaften gibt es viele Kategorien. Ein menschlicher Bearbeiter ohne mehrjähriges Domänenwissen, kann kaum alle Zuweisungsregeln auswendig kennen, was zu „Ping-Pong-Effekten“ führt, bei denen Tickets mehrfach zwischen Tiers hin- und hergeschoben werden. Falsche Ticket-Zuweisungen sind mit Grund für lange Bearbeitungszeiten. [21]

Genau diese Defizite der manuellen Verarbeitung begründen den in Kapitel 1.2 formulierten Forschungsbedarf nach einer automatisierten, wissensbasierten Klassifikation durch Deep Learning.

2.2. Technische Grundlage: Die Transformer-Architektur

Die Transformer-Architektur dient als Grundlage der Umsetzung unseres Ansatzes zum NLP und zur Durchführung dieser Arbeit. Zum Verständnis müssen einige Grundlagen bezüglich der Architektur erläutert und die Wahl dieser Architektur begründet werden.

2.2.1. Tokenizer und Embedding-Layer

Da alle in dieser Arbeit referenzierten neuronalen Netze die Eingaben auf rein mathematischer Basis verarbeiten, müssen alle Wörter aus Eingabesequenzen erst durch den sogenannten „Tokenizer“ in Token-IDs/Indizes und anschließend durch einen „Embedding

[21] Vgl. *Al-Hawari, F.*, machine learning based helpdesk, 2021, S. 708.

Layer“ von Token-IDs/Indizes in entsprechende Vektorrepräsentationen der Wörter umgewandelt werden. Vorab: Jedes Wort einer Eingabesequenz wird umgangssprachlich als „Token“ bezeichnet. Die vom Tokenizer „nachgeschlagenen“ Nummern der Tokens sind entweder die Token-IDs oder Indizes.

Der Tokenizer lässt sich vereinfacht als Liste aller Tokens beschreiben, in welcher jedes Wort (jeder Token) eine feste Zahl (die Token-ID/den Index) zugewiesen hat.

Um die Funktionsweise des Embedding Layer anschaulich zu beschreiben, lässt sich das Konzept mit einem umfangreichen Nachschlagewerk vergleichen.

In einem herkömmlichen System wäre ein Wort lediglich durch seine Nummer (die Token-IDs/Indizes i) identifiziert. Bengio et al. führten jedoch eine Matrix C ein, die wie eine Sammlung von detaillierten „Steckbriefen“ fungiert. Wenn das Modell den Index eines Wortes erhält, „schlägt“ es in dieser Matrix nach und extrahiert den für diesen Index hinterlegten Eintrag $C(i)$. (Technisch wird die Matrix C als zweidimensionales Array (oder Lookup-Table) der Dimension $|V|*m$ realisiert, wobei $|V|$ die Größe des Vokabulars und m die Dimension des Feature-Vektors ist.)

Dieser Eintrag ist keine einzelne Zahl, sondern eine Reihe von vielen Zahlenwerten (ein Vektor), die gemeinsam die Eigenschaften des Wortes beschreiben. Der entscheidende Vorteil dieses Verfahrens ist, dass Wörter mit ähnlicher Bedeutung (z.B. „Fehler“ und „Störung“) im Laufe des Trainings eines NLM sehr ähnliche Zahlenreihen in ihren Steckbriefen entwickeln. Das Modell lernt somit, dass diese Wörter mathematisch verwandt sind, ohne dass ein Mensch diese Regeln vorab definieren musste.

Solche in neuronalen Netzen verwendeten Vektorrepräsentationen von Wörtern nennt man „Word Embeddings“ oder kurz einfach „Embeddings“. [22]

2.2.2. Der Paradigmenwechsel: Von RNNs/LSTMs zum Transformer

Vor der Etablierung der Transformer-Architektur stellten RNNs (Recurrent Neural Networks) den Standard für die Verarbeitung natürlicher Sprache dar. RNNs verarbeiten die

[22] Vgl. *Bengio, Y.*, NPL Model, 2003, Section 2: A Neural Model, S. 1141-1143.

Word Embeddings der Eingabesequenz iterativ, wobei der Hidden State (verborgene Zustand) des aktuellen Zeitschritts sowohl von dem Word Embedding des aktuellen Tokens als auch von den Zuständen aller Vorgänger abhängt.

Diese wiederkehrende Struktur ermöglicht theoretisch die Modellierung von Kontext. In der Praxis leiden einfache RNNs jedoch unter dem Problem des verschwindenden Gradienten (Vanishing Gradient Problem), was dazu führt, dass Abhängigkeiten in längeren Textsequenzen – wie sie in ITSM-Tickets üblich sind – verloren gehen. [23]

LSTMs (Long Short-Term Memory Netzwerke) adressieren dieses Defizit durch die Einführung eines Zellzustands (Cell State) und spezialisierter Gating-Mechanismen (Input-, Forget- und Output-Gate), die den Informationsfluss regulieren. Obwohl LSTMs Kontext über längere Eingabesequenzen erhalten können, bleibt ihre fundamentale Schwäche die sequenzielle Verarbeitung.

Da die Berechnung des Zustands zu einem Zeitschritt *t* zwingend den Abschluss der Berechnung des Schritts *t*-1 erfordert, ist eine Parallelisierung der Berechnung innerhalb einer Trainingssequenz nicht möglich. Dies limitiert die Trainingsgeschwindigkeit und die effektive Nutzung großer Datensätze mit komplexen kontextuellen Zusammenhängen massiv. [24]

In einem Review zum Machine Learning beschreibt Alessandro V. Villareal, dass klassische Machine-Learning-Ansätze zur Klassifizierung von ITSM-Tickets häufig auf SVM (Support Vector Machine), Random Forest, Decision Trees, Naive Bayes, Logistischer Regression oder kNN (K-Nearest Neighbors) basieren. [25] Diese Ansätze erforderten, im Gegensatz zur Transformer-Architektur, dass jedes Modell von Grund auf trainiert wird.

2.2.3. Das Kernkonzept: Self-Attention

Hier setzt die Transformer-Architektur an, die einem Modell durch den Self-Attention-Mechanismus erlaubt, jedes Token der Eingabesequenz mit jedem anderen Token in Beziehung zu setzen. Dies erlaubt eine Berechnung der „Relevanz“ eines Tokens relativ zu jedem anderen Token der Eingabesequenz. Im Gegensatz zu RNNs und LSTMs kann

---

[23] Vgl. *Bengio, Y.*, Gradient Descent, 1994, Section 1: Introduction, S. 157–166.
[24] Vgl. *Hochreiter, S.*, LSTM, 1997, Section 4: Long Short-Term Memory, S. 6–9.
[25] *Villareal, A.*, Literature Review, 2022, S. 216 ff.

durch das Konzept der Self-Attention die Relevanz aller Tokens zueinander zeitgleich berechnet werden.

Um den Self-Attention-Score (die Relevanz) eines jeden Tokens zu errechnen, wird das Embedding jedes Tokens zunächst mittels gelernter linearer Transformationen in drei spezifische Vektoren projiziert: Query (*Q*), Key (*K*) und Value (*V*). Die Relevanz wird durch das Skalarprodukt von *Q* und *K* berechnet, skaliert durch die Wurzel der Dimension d_k und normalisiert durch eine Softmax-Funktion. Diese Relevanz gewichtet anschließend *V*, welcher schließlich die neue kontextualisierte Vektorrepräsentation des Wortes ergibt. [26]

Die Formel wird in Abbildung 3 dargestellt.

Abbildung 3: Attention Formel

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Quelle: *Vaswani, A.*, Attention, 2017, Section 3.2: Attention, S. 4

Ein Beispiel zum Verständnis:

Der Self-Attention-Mechanismus vergleicht alle Wörter des Satzes „Die Bank gibt Geld" miteinander. Dies ist notwendig, da das Wort „Bank" isoliert betrachtet mehrdeutig ist (Finanzinstitut vs. Parkbank). Wenn der Mechanismus das Wort „Bank" verarbeitet (als Query), ermittelt er durch das Skalarprodukt mit dem Key von „Geld" eine hohe Übereinstimmung. Das resultierende hohe Attention-Gewicht sorgt dafür, dass Informationen vom Value-Vektor des Wortes „Geld" stark in die neue Repräsentation von „Bank" einfließen. Das Modell „versteht" somit mathematisch, dass „Bank" in diesem Kontext finanzbezogen ist.

Da Self-Attention die Eingabe als ungeordnete Menge von Vektoren betrachtet, geht die Information über die Wortreihenfolge initial verloren. Um dies zu korrigieren, wird vor der Einspeisung in den Self-Attention-Layer ein sogenanntes „Positional Encoding" auf

[26] Vgl. *Vaswani, A.*, Attention, 2017, Section 3.2: Attention, S. 3–5.

die Embeddings addiert. Ohne diesen Schritt wären Sätze wie „Die Bank gibt Geld“ und „Die Geld gibt Bank“ für das Modell mathematisch identisch. [27]

2.2.4. Encoder-Modelle

Klassische Transformer bestehen aus zwei Teilen (Stacks): dem Encoder und dem Decoder.

Der Encoder ist der Teil des neuronalen Netzwerks, der für das Verständnis der Eingabesequenz zuständig ist. Zuerst holt sich der Encoder die entsprechenden Indizes zu allen Tokens. Damit übersetzt der Encoder die Tokens der Eingabesequenz anhand des Embedding Layers in Word Embeddings.

Anschließend verwendet er bidirektionale Self-Attention, wodurch parallel kontextuelle Zusammenhänge zwischen allen Tokens hergestellt werden. Das bedeutet, das Modell „liest“ alle Wörter aus der Eingabesequenz gleichzeitig und kann Zusammenhänge zwischen Wörtern in beide Richtungen (links-nach-rechts und rechts-nach-links) bilden. Nach der Self-Attention folgt in jedem Layer des neuronalen Netzes des Encoders ein sogenanntes Feed-Forward Network (FFN), welches die gesammelten Informationen weiterverarbeitet.

Weil bei der Anwendung des Self-Attention-Mechanismus die Reihenfolge der Wörter aus der Eingabesequenz verloren geht, muss der Encoder zusätzlich durch das vorher erwähnte Positional Encoding die Reihenfolge der Wörter aus der Eingabesequenz erhalten.

Mit der Durchführung dieser Schritte erreicht der Encoder zum Schluss eine hochdimensionale, kontextsensitive Vektorrepräsentation, welche die semantischen Merkmale der Eingabesequenz abstrahiert. Bei dem in dieser Arbeit antrainierten und verwendeten NLM „DistilBERT“ handelt es sich um ein solches Encoder-Modell.

Für die Klassifikation von ITSM-Tickets wird diese Repräsentation anschließend durch sogenanntes „Pooling“ auf einen einzigen Vektor reduziert, der als Input für den finalen Klassifikator (Classification Head) dient, um den Status des Tickets zu erhalten. In BERT-basierten Modellen wird hierfür standardmäßig der finale Vektor des speziellen

[27] Vgl. *Vaswani, A.*, Attention, 2017, Section 3.5: Positional Encoding, S. 5 ff.

[CLS]-Tokens verwendet, der als aggregierte Vektorrepräsentation der gesamten Eingabesequenz fungiert. [28]

Diese aggregierte Vektorrepräsentation des [CLS]-Tokens dient als Input für den finalen Classification Head. Technisch handelt es sich dabei um eine einfache lineare Schicht (Fully Connected Layer), die auf das vortrainierte Modell aufgesetzt wird. Diese Schicht projiziert den hochdimensionalen Vektor (bei DistilBERT 768 Dimensionen) auf einen Vektor der Größe *K*, wobei *K* der Anzahl der möglichen Ticket-Status entspricht (die sogenannten Logits). Eine abschließende Softmax-Funktion wandelt diese Logits in Wahrscheinlichkeitswerte um, deren Summe 1 ergibt. [29]

2.2.5. Decoder-Modelle

Während der Encoder darauf spezialisiert ist, eine vollständige Eingabe zu analysieren und zu repräsentieren, fungiert der Decoder als der generative Teil der Transformer-Architektur. Sein Ziel ist es, basierend auf den Informationen des Encoders und den bereits generierten Worten das nächste Wort der Ausgabesequenz vorherzusagen.

Ähnlich wie der Encoder besteht auch der Decoder aus einem Stack mehrerer identischer Layer. Der Prozess beginnt ebenfalls mit der Umwandlung von Token-IDs/Indizes in Word Embeddings und der Addition von Positional Encodings, da auch hier keine inhärente Zeitachse existiert. Innerhalb der Layer gibt es jedoch zwei fundamentale Unterschiede zum Encoder:

1. Masked Self-Attention:

   Der Decoder darf bei der Vorhersage eines Wortes nicht auf zukünftige Wörter zugreifen, da diese zum Zeitpunkt der Generierung noch nicht existieren. Um das im Training zu simulieren, wird die Self-Attention „maskiert“. Das bedeutet, dass alle Positionen rechts vom aktuellen Token mathematisch ausgeblendet werden, sodass der Decoder nur Zusammenhänge zu bereits generierten Wörtern („in die Vergangenheit“) herstellen kann.

2. Encoder-Decoder Attention (Cross-Attention):

---

[28] Vgl. *Devlin, J.*, BERT Pre-training, 2019, Section 3: BERT, S. 4173–4175.
[29] Vgl. *Devlin, J.*, BERT Pre-training, 2019, Section 4.1: GLUE, S. 4175 ff.

Dies ist die Schnittstelle zwischen den beiden Stacks. Nach der maskierten Self-Attention greift der Decoder auf die vom Encoder erstellte kontextsensitive Vektorrepräsentation der Eingabe zu. Technisch bedeutet dies, dass die Queries (Suchanfragen, *Q*) aus dem Decoder stammen, während die Keys (*K*) und Values (*V*) (Inhalte) aus dem Output des Encoders geliefert werden. So kann der Decoder bei jedem Schritt entscheiden, welche Teile der ursprünglichen Eingabe (z. B. des Tickets) für das nächste zu generierende Wort relevant sind.

Wie im Encoder folgt auch hier in jedem Layer ein Feed-Forward Network (FFN). Der Output des letzten Decoder-Layers wird dann in Wahrscheinlichkeiten für das nächste Wort aus dem gesamten Vokabular umgewandelt. [30] [31]

Die Relevanz dieser Decoder-Architektur ergibt sich für diese Arbeit aus dem angestrebten Vergleich der Modelle. Zwar nutzt das Modell „DistilBERT“ als spezialisierter Klassifikator ausschließlich den Encoder-Stack, allerdings ist ein Kernziel dieser Untersuchung der Vergleich der Klassifizierungsgüte dieses Ansatzes mit modernen Large Language Models (LLMs). Diese LLM-Modelle basieren technisch auf der hier beschriebenen (generativen) Decoder-Architektur.

Der wesentliche Unterschied im Vorgehen liegt darin, dass DistilBERT eine mathematische Klassifikation durch den Classification Head durchführt, während Decoder-basierte LLMs die Klassifikationsaufgabe lösen, indem sie das gewünschte Label (z. B. „Priorität: hoch“) als Text generieren.

2.3. Abgrenzung: Deep Learning vs. Machine Learning

Da „Deep Learning” und „Machine Learning” im Bereich von KI häufig vertauscht oder synonym verwendet werden, ist es hilfreich diese Begriffe einmal kurz zu definieren und zu differenzieren.

Machine Learning ist der Überbegriff für Algorithmen, die Muster in Daten erkennen und daraus Vorhersagen ableiten, ohne explizit für jeden Einzelfall programmiert zu sein. Deep Learning ist eine Teilmenge des Machine Learning. Während klassisches Machine Learning oft auf manuell definierte Merkmale angewiesen ist, spezialisiert sich Deep

[30] Vgl. *Vaswani, A.*, Attention, 2017, Section 3.1: Encoder and Decoder Stacks, S. 2 ff.
[31] Vgl. *Vaswani, A.*, Attention, 2017, Section 3.2.3: Applications of Attention to our Model, S. 5.

Learning darauf diese Merkmale, mittels künstlicher neuronaler Netze mit vielen Schichten (daher „Deep“), selbstständig aus den Daten zu extrahieren. Da die in dieser Arbeit verwendeten Transformer-Modelle auf vielschichtigen neuronalen Architekturen basieren, handelt es sich explizit um ein Deep-Learning-Verfahren. [32]

2.4. Das Paradigma: Transfer Learning

Die in dieser Arbeit angewandte Methode basiert auf dem Paradigma des Transfer Learning. Anstatt ein neuronales Netz von Grund auf nur für die Klassifikation von IT-Tickets zu trainieren – was Millionen von korrekt qualifizierten Tickets erfordern würde – wird ein zweistufiger Prozess gewählt. Zunächst wird ein Modell welches auf riesigen, allgemeinen Textmengen vortrainiert ist gewählt, um eine Basis an Sprachverständnis vorauszusetzen. Dieses Wissen wird anschließend auf die spezifische Domäne „ITSM“ transferiert. [33]

2.5. Underfitting, Overfitting und Generalisierung

Das zentrale Ziel beim Training neuronaler Netze ist die Generalisierung. Generalisierung meint die Fähigkeit eines Modells, auch auf bisher unbekannten Daten (z.B. Testdaten) korrekte Vorhersagen zu treffen. Gelingt dies nicht, liegt meist einer von zwei Fehlerzuständen vor:

Underfitting: Underfitting tritt auf, wenn das Modell bereits auf dem Trainingsdatensatz keinen ausreichend niedrigen Fehlerwert erreicht, weil beispielsweise die Kapazität des Modells nicht genügt, um die Komplexität der Daten abzubilden.

Overfitting: Overfitting entsteht, wenn die Fehlerquote auf den Trainingsdaten zwar minimal ist, die Fehlerrate auf den Testdaten jedoch hoch ausfällt. Dies geschieht, wenn das Modell begonnen hat die Trainingsdaten auswendig zu lernen, anstatt die allgemeingültigen Muster in den Daten zu erkennen und zu lernen. [34]

---

[32] Vgl. *Goodfellow, Ian,* Deep Learning, 2016, Figure 1.4., S. 9.
[33] Vgl. *Google Cloud,* Deep Learning auf Google Cloud, 2026, *Zugriff am 25.01.2026*
[34] Vgl. *Goodfellow, Ian,* Deep Learning, 2016, S. 110–116.

3. Umsetzung: Experimentelles Design und Durchführung

Dieses Kapitel beschreibt die technische Durchführung des Forschungsprojekts, welche zur Erreichung der in **Fehler! Verweisquelle konnte nicht gefunden werden.** definierten Ziele notwendig sind. Dazu zählen sowohl die technischen Vorbereitungen zur Normalisierung der Datensätze, die entsprechende Entwicklung und korrekte Konfiguration des Skriptes zum Fine-Tuning des DistilBERT-Modells, die anschließende eigentliche Durchführung des Experiments und das finale Vergleichen der Ergebnisse der verschiedenen Modelle.

3.1. DistilBERT

Bevor ein Transformer-Modell für eine spezifische Aufgabe wie die Klassifikation von ITSM-Tickets eingesetzt werden kann, muss es zunächst ein grundlegendes Verständnis von Sprache erlernen. Für das Experiment wird auf das vortrainierte DistilBERT-Modell „distilbert-base-uncased" zurückgegriffen. Dieses wird auf der Huggingface-Webseite bereitgestellt. [35] Das DistilBERT-Modell basiert auf BERT [36], einem bereits mit Wikipedia-Texten trainierten NLM das auf 110 Millionen Parametern basiert. Es besitzt dadurch die erforderliche Fähigkeit, um einfache Wortzusammenhänge in Texten erkennen zu können. „Uncased" bedeutet dabei, dass die Textquellen, welche bei diesem Modell zum Vortraining verwendet wurden, zuvor in Kleinbuchstaben umgewandelt wurden. Es behandelt daher keine Groß-/Kleinschreibung und wurde zudem ausschließlich mit Texten der englischen Sprache trainiert. [37]

Durch das Umwandeln des Vokabulars in „lower-case" (kleingeschriebenes) ohne Majuskeln und Versalien, wird die Größe des Modells reduziert.

Um die Größe des Modells noch weiter zu reduzieren, wurde die Menge der Input-Token des Attention-Mechanismus bei DistilBERT um ein Drittel verringert. [38] Hierdurch entsteht ein systematischer Fehler, welcher als Kompromiss, durch nur eine geringe Veränderung der Klassifizierungsgenauigkeit gegenüber dem BERT-Modell, hingenommen wird. Eigennamen wie zum Beispiel „Apple" (Unternehmen) und „apple" (Frucht) kann

---

35 *Huggingface*, distilbert_distilbert-base-uncased, 2019, *Zugriff am 15.01.2026*
36 *Huggingface*, bert-base-uncased, 2023, *Zugriff am 15.01.2026*
37 *Github*, google-research_bert, 2020, *Zugriff am 15.01.2026*
38 *Talaat, A. S.*, DistilBERT classification methods, S. 25928.

ein uncased Modell nicht unterscheiden. Hierdurch entsteht ein Bias zugunsten der häufigeren, generischen Bedeutung. Dadurch entstehen Bedeutungsverschiebungen, Embeddings werden weniger präzise und es tritt eine stärkere Vermischung von Wortbedeutungen auf. Das Modell bevorzugt englische Strukturen und ist für deutsche Syntaxsignale (im deutschen alle Substantive) nicht geeignet.

Da die bislang vorliegenden Datensätze zu großen Teilen in deutscher Sprache vorliegen, muss dies bei der Datenaufbereitung (3.4.1 unten) berücksichtigt werden.

BERT steht für „Bidirektional Encoder Representations from Transformers" was auf die Arbeitsweise des Modells hinweist. Es betrachtet Texte nicht nur von links nach rechts, sondern auch von rechts nach links mit Hilfe der Self-Attention. [39]

„Distil" in DistilBERT bedeutet, dass das Modell durch sogenannte Knowledge Distillation verkleinert wurde. Hierbei wurde das zuvor vortrainierte BERT-Modell, welches mit 110 Millionen Parametern trainiert wurde, verwendet um ein zu fast den gleichen Ergebnissen führendes Modell (DistilBERT) zu trainieren. Es wurde das Teacher und Student Modell für das Training von DistilBERT verwendet. Auf Grund dieser Trainingsarchitektur sind nur zwei Drittel der Attention-Layer erforderlich, wodurch das Modell schneller und kleiner wird. Es übernimmt den Bias des Teacher Modells und besitzt auf Grund dieses Verkleinerungsverfahren eine 3% geringere Genauigkeit. Es ist damit ganze 40% kleiner als BERT und 60% schneller. [40]

[39] Vgl. *Devlin, J.*, BERT Pre-training, 2019, S. 1.
[40] Vgl. *Sanh, V.*, DistilBERT Working-Paper, 2020, S. 5.

Abbildung 4: Vergleich BERT – DistilBERT

	BERT	DistilBERT
Size (millions)	**Base**: 110 **Large**: 340	**Base:** 66
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	3% degradation from BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	16 GB BERT data. 3.3 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT Distillation

DistilBERT was able to **reduce the size of a BERT model by 40%**, while **retaining 97% of its language understanding** capabilities and being **60% faster.**

Quelle: *Sagar, D.*, Knowledge Distillation, 2021 in Anlehnung an *Sanh, Victor et al.*, DistiBERT Working-Paper, 2020, S. 1–3.

Das Pre-Training wurde von Google und Huggingface auf riesigen Textkorpora (Wikipedia, BookCorpus) durchgeführt, hierdurch hat das Modell bereits ein tiefes Verständnis für Syntax, Grammatik und semantische Zusammenhänge erlernt.

Technisch basiert dies auf dem „Masked Language Modeling“ (MLM). Dabei werden Wörter in einem Satz maskiert (z. B. „Der Server ist [MASK]“), und das Modell muss das fehlende Wort vorhersagen.

Für das vorliegende Projekt bedeutet dies: Das Modell weiß bereits, dass zum Beispiel „langsam“ ein Adjektiv ist, das oft negative Zustände beschreibt, oder dass „Drucker“ und „Papier“ semantisch verwandt sind. Dieses „Allgemeinwissen“ ist im Artefakt bereits in den neuronalen Gewichten gespeichert, bevor das Fine-Tuning mit den ITSM-Tickets beginnt.

3.2. Die Kontrahenten

In diesem Abschnitt wird auf die Wahl der LLMs (die Decoder-Modelle) eingegangen und begründet, weshalb diese zum Vergleich mit dem in dieser Arbeit antrainierten DistilBERT-Modell, herangezogen wurden. Dabei ist die Wahl der LLMs – durch die Wirtschaftlichkeit und Verfügbarkeit – nicht wie ursprünglich geplant auf ChatGPT-und/oder Gemini-Modelle gefallen, sondern auf vergleichsweise kleinere LLMs, die dafür jedoch mit lokal zugänglicher Rechenleistung betrieben werden konnten.

Da die Wahl der LLMs auf kleinere Modelle gefallen ist, wurde versucht den Verlust in der Qualität der kontrahierenden LLMs durch die Quantität etwas auszugleichen. Die hier gewählten LLMs sind kostenlos und entsprechen dem aktuellen Stand der Technik, da sie von verschiedenen namentlichen Herstellern wie z.B. Meta AI, Microsoft, Google & Co. erstellt wurden. Einige der hier verwendeten LLMs galten – zum Zeitpunkt ihrer Veröffentlichung – in ihrem Bereich der Small Large Language Models (SLLMs) nach Standard-Benchmarks wie MMLU, GPQA und HumanEval auch zu den führenden Modellen. [41] [42] [43]

Folglich ergibt sich die in Tabelle 2 dargestellte Aufstellung der zum Vergleich herangezogenen LLMs:

Tabelle 2: Übersicht der kontrahierenden LLMs

Model	Parameteranzahl (in Milliarden)	Hersteller	Erscheinungsjahr
llama-4-scout-17b-16e-instruct	17	Meta AI	2025
qwen_qwen2.5-coder-14b	14	Alibaba Cloud	2024
phi-3-large-5.6b	5,6	Microsoft	2024
google_gemma-2-9b	9	Google	2024
falcon3-3b-instruct	3	TII	2024
mistral_latest	7	Mistral AI	2024
falcon3-1b-instruct	1	TII	2024
gemma-1b	1	Google	2024
darkidol-llama-3.1-8b-instruct-1.2-uncensored	8	Meta AI	2024

Quelle: Eigene Darstellung

[41] Vgl. *Grattafiori, A.,* Llama 3, 2024, S. 3.
[42] Vgl. *Jiang, A. Q.,* Mistral, 2023, S. 4.
[43] Vgl. *Abdin, M.,* Phi-3, 2024, S. 6.

3.3. Wissenschaftliche Evaluierungsmetriken

Gemäß der Anforderung an „Research Rigor" im Bereich des „Design Science Research" wurde ein quantitativer Evaluierungsansatz gewählt. [44] Im Rahmen dieser Arbeit wurden deshalb die Accuracy (Genauigkeit), der F1-Score, die Konfusionsmatrix sowie die Verlustfunktion (Loss) definiert und verwendet. [45] [46]

Der Einsatz dieser Metriken (Scores) erfolgt unterschiedlich je nach dem betrachteten Stack der Transformer-Architektur (Encoder oder Decoder). Für den direkten Leistungsvergleich zwischen dem klassifizierenden DistilBERT-Modell (Encoder) und den generativen LLMs (Decoder) werden die ergebnisorientierten Metriken der Accuracy, des F1-Scores und der Konfusionsmatrix herangezogen. Die Verlustfunktion (Loss) hingegen findet ausschließlich beim Training des DistilBERT-Modells Anwendung, um dessen Konvergenzverhalten zu beurteilen.

Zusammenfassend werden die genannten Evaluierungsmethoden für das Verständnis bei den Ergebnissen im Folgenden kurz definiert:

- Accuracy:
  Die Accuracy ist das intuitivste Maß: Sie gibt schlicht an, wie viel Prozent der Entscheidungen des Modells korrekt waren. Man teilt die Anzahl der richtigen Antworten durch die Gesamtzahl der Fragen. [47]

  Das Problem mit der Accuracy ist allerdings, dass sie täuschen kann. Wenn beispielsweise fast alle zum Testen verwendeten Tickets „unkritisch" sind, kann das Modell eine hohe Accuracy erreichen, indem es einfach immer „unkritisch" rät, obwohl es die wenigen kritischen Fälle komplett übersehen hat.

- Konfusionsmatrix:
  Während die Accuracy nur eine einzelne Zahl liefert (z. B. „90 % korrekt"), gibt

[44] Vgl. *Hevner, A.*, DSISR, 2004, S. 87 ff.
[45] Vgl. *Grandini, M.*, Multi-Class Metrics, 2020, S. 2–8.
[46] Vgl. *Goodfellow, I.*, Deep Learning, 2016, S. 82 ff.
[47] Vgl. *Grandini, M.*, Multi-Class Metrics, 2020, S. 3 ff.

die Konfusionsmatrix eine genaue Übersicht darüber welche Fehler gemacht wurden. Sie ist eine Tabelle, die das „Soll“ (die wahre Kategorie/Priorität) dem „Ist“ (der Vorhersage des Modells) gegenüberstellt. [48]

Dadurch sieht man auf einen Blick, ob das Modell nur harmlose Verwechslungen macht (z.B. Priorität „Niedrig“ statt „Sehr niedrig“) oder ob es gefährliche Fehler begeht (z.B. eine „Kritische“ Störung als „Niedrig“ einstuft). Sie macht das Fehlerverhalten transparent.

- F1-Score:

  Der F1-Score ist die wichtigste Metrik für unausgeglichene Datensätze. Er ist eine Kombination aus zwei Perspektiven: Der Precision und dem Recall.

  Precision (Genauigkeit):
  Sie beschreibt den Anteil der korrekt klassifizierten Tickets an der Gesamtheit aller Tickets, die das Modell für diese Priorität vorhergesagt hat.

  Recall (Sensitivität):
  Sie beschreibt den Anteil der korrekt erkannten Tickets im Verhältnis zur Gesamtzahl der Tickets, die tatsächlich dieser Klasse angehören.

  Beispiel zum Verständnis: Angenommen es liegt ein Test-Datensatz von 100 Tickets vor, von welchen 10 mit einer wahren Priorität von “Kritisch” bewertet sind. Wenn das Modell 5 der 100 Tickets als “Kritisch” klassifiziert, von denen jedoch nur 4 tatsächlich “Kritisch” sind, hat es eine Precision von 80% (4 von 5 Treffern), jedoch nur einen Recall von 40% (4 von 10 Fällen gefunden).

  Der F1-Score (als harmonisches Mittel dieser beiden Werte) ist deshalb entscheidend, da er einen Mittelweg erzwingt: Ziel ist es, so wenig Fehlalarme wie möglich zu produzieren (hohe Precision), ohne dabei echte Notfälle zu übersehen (hoher Recall). [49]

[48] Vgl. *Grandini, Margherita,* Multi-Class Metrics, 2020, S. 2.
[49] Vgl. *Grandini, Margherita,* Multi-Class Metrics, 2020, S. 3–8.

- Loss/Verlust:

  Im Gegensatz zu den drei oben genannten Metriken, die der nachträglichen Bewertung dienen, ist der Loss (Verlustwert) eine rein interne Zielfunktion des Optimierungsalgorithmus (Optimizers), die ausschließlich während der Trainingsphase des DistilBERT-Modells Anwendung findet.

  In jeder Trainingsiteration des Modells versucht der Optimizer diesen Fehlerwert schrittweise zu verringern. Dafür nutzt er das Verfahren der sogenannten Backpropagation: Er verfolgt den Fehler durch das neuronale Netz zurück und passt die internen Gewichte des Netzes minimal an, damit das Modell beim nächsten Durchlauf etwas genauer liegt.

  Sobald das Training abgeschlossen ist, verliert der Loss jedoch seine Funktion. Um zu beurteilen, wie gut das fertige Modell im echten Betrieb arbeitet, werden statt des Fehlerwerts dann Metriken wie der F1-Score, die Konfusionsmatrix und/oder die Accuracy verwendet. [50]

Zusätzlich zu den zuvor genannten Evaluierungsmetriken gibt es auch im Umgang mit den Datensätzen etwas zu beachten:

Um beim Antrainieren des DistilBERT-Modells und beim darauffolgenden Testen dieses Modells nicht auf die Problematik des Overfitting zu laufen, ist ein sogenannter Train-Test-Split unerlässlich. Beim Train-Test-Split handelt es sich um eine strikte Unterteilung aller ITSM-Ticket-Datensätze, welche bereits vor Beginn der Trainingsphase stattfindet. Dabei wird der gesamte ITSM-Ticket-Datensatz in 2 Teile unterteilt: Die Test-Datensätze und die Trainings-Datensätze.

Die Test-Datensätze werden dabei strikt bis nach der Trainingsphase des DistilBERT-Modells vorgehalten, um die Performance aller (sowohl Encoder- als auch Decoder-) Modelle miteinander zu vergleichen, während die Trainings-Datensätze ausschließlich zum Trainieren des DistilBERT-Modells verwendet werden. [51]

---

[50] Vgl. *Goodfellow, Ian,* Deep Learning, 2016, S. 82 ff.
[51] Vgl. *Goodfellow, Ian,* Deep Learning, 2016, S. 110 ff.

3.4. Versuchsaufbau und Durchführung

Im Anschluss an die Darstellung der in diesem Projekt verwendeten Evaluierungsmetriken wird in diesem Kapitel der Verlauf des Experiments erläutert. Die Aufbereitung der Datenbasis bildet die zentrale Voraussetzung für den Versuchsaufbau. Nachfolgend wird die Durchführung des Modell-Trainings der Modellevaluierung sowie die Trainingsparameter erläutert.

3.4.1. Die Datenbasis

Bei einer Recherche zur Datenbasis wurden verschiedene relevante Datensätze gefunden.

In diesem Experiment haben wir uns für die Verwendung von öffentlichen Datensätzen entschieden.

Es wird eine hohe Qualität der Trainingsdatensätze in dieser Projektarbeit vorausgesetzt, sie wurden jedoch nur stichprobenweise bei der Wahl der Datenquellen überprüft.

Um ein Training für die Klassifizierung von Prioritäten durchführen zu können, müssen die zu verwendenden Datensätze den folgenden beschriebenen Kriterien entsprechen und einen hochwertigen Kontext im Nachrichtenkörper, sowie eine präzise Qualifizierung vorweisen.

Die Spalten „Subject“ (Betreff), „Body“ (Nachrichtenkörper mit Anfragekontext) müssen vorhanden sein. Auch das zugeordnete kategorial ordinale Attribut „Priority“ muss vorhanden sein oder aus anderen Attributen abgeleitet werden können.

Zwar stellen Prioritätsstufen mathematisch eine geordnete Reihenfolge dar, jedoch handelt es sich operativ um diskrete Kategorien einer Ordinalskala, deren Abstände nicht linear interpretierbar sind. Der semantische Abstand zwischen einer sehr niedrigen und einer niedrigen Priorität ist nicht äquivalent zum Abstand einer hohen zu einer kritischen. Ein Regressionsmodell würde diese nicht-linearen semantischen Unterschiede fälschlich in eine lineare Skala überführen und dadurch künstliche Zwischenwerte erzeugen, die real nicht existieren und ein fehleranfälliges nachträgliches Runden erfordern würden. [52]

[52] *Github, Żak, K.*, Impact Dataset, 2019, *Zugriff am 15.01.2026*

Durch die Wahl eines klassifikationsbasierten Ansatzes wird das Modell gezwungen klare Entscheidungen analog zur Arbeitsweise eines menschlichen Agenten zu treffen, was die spätere Prozessintegration deutlich vereinfacht.

Es wird angenommen, dass mit Zunahme der Trainingsdatenmenge eine höhere Genauigkeit bei der Klassifizierung erzielt werden kann. Daher werden anhand einer Skalenharmonisierung Trainingsdatensätze aus unterschiedlichen Quellen (und damit unterschiedlichen Unternehmenskontexten) zu einem allgemeinen Trainingsdatenset zusammengeführt. Das zentrale Problem stellt hierbei das Zusammenführen der Zielvariablen dar.

Um die Varianz und Vielfältigkeit des Trainingsdatensatzes zu erhöhen, werden die Datensätze (D_A [53] und D_B [54]) ausgewählt, die sich mithilfe des in Abbildung 2 dargestellten Additionsschemas in kategorial-ordinale Prioritäten überführen lassen. Auf Basis der Prioritätsmatrix D_A und Harmonisierte Prioritätsmatrix D_B können beide Datenquellen zu einem gemeinsamen Trainingsdatenset D_{total} [55] vereinigt werden.

Es liegen die im folgenden beschriebenen Datensätze vor:

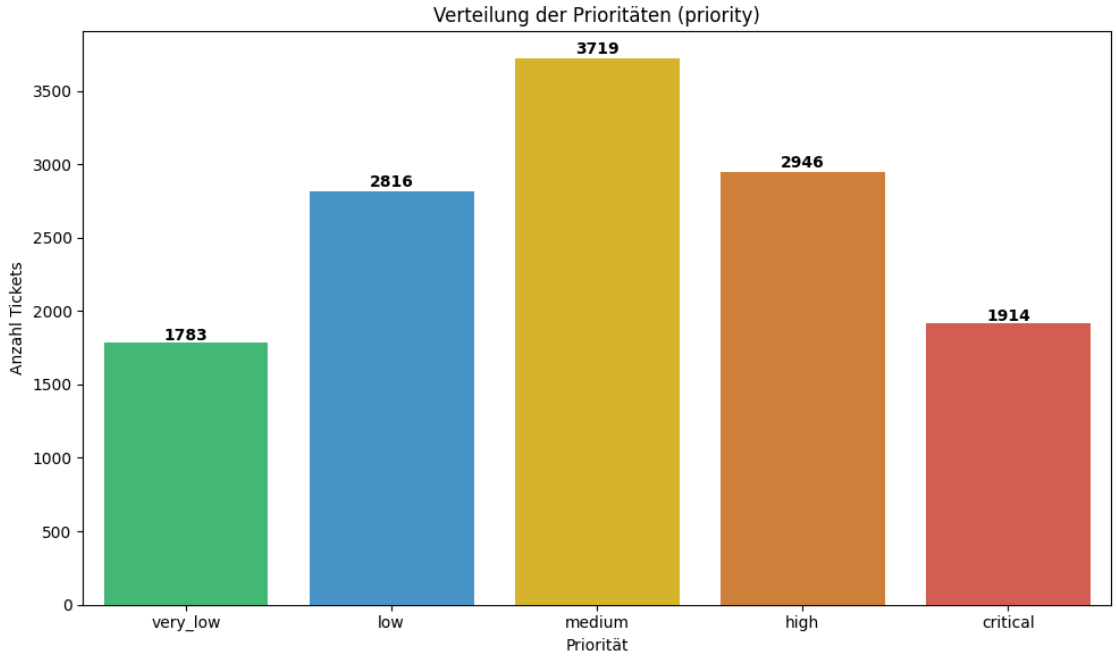
Trainingsdaten D_A:
„dataset-tickets-german_normalized_50_5_2.csv“, bereitgestellt auf der Kaggle-Plattform von Tobias Bueck [56], qualifiziert in 5 Prioritätsstufen wie in Abschnitt 2.1.3 oben beschriebenen. Es handelt sich um 13.178 Deutschsprachige Tickets.

Verteilung der Prioritäten D_A:

- critical: 1.914 Tickets (14.52%)
- high: 2.946 Tickets (22.36%)
- medium: 3.719 Tickets (28.22%)
- low: 2.816 Tickets (21.37%)
- very_low: 1.783 Tickets (13.53%)

[53] Anhang: „Quelltexte/#1 Original/ dataset-tickets-german_normalized_50_5_2.csv“
[54] Anhang: „Quelltexte/#1 Original/all_tickets.csv“
[55] Anhang: „Quelltexte/#2 Aufbereiten, Übersetzen und Harmonisieren/helsinki_final_training_dataset_harmonized.csv“
[56] *Bueck, T.,* Ticket Dataset, 2025, *Zugriff am 15.01.2026*

Abbildung 5: Verteilung der Prioritäten D_A



Quelle: Eigene Darstellung

Da BERT und auch DistilBERT auf der englischen Sprache basieren, wurden die vorliegenden Datensätze der Ticketdaten in die englische Sprache übersetzt.

Dieser Vorgang wurde mit Hilfe des von dem von der University of Helsinki für spezifische Sprachübersetzungen trainierten Modell „Helsinki-NLP/opus-mt-{src}-{tgt}“ [57] durchgeführt und stichprobenweise überprüft.

Zuvor musste, um den „Body“ der Tickets für die Übersetzung vorzubereiten, Zeilenumbrüche und Absätze mit RegEx freigestellt werden, da nicht freigestellte Absatzzeichen von dem verarbeitenden Modell sonst nicht korrekt verarbeitet wurden. [58]

Die Qualität der Übersetzungen war anschließend für ein Training ausreichend.

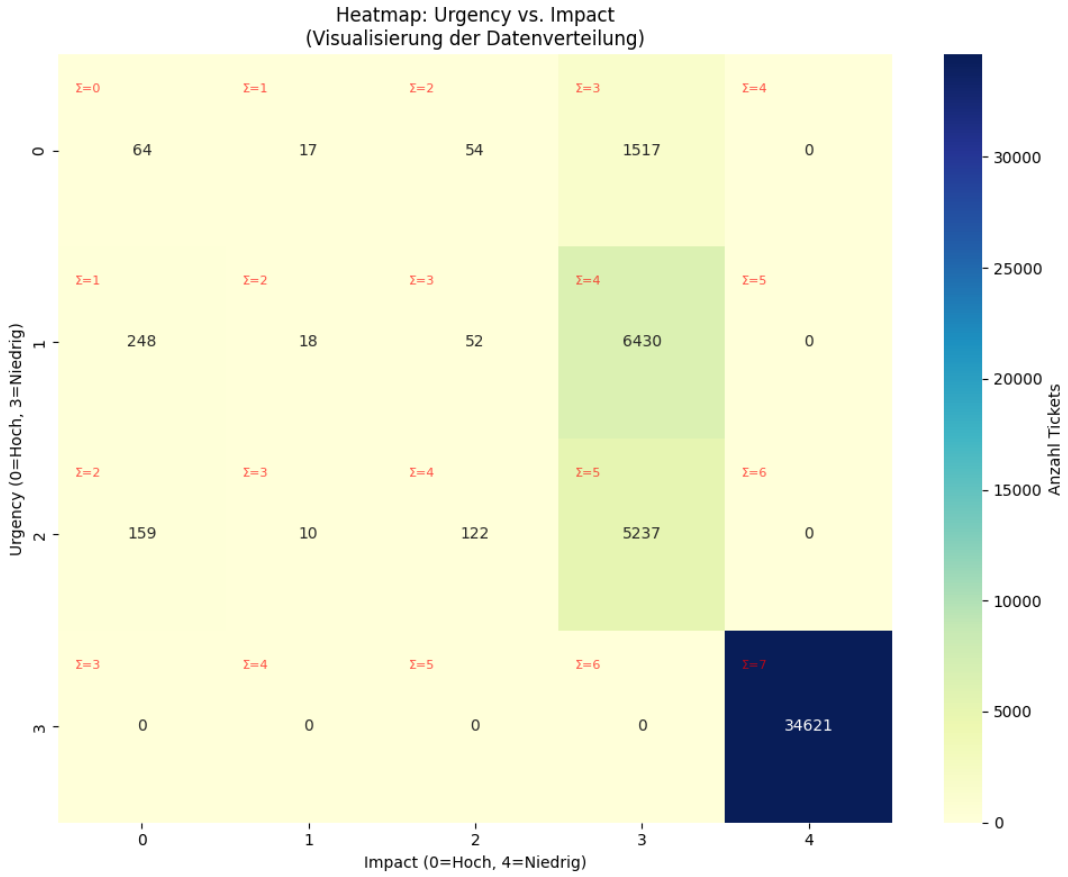
Trainingsdaten D_B:
„all_tickets.csv“, bereitgestellt auf der Kaggle Plattform von dem User Aniket. [59] Die Priorisierung liegt in Auswirkung (Fünf Klassen) und Dringlichkeit (Vier Klassen) vor.

---

[57] *Github, University of Helsinki*, Helsinki NLP, 2021, *Zugriff am 15.01.2026*
[58] Anhang: KI-Verzeichnis, 7. Skript-Verbesserung für Batch-Übersetzung, Turn 14, 2025, S. 1532–1551.
[59] *Aniket*, Support-tickets-classification, 2019, *Zugriff am 26.01.2026*

Stichproben haben ergeben, dass bei einem Großteil der Datensätze, häufig stichwortartig formulierte Tickets vorliegen. Die Reihenfolge der Stichworte bildet dennoch meist einen zusammenhängenden Kontext und ist dahingehend korrekt klassifiziert. Dies wird für das Training des DistilBERT-Modells als hinreichend eingestuft.

Abbildung 6: Visualisierung der Datenverteilung D_B



Quelle: Eigene Darstellung

Um die Datensätze ohne Informationsverlust in die relevanten 5 Prioritätsstufen zu skalieren, werden Dringlichkeit und Auswirkung von D_B ebenfalls den 5 Prioritätsstufen zugeordnet.

Tabelle 3: Harmonisierte Prioritätsmatrix D_B

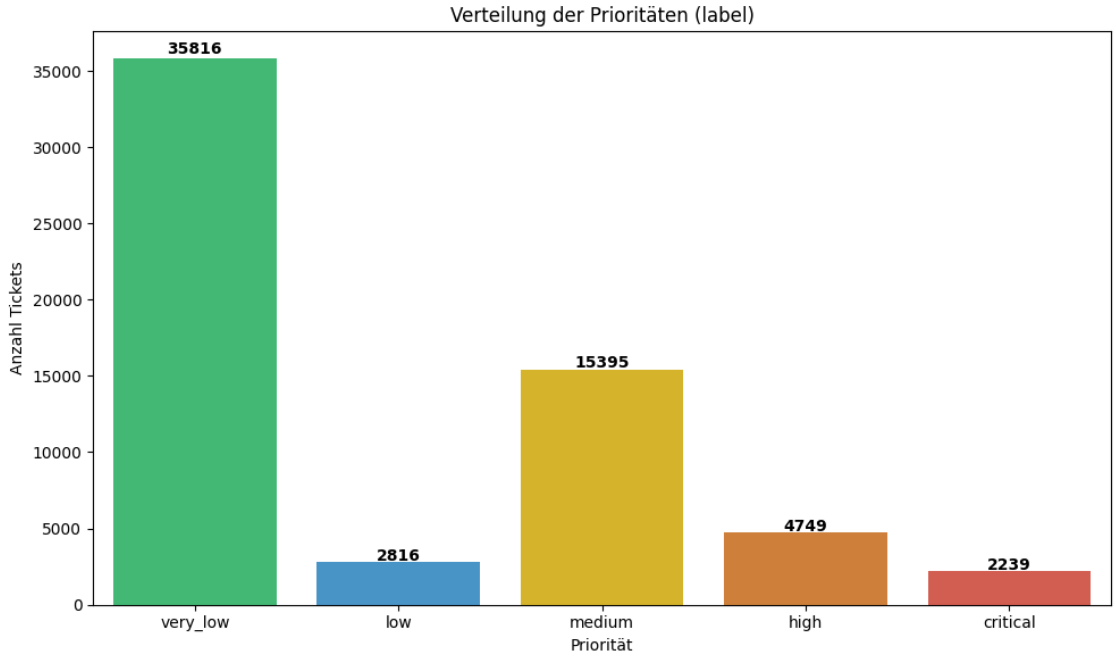
		Impact				
		4	3	2	1	0
Urgency	3	7 (very low)	6 (low)	5 (medium)	4 (medium)	3 (high)
	2	6	5	4	3	2 (high)
	1	5	4	3	2	1 (critical)
	0	4	3	2	1	0 (critical)

Quelle: Eigene Darstellung in Anlehnung an *Topalovic, D.*, Advisera, 2026

Verteilung der Prioritäten D_B:

- critical: 329 Tickets (0.68%)
- high: 1.810 Tickets (3.73%)
- medium: 11.789 Tickets (24.28%)
- low: 0 Tickets (0%)
- very_low: 34.621 Tickets (71.31%)

Abbildung 7: Verteilung der Prioritäten $D_{total} = D_A \cup D_B$



Quelle: Eigene Darstellung

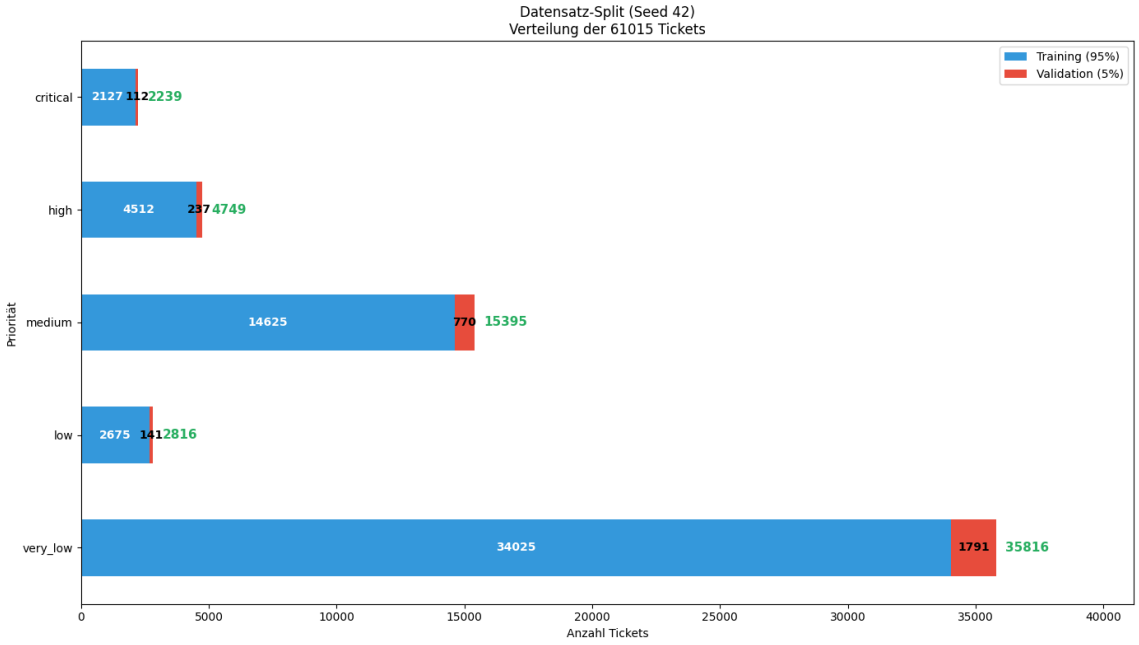
3.4.2. Die Evaluierung und Validierung

Um einen gerechten Vergleich des trainierten Modells mit generativen LLMs zu ermöglichen, wird das Trainingsdatenset in Trainings- und Validierungsdatenset unterteilt.

Je Prioritätsstufe werden jeweils 5% der vorliegenden Datensätze zufällig gewählt und für den Vergleich in Kapitel 3.4.4 beiseitegelegt. [60]

Die Validation-Datensätze (3.051 Tickets) werden nicht für das Training verwendet.

Abbildung 8: Datensatz-Split: $D_{total} \supset D_{Training}, D_{Validation}$



Quelle: Eigene Darstellung

Dies dient dazu, um einen fairen Vergleich mit den generischen Modellen sicherzustellen.

3.4.3. Das Training

Aufbauend auf der beschriebenen Datenbasis D_{total} mit 61.015 Datensätzen folgt nun die Darstellung des Trainingsprozesses, in welchem das DistilBERT-Modell mithilfe der aufbereiteten Tickets für die Klassifizierung der Priorisierung spezialisiert wird.

Das DistilBERT-Modell kennt noch nicht die spezifischen Prioritätenregeln eines IT-Servicedesk (z.B., dass ein „Serverausfall“ kritischer ist als ein „Passwortreset“), daher wird

---

60 *Google Developers*, Dataset Split, 2025, *Zugriff am 26.01.2026*

im Rahmen des Experiments der Kopf des neuronalen Netzes („Classification Head“) ausgetauscht und auf die fünf spezifischen Klassen trainiert. [61]

Dieser neue Klassifizierungslayer erlaubt dem Modell „Subject“ und „Body“ einer vorgegebenen Priorität zuzuordnen.

Beim Erstellen dieser Layer ist es erforderlich das Attribut Priority, welches sich wie in dem Abschnitt 2.1.4 beschrieben zusammensetzt, in Zahlenwerte zu konvertieren. [62]

Nachdem die Validierungsdatensätze beiseitegelegt wurden, bleiben noch 57.964 Ticketdatensätze ($D_{Training}$), welche für das Training zur Verfügung stehen. Diese werden nochmals aufgeteilt. Je Prioritätsklasse werden 20% von $D_{Training}$ für den Test des Trainingserfolges zwischen den Trainingsepochen beiseitegelegt.

Das Trainingsdatenset $D_{Training\text{-}Rest}$ enthielt dann noch 46.371 Tickets und das Test-Datenset D_{Test} 11.593 Tickets.

Effektiv fand das Training hierdurch mit 46.371 Datensätzen je Epoche statt.

Eine Epoche entspricht einer Trainingsiteration über alle Trainingsdaten. Nach jeder Epoche wurde mit Hilfe der Tickets D_{Test} der $F1_{Gesamt}$-Score berechnet, um den Zeitpunkt zu bestimmen, an dem das Modell mit den Trainingsdaten gesättigt und damit erfolgreich trainiert wurde. Um wie in 2.5 oben beschrieben Overfitting zu vermeiden, werden nach jeder Epoche die Evaluation-Loss Metrik, die „learning_rate“ bestimmt und ein Checkpoint erstellt. In darauffolgenden Epochen ist an diesen Metriken zu erkennen, ob Overfitting bereits eingetreten ist, um auf den letzten Checkpoint zurückzuspringen und damit das Training zu beenden.

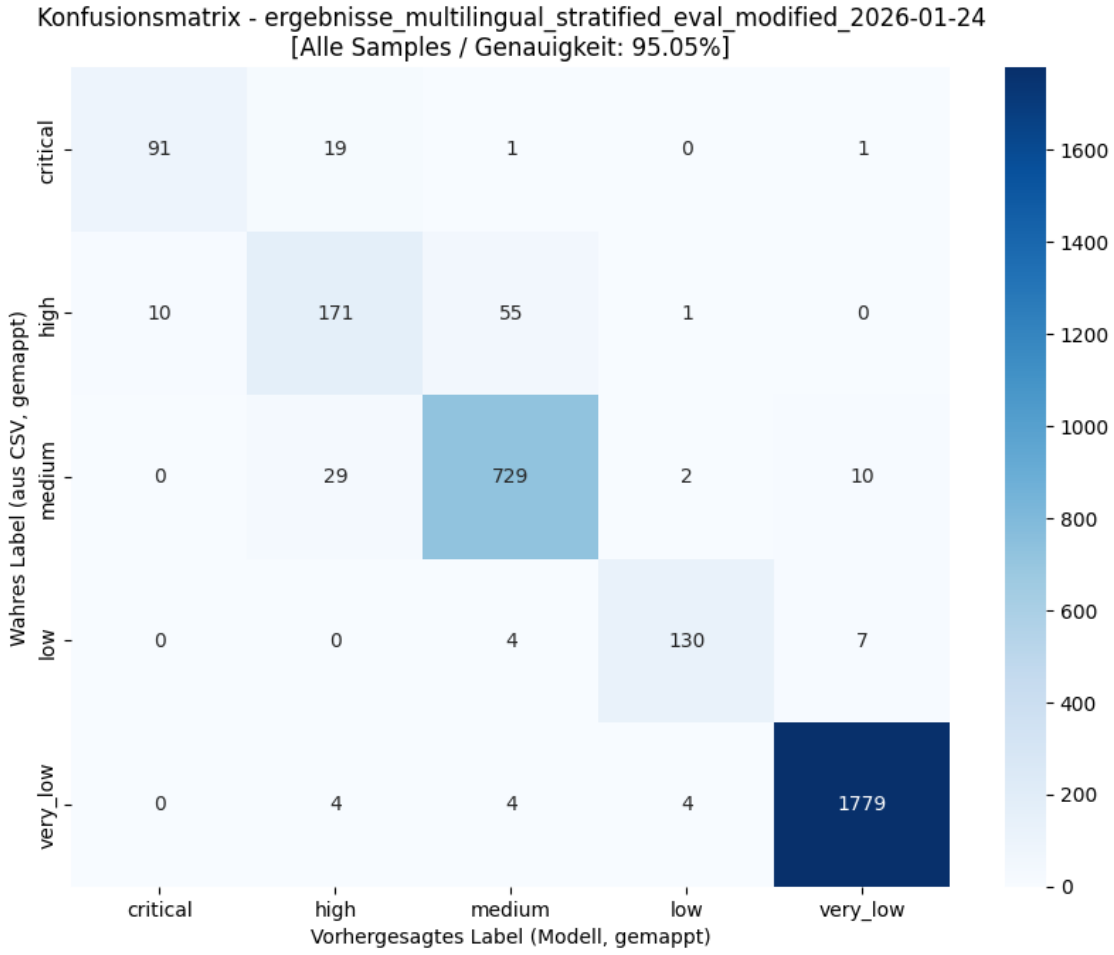
Abbildung 9: Evaluation-Loss Diagramm

Quelle: Eigene Darstellung

[61] Anhang: *KI-Verzeichnis*, 16. Jupyter und LLM-Training auf Pop!_OS, Turn 25, S. 2862–2863.
[62] Anhang: *KI-Verzeichnis*, 9. Scriptoptimierung für Trainingsevaluierung, S. 2216–2412.

Wie in Abbildung 9Abbildung 9: Evaluation-Loss Diagramm erkennbar ist, begann von Epoche 6 auf 7 die Evaluation-Loss Metrik zu steigen. Auch die F1-Metriken bestätigen, dass das Modell nach Epoche 6 keine nennenswerten Verbesserungen mehr erzielte und das Training so mit Abschluss der Epoche 6 und einem $F1_{Gesamt}$-Score von 0,95 (95%) beendet werden konnte.

Abbildung 10: DistilBERT - Konfusionsmatrix $D_{Validation}$



Quelle: Eigene Darstellung

3.4.4. Die Test-Pipeline

Für die Validierung der für den Vergleich herangezogenen Modelle, wurden die Ticket-Validierungsdatensätze verwendet welche vor dem Training (in 3.4.1 oben beschrieben) bei Seite gelegt wurden. Um eine Konfusionsmatrix zu erstellen, wurden mit Hilfe von *Gemini Pro* Python Scripts erstellt, mit denen die Datensätze nacheinander einzeln durch

die verschiedenen generativen LLM-Modelle qualifiziert wurden. [63] Hierbei wurde die Spalte, welche die Prioritäten enthält, verschleiert und nicht an das geladene Modell übergeben.

Die geladenen Modelle wurden alle im Zero-Shot-Ansatz mit dem folgenden Systemprompt und Userprompt angesprochen.

Systemprompt:

„You are an agent responsible for qualifying and prioritizing ITSM tickets. You classify the ticket based on urgency and impact into: very low [1], low [2], medium [3], high [4], critical [5]. You output ONLY the numerical value (e.g. [1]) assigned by your qualification."

Userprompt:

„You are an agent responsible for qualifying and prioritizing ITSM tickets. You receive information about the subject and the message. You classify the ticket based on this information into the following priority levels, which are determined by urgency and impact: very low [1], low [2], medium [3], high [4], critical [5]. You output only the numerical value assigned by your qualification. This is your input:
Subject: {subject}
Message: {body}"

„{subject}" und „{body}" wurden hierbei jeweils durch den Betreff und Nachrichtenkörper des jeweiligen Datensatzes ausgetauscht. [64]

Damit der gesamte Kontext der Nachricht durch die generativen KI verarbeitet werden konnte, musste die Anzahl der zur Verfügung stehenden Input-Token an die Kontextlänge der Trainingsdatensätze pauschal auf die Anzahl der maximal notwendigen Input-Token der Trainingsdatensätze (in Summe 215 Input-Token) auf die Länge des Systemprompt + Userprompt angepasst werden. Die Wahl viel hierbei auf das 1,5-fache (322 Token).

---

63 Anhang: *KI-Verzeichnis*, 1. Finale Projektdurchführung, Turn 2, 2025, S. 18–28.
64 Vgl. Anhang: *Batch-Klassifizierungs-Script-LLMs*, Zeile 28–37.

Tabelle 4: Übersicht der Validierungsergebnisse $D_{Validation}$

Model	Accuracy	Errorrate	Tatsächlich bewertete Tickets	F1-Score					
				very low	low	medium	high	critical	Macro Average F1
NEUES MODELL (DistilBERT)	0,95	0,00	3.051	0,99	0,94	0,93	0,74	0,85	0,89
llama-4-scout-17b-16e-instruct	0,33	0,03	2.956	0,54	0,13	0,19	0,13	0,55	0,31
qwen_qwen2.5-coder-14b	0,28	0,01	3.017	0,17	0,06	0,50	0,24	0,54	0,30
phi-3-large-5.6b	0,25	0,45	1.691	0,05	0,13	0,47	0,33	0,53	0,30
google_gemma-2-9b	0,25	0,02	3.002	0,04	0,09	0,58	0,33	0,51	0,31
falcon3-3b-instruct	0,22	0,01	3.016	0,29	0,10	0,27	0,19	0,38	0,25
mistral_latest	0,20	0,02	2.977	0,02	0,07	0,31	0,18	0,51	0,22
falcon3-1b-instruct	0,19	0,05	2.912	0,33	0,05	0,03	0,06	0,24	0,14
gemma-1b	0,19	0,91	281	0,40	0,03	0,02	0,00	0,22	0,13
darkidol-llama-3.1-8b-instruct-1.2-uncensored	0,16	0,01	3.015	0,00	0,09	0,42	0,18	0,55	0,25

Quelle: Eigene Darstellung

Obwohl es sich um kleine LLMs handelt, welche für den Vergleich verwendet wurden, hat die Validierung auf Grund fehlender Optimierung, insbesondere bei den größeren Modellen, sehr lange (mehrere Stunden) gedauert.

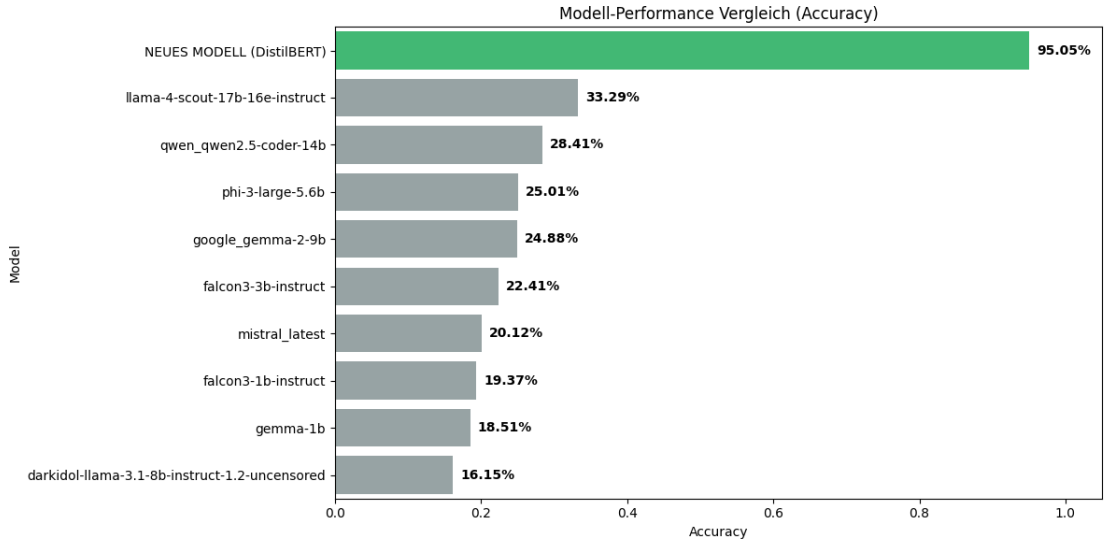
In der Geschwindigkeit liegt DistilBERT mit einer Validierungsdauer der 3.051 Validierungsdatensätzen von 30,47 Sekunden weit vor den LLMs und ist damit bedeutend schneller.

Das Training des DistilBERT Modells hat bis Epoche 10 insgesamt 23 Minuten und 31 Sekunden gedauert. [65]

Trotz der geringen Trainingszeit erreicht das trainierte Modell eine sehr viel höhere Genauigkeit als das Größte für den Vergleich herangezogene Generative Modell Llama4-Scout-Instruct (Tabelle 4: Übersicht der Validierungsergebnisse $D_{Validation}$).

[65] Anhang: *Trainings-Log*

Abbildung 11: Modell-Performance Vergleich Genauigkeit



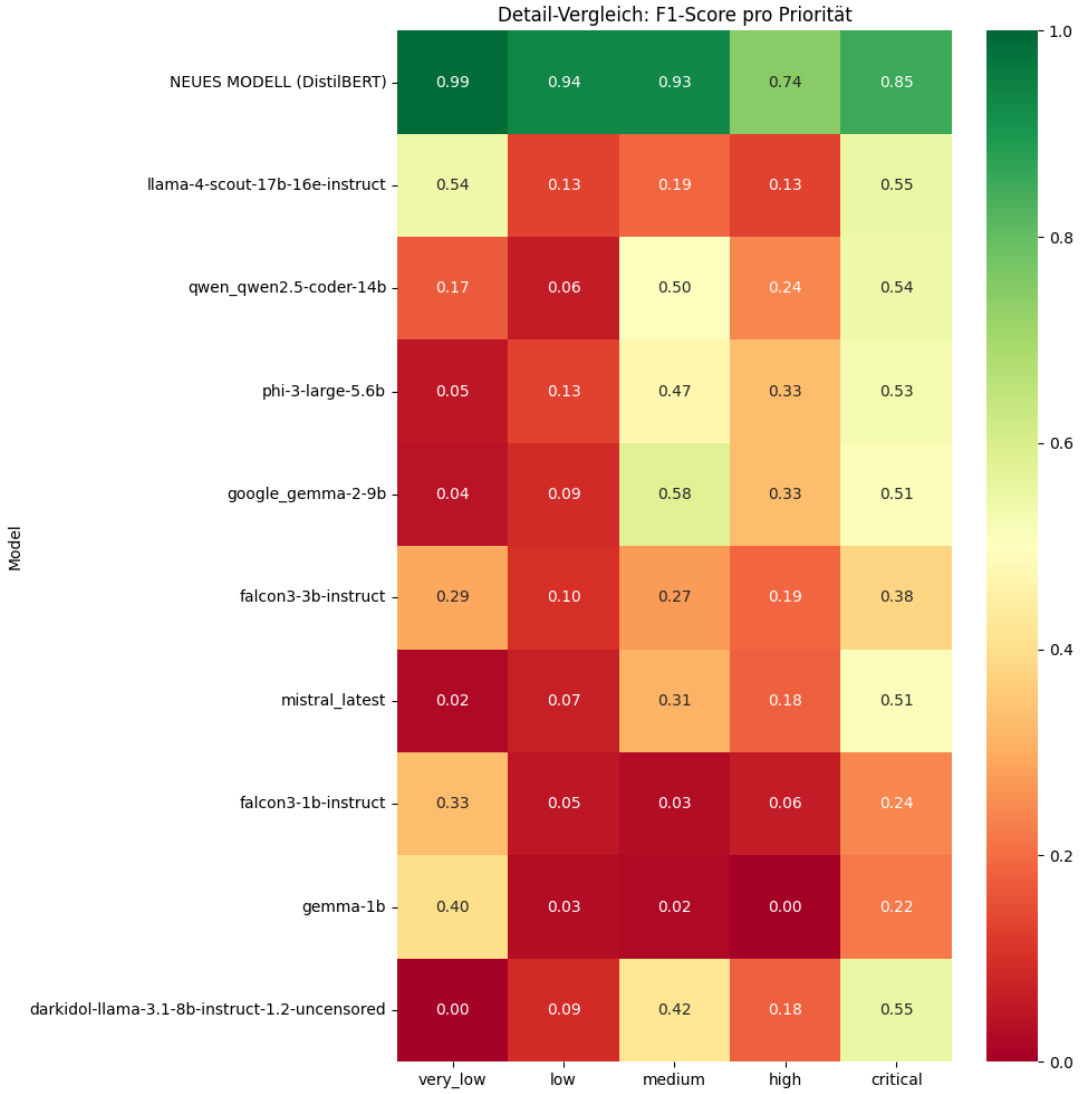
Quelle: Eigene Darstellung

3.5. Ergebnisse

In diesem Abschnitt geht es um die Beantwortung der in Kapitel 1.2 gestellten Forschungsfragen. Dafür wird auf die gesammelten Ergebnisse der durchgeführten Vergleiche zurückgegriffen und die F1-Scores und Konfusionsmatrizen der Modelle dargelegt, sodass man mit einem Blick auf die visuellen Darstellungen die Leistungsunterschiede ablesen kann.

Die erste in Kapitel 1.2 gestellte Forschungsfrage lässt sich mit den in Abbildung 12 dargestellten $F1_{Priorität}$-Scores beantworten.

Abbildung 12: F1-Score aller Modelle pro Priorität



Quelle: Eigene Darstellung

Hier zu sehen sind die $F1_{Priorität}$-Scores aller getesteten Modelle, gruppiert nach der Priorität der Tickets. Dadurch ist zu erkennen, wie die Modelle beim Kategorisieren einzelner Ticket-Prioritäten abgeschnitten haben.

Dabei ist anzumerken, dass bei der Berechnung der $F1_{Priorität}$-Scores der Modelle eine Besonderheit im Umgang mit ungültigen Antworten zu beachten ist. Da die generalistischen LLMs im Zero-Shot-Verfahren teilweise Antworten generieren, die nicht dem in Kapitel 3.4.4 geforderten Zielformat entsprechen, muss die obige Grafik unbedingt im Zusammenhang mit der entsprechenden validen Antwortquote betrachtet werden, um den $F1_{Priorität}$-Scores die korrekte Aussagekraft zu verleihen.

Denn die Berechnung der $F1_{Priorität}$-Scores der LLMs bezieht sich ausschließlich auf die Teilmenge der syntaktisch validen Antworten der LLMs. Das bedeutet: Ein Modell, das nur wenige, aber dafür einfache Tickets korrekt beantwortet und bei komplexen Fällen ein ungültiges Format liefert, kann einen allgemein künstlich erhöhten F1-Score erzielen (Survivorship Bias).

Um diese Verzerrung aufzuklären und die Ergebnisse korrekt einordnen zu können, muss zusätzlich zum $F1_{Priorität}$-Score die Response Rate (Antwortquote) betrachtet werden. Sie gibt an, für wie viel Prozent des Testdatensatzes das Modell überhaupt eine verwertbare Klassifizierung abgegeben hat. Dies muss ungeachtet dessen geschehen, ob es sich bei den falsch formatierten Antworten tatsächlich um einen Survivorship Bias des Modells handelt, oder nur um eine eingeschränkte Fähigkeit sich an vordefinierte Ausgabeformate zu halten.

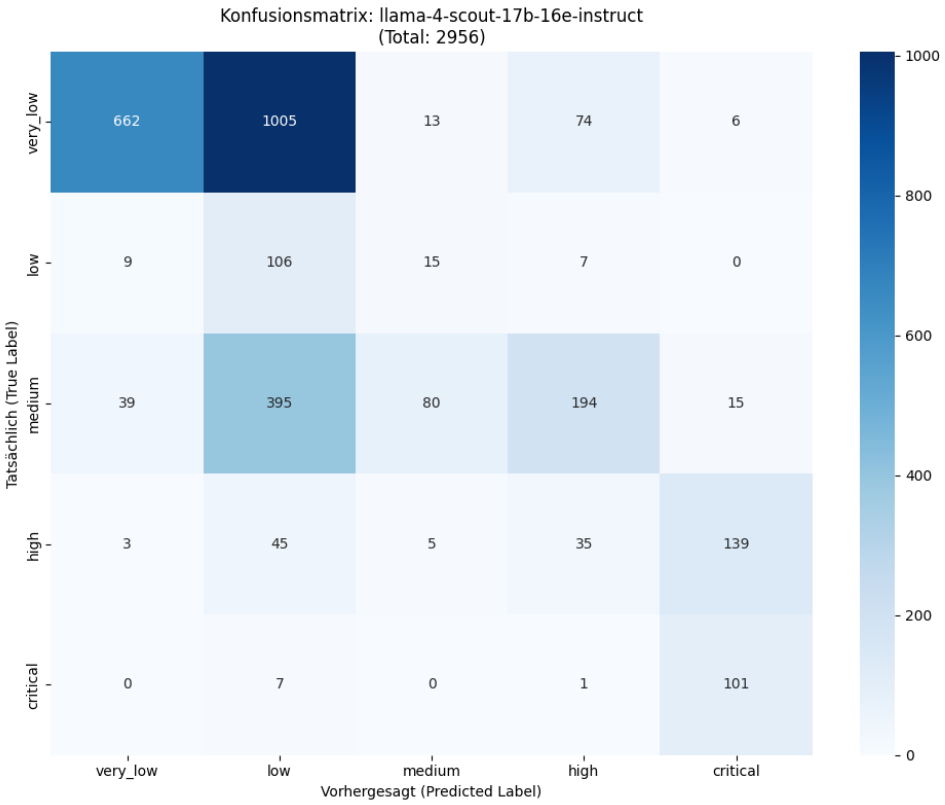
Die angesprochene Antwortquote aller Modelle ist der Spalte „Errorrate" aus Tabelle 4 zu entnehmen. Dabei stellt die Errorrate das Komplement zur obig beschriebenen Antwortquote dar (100 % abzüglich der Antwortquote). Da alle LLMs – außer „phi-3-large-5.6b" und „gemma-1b" – eine Errorrate von $\leq 5\%$ aufweisen, sind die Ergebnisse dieser LLMs durchaus zum Vergleich untereinander und mit DistilBERT verwendbar.

In diesem Kontext kann man nun der Abbildung 12 entnehmen, dass das in dieser Seminararbeit antrainierte DistilBERT-Modell deutlich bessere Ergebnisse in allen Prioritäten erbringt als alle zum Vergleich herangezogenen LLMs.

Die Gesamt-Performance der Modelle können miteinander vergleichen werden, indem man die Metrik des $F1_{Macro\_Average}$-Scores aus der entsprechenden Spalte der Tabelle 4 betrachtet. Diese Metrik ist lediglich das arithmetische Mittel der fünf $F1_{Priorität}$-Scores eines Modells.

Folglich ist noch exemplarisch die Konfusionsmatrix des „Llama 4 scout"-Modells dargestellt, da es unter den LLMs das beste Ergebnis im Hinblick auf den $F1_{Macro\_Average}$-Score aus Tabelle 4 erzielt hat.

Abbildung 13: Llama 4:17b Konfusionsmatrix



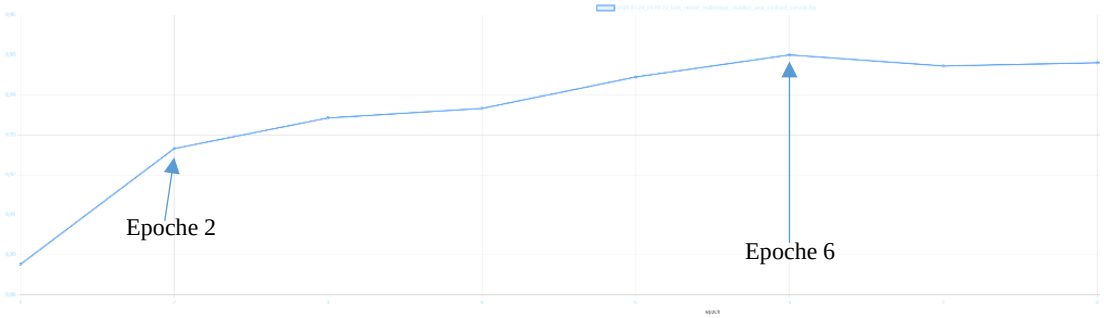
Quelle: Eigene Darstellung

Die Konfusionsmatrizen der anderen LLMs sind im Anhang unter „Quelltexte/#5 Generative Modelle/stats/[Modellname]" zu finden. Hinweis: Die Konfusionsmatrix von DistilBERT ist in Abbildung 10 zu sehen.

Die Antwort zur zweiten in Kapitel 1.2 gestellten Forschungsfrage, lässt sich aus den Kapiteln 3.4.2 und 3.4.3 ableiten.

Die erforderliche Anzahl an Iterationen für ein optimales Ergebnis betrug 6 Epochen, weil die Analyse der Hyperparameter (Evaluation-Loss und Learning Rate) zeigte, dass ab dem Übergang von Epoche 6 zu 7 der Evaluation-Loss anstieg, was den Beginn von Overfitting signalisierte.

Abbildung 14: DistilBERT F1-Score in Training



Quelle: Eigene Darstellung

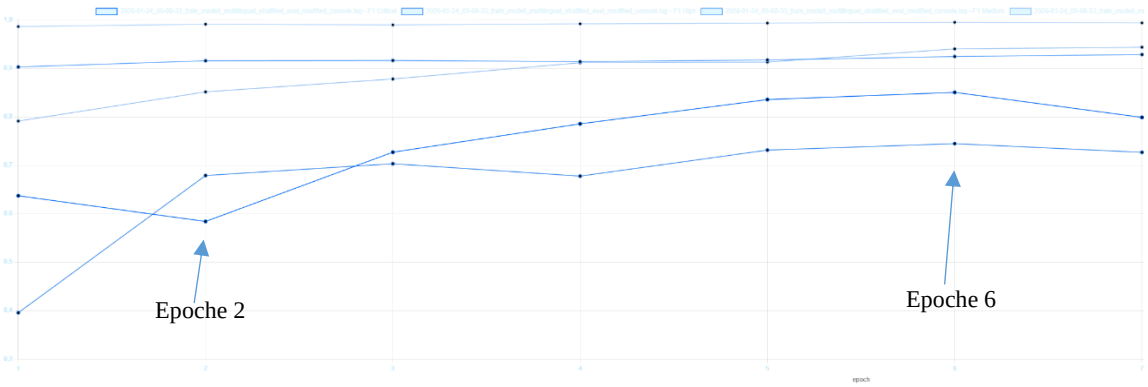
Gleichzeitig stagnierten, wie in Abbildung 14 zu sehen ist, die F1-Metriken, sodass nach Epoche 6 mit einem F1-Gesamtwert von 0,95 (95%) der Sättigungspunkt erreicht war und das Training beendet wurde.

Bei näherer Betrachtung der Genauigkeit über die gesamten Epochen hinweg, wurde faktisch nach der zweiten Trainingsepoche bereits eine Genauigkeit von 92,7% erreicht.

Hierbei muss erwähnt werden, dass sich dieses Ergebnis auf den $F1_{Weighted}$-Score wie in Abbildung 14 zu sehen bezieht.

Die Fähigkeit kritische Tickets auch als kritisch zu erkennen war in dieser Epoche mit 58,4% noch niedrig (Abbildung 15), hat sich durch weitere Trainingsiterationen bis zur Epoche 6 auf 74,5% erhöht.

Abbildung 15: DistilBERT Per Class F1-Score



Quelle: Eigene Darstellung

4. Fazit und Ausblick

4.1. Zusammenfassung der Ergebnisse

Das DistilBERT-Modell übertrifft mit der qualitativen Metrik der F1-Genauigkeit die in diesem Experiment getesteten LLMs in allen Prioritätsstufen. Es wird ein besseres Ergebnis in der Klassifizierungs- und Priorisierungsgüte von ITSM-Tickets erreicht, als nicht für diesen Zweck trainierte generische LLMs im Zero-Shot Verfahren erreichen. Die Erste Forschungsfrage kann mit diesem Experiment positiv beantwortet werden.

Die zweite Forschungsfrage, über den Umfang der nötigen Trainingsdaten, sowie Trainingsiterationen kann ebenfalls beantwortet werden. Es sind mindestens 2 Epochen Training erforderlich, um eines der Vergleichsmodelle zu übertreffen. Maximal 6 Epochen können mit 57.964 Ticketdatensätzen trainiert werden, bevor Overfitting einsetzt. In der sechsten Epoche setzt sich somit das maximal mit dieser Menge an Trainingsdaten mögliche Ergebnis fest.

4.2. Kritische Würdigung

Beim Durchführen dieser Arbeit ergaben sich einige Limitationen/Eigenschaften, weshalb die Ergebnisse dieser Arbeit unter dem Licht der nachstehenden Punkte mit Vorsicht zu genießen sind:

- Informationsverlust durch maschinelle Übersetzung:
  Da das verwendete DistilBERT-Modell maßgeblich auf englischsprachigen Daten trainiert wurde und seine optimale Leistungsfähigkeit folglich bei der Verarbeitung englischer Texte entfaltet, war eine Übersetzung der ursprünglich 13.178 deutschen ITSM-Tickets erforderlich. Diese übersetzten Datensätze machen im gesamten genutzten ITSM-Ticket-Datensatz ungefähr einen Anteil von 21,35% aus.

  Dieser notwendige Vorverarbeitungsschritt (mittels „Helsinki-NLP/opus-mt-de-en“) führt jedoch eine zusätzliche Ungenauigkeit hinzu, welche zu verzerrten Ergebnissen führen kann. Trotz der hohen Qualität moderner Übersetzungsmodelle ist nicht auszuschließen, dass bei der Übertragung von deutschem IT-Fachjargon ins Englische feine semantische Nuancen verloren gegangen sind oder fehlinter-

pretiert wurden. Aufgrund des enormen Aufwandes wurden die 13.178 übersetzten Datensätze nur stichprobenartig auf eine ordentliche Übersetzung geprüft. Die Ergebnisse der Arbeit unterliegen somit der Einschränkung, dass die Priorisierung der Tickets in diesen Fällen auf einer englischen Repräsentation basiert, was potenziell zu Abweichungen gegenüber einer Bewertung des deutschen Originaltextes führen kann.

- Domänenspezifischer Bias:
  Die zur Durchführung dieses Experiments verwendeten Ticket-Datensätze beinhalten zwangsweise einen gewissen domänenspezifischen Kontext. Das hier verwendete DistilBERT-Modell wurde auf einem Teil dieser Datensätze trainiert und auf einem anderen Teil getestet. Es ist daher nicht auszuschließen, dass die gemessene Performance zugunsten von dem DistilBERT-Modell verzerrt ist, da es sich stärker an die spezifischen Terminologien der Datensätze anpassen konnte als die zum Vergleich herangezogenen LLMs.
- Limitierte Datengrundlage:
  Die absolute Anzahl der verfügbaren Trainingsdaten war begrenzt. Da die Leistungsfähigkeit von Deep-Learning-Modellen stark mit der Menge der Trainingsdaten korreliert, repräsentieren die hier erzielten Ergebnisse vermutlich nicht das volle Potenzial des DistilBERT-Modells. Eine Vergrößerung des Datensatzes sowie eine höhere Heterogenität der Daten (Tickets aus verschiedenen Unternehmenskontexten) könnten die Ergebnisse signifikant verbessern.
- Mangelnde Datenverifizierung:
  Bei den verwendeten Ticket-Datensätzen wurde vorausgesetzt, dass sie in einer korrekten Priorisierung vorliegen und auch die Titel und Nachrichteninhalte der Tickets inhaltlich schlüssig sind. Diese Voraussetzungen wurden bei den verwendeten Ticket-Datensätzen lediglich Stichprobenartig überprüft. Ansonsten wurden die genannten Voraussetzungen als gegeben betrachtet, da eine manuelle Überprüfung des gesamten Datensatzes für den Umfang dieser Arbeit als zu groß evaluiert wurde.

4.3. Ausblick

Die vorliegende Arbeit hat gezeigt, dass ein spezialisiertes, durch Fine-Tuning optimiertes Encoder-Modell die Effizienz der ITSM-Ticketklassifizierung massiv steigern kann. Dennoch ergeben sich aus den gewonnenen Erkenntnissen verschiedene Anknüpfungspunkte für zukünftige Untersuchungen:

Praktische Prozessintegration: Eine zentrale Frage für die Zukunft ist die Einsetzbarkeit des Modells als Vorklassifizierungs-Agent im Live-Betrieb, um die manuelle Last am Service Desk weiter zu reduzieren.

Optimierung der Sprachbasis: Da die Verwendung von Übersetzungsmodellen potenzielle semantische Verluste mit sich bringt, wäre die Erprobung nativ deutschsprachiger Modelle ein logischer nächster Schritt, um den Einfluss von Fachjargon präziser abzubilden.

Erweiterung der Datenheterogenität: Zukünftige Forschung sollte untersuchen, ob die Einbeziehung noch vielfältigerer Datensätze aus unterschiedlichen Unternehmenskontexten die Robustheit des Modells gegenüber variierenden Ticket-Stilen weiter verbessern kann.

Hybrid-Ansätze: Es wäre zu prüfen, inwiefern eine Kombination aus der hohen Klassifizierungsgeschwindigkeit von DistilBERT und der generativen Stärke großer LLMs (z.B. für die automatische Erstellung von Lösungsvorschlägen) einen Mehrwert für das Incident Management bietet.

Anhang

KI-Verzeichnis:

„KI-Verzeichnis/KI-Verzeichnis_2026-01-25.pdf“

Konfusionsmatrizen der Vergleichsmodelle:

„Quelltexte/#5 Generative Modelle/stats/[Modellname]“

Datenanalyse-Gewichtungen:

„Quelltexte/#2 Aufbereiten, Übersetzen und Harmonisieren/analyse_csv.py“

Batch-Datenübersetzung:

„Quelltexte/#2 Aufbereiten, Übersetzen und Harmonisieren/translate_csv.py“

Übersetzter Trainingsdatensatz D_A:

„Quelltexte/#2 Aufbereiten, Übersetzen und Harmonisieren/dataset-tickets-german_normalized_50_5_2_Translate_Helsinki_DE-EN.csv“

Datenharmonisierung und Merge:

„Quelltexte/#2 Aufbereiten, Übersetzen und Harmonisieren/datenharmonisierung.py“

5% Stratifizierungs-Script:

„Quelltexte/#3 Split - Evaluierung, Validierung/split_dataset_stratified.py“

5% Validierungsdatensatz:

„Quelltexte/#3 Split - Evaluierung, Validierung/helsinki_final_training_dataset_harmonized_val_5.csv“

95% Trainingsdatensatz:

„Quelltexte/#3 Split - Evaluierung, Validierung/helsinki_final_training_dataset_harmonized_train_95.csv“

DistilBERT Evaluierungs-Script:

„Quelltexte/#3 Split - Evaluierung, Validierung/train_modell_multilingual_stratified_eval_modified.py“

Chat-Script:

„Quelltexte/#6 Empirischer Vergleich/chat_with_modell.py“

Batch-Klassifizierungs-Script-LLMs:

„Quelltexte/#6 Empirischer Vergleich/generic_classification_withAPIKey_safe.py“

Modell-Vergleichsauswertung-Script:

„Quelltexte\#6 Empirischer Vergleich\compare_models_results.py“

Trainings-Log:

„Quelltexte/#4 ergebnisse_multilingual_stratified_eval_modified_2026-01-24/2026-01-24_05-08-33_train_modell_multilingual_stratified_eval_modified_console.log“

Literaturverzeichnis

Abdin, Marah, Aneja, J., Awadalla, H., et al. (Phi-3, 2024): Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone, In: arXiv, 2024

Agutter, Claire (Incident Management, 2020): ITIL Foundation Essentials ITIL 4 Edition – The ultimate revision guide, 2. Aufl., Cambridgeshire: IT Governance Publishing, 2020

Axelos Ltd. (ITIL 4 Edition, 2019): ITIL Foundation ITIL 4 Edition, Nikosia: Peoplecert International Ltd., 2019

Al-Hawari, Feras, Barham, H. (machine learning based helpdesk, 2021): A machine learning based help desk system for IT service management, In: Journal of King Saud University – Computer and Information Sciences, Band 33, 2021, S.702 – 718

Bengio, Yoshua, Simard, P., Frasconi, P. (Gradient Descent, 1994): Learning Long-Term Dependencies with Gradient Descent is Difficult, In: IEEE Transactions on neural Networks, Volume 5, No. 2, 1994, S. 157–166

Bengio, Yoshua, Rejean, D., Pascal, V. (NPL Model, 2003): A Neural Probabilistic Language Model, In: Journal of Machine Learning Research 3, 2003, S. 1137–1155

Chen, Z., Kang, Y., Li, L., Zhang, X., Zhang, H., Xu, H., Zhou, Y., Yang, L., Sun, J., Xu, Z. (Software Engineering Conference, 2020): Towards intelligent incident management: why we need it and how we make it, In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, S. 1487–1496

Goodfellow, Ian, Bengio, Y., Courville, A. (Deep Learning, 2016): Deep Learning, In: Cambridge Massachusetts, MIT Press, 2016

Devlin, Jacob, Chang, M., Lee, K., Toutanova, K. (BERT Pre-training, 2019): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, In: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1, 2019, S. 4171–4186

Eichstädt, Timm, Spieker, S. (Support, 2024): 52 Stunden Informatik – Was jeder über Informatik wissen sollte, 2. Aufl., Wiesbaden: Springer, 2024

Gupta, Rajeev, Prasad, K. H., Mohania, M. (Incident classification, 2008): Automating ITSM Incident Management Process, In: International Conference on Autonomic Computing, 2008, S. 141–150

Grattafiori, Aaron, Dubey, A., Jauhri, A., et al. (Llama 3, 2024): The Llama 3 Herd of Models, In: arXiv, 2024

Hevner, Alan R., Chatterje, S. (Design Science Research Methodology, 2010): Design Research in Information Systems – Theory and Practice, In: Integrated Series in Information Systems, Band 22, Springer: New York, 2010

Hevner, Alan R., March, S., Park, J., Ram, S. (DSISR, 2004): Design Science in Information Systems Research, In: MIS Quarterly, Volume 28 No. 1, 2004, S. 75–105

Hochreiter, Sepp, Schmidhuber, J. (LSTM, 1997): Long Short-Term Memory, In: Neural Computation, Vol. 9(8), 1997, S. 1735–1780

Grandini, Margherita, Bagli, E., Visani, G. (Multi-Class Metrics, 2020): Metrics for Multi-Class Classification: An Overview, In: arXiv, 2020

Jiang, Albert Q., Sablayrolles, A., Mensch, A., et al. (Mistral, 2023), Mistral 7B, In: arXiv, 2023

Österle, Hubert, Becker, J., Frank, U., Hess, T., Karagiannis, D., Krcmar, H., Loos, P., Mertens, P., Oberweis, A., Sinz, E. J. (Gestaltungsorientierte Wirtschaftsinformatik, 2010): Memorandum zur gestaltungsorientierten Wirtschaftsinformatik, In: Schmalenbachs Zeitschrift für betriebswirtschaftliche Forschung, Band 62, 2010

Pilogret, Lionel (Managing IT, 2025): Managing IT in einer digitalen Welt – Zusammenhänge der IT verstehen und die Kriterien für eine erfolgreiche Transformation beherrschen, Wiesbaden: Springer, 2025

Pfitzinger, Bernd, Jestädt, T. (IT-Betrieb, 2016): IT-Betrieb – Management und Betrieb der IT in Unternehmen, Berlin: Springer, 2016

Sanh, Victor, Debut, L., Chaumond, J., Wolf, T. (DistilBERT Working-Paper, 2020): DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, In: arxiv, 2020

Talaat, Amira S. (DistilBERT classification methods, 2025): A novel double- and triple-BERT and DistilBERT classification methods, In: Neural Computing and Applications, 2025, Jahrgang 37, S. 25923–25944

Vaswani, Ashish, Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (Attention, 2017): Attention Is All You Need, In: 31st Conference on Neural Information Processing Systems, 2017

Villareal, Alessandro V., Garcia, E. V., De Los Santos, A. C. M. (Literature Review, 2022): Machine Learning to automate support ticket systems: A literature review, In: Revista Campus, Jahrgang 27, Nr.34, 2022

Yu, Qingyang, Zhao, N., Li, M., Li, Z., Wang, H., Zhang, W., Sui, K. & Pei, D. (Artificial Intelligence to enhance IT Operations, 2024): A survey on intelligent management of alerts and incidents in IT services, In: Journal of Network and Computer Applications, Band 224, 2024, Artikelnr. 103842

Internetquellen

Aniket (Support-tickets-classification, 2019): „all_tickets.csv“ <https://www.kaggle.com/datasets/aniketg11/supportticketsclassification> [Zugriff am 26.01.2026]

Bueck, Tobias (Alternative Dataset, 2025): Customer IT Support – Ticket Dataset, 2025. <https://www.kaggle.com/dsv/13959267> [Zugriff am 15.01.2026]

Bueck, Tobias (Ticket Dataset, 2025): „dataset-tickets-german_normalized_50_5_2.csv“ <https://www.kaggle.com/datasets/tobiasbueck/multilingual-customer-support-tickets?resource=download&select=dataset-tickets-german_normalized_50_5_2.csv> [Zugriff am 18.01.2026]

HDI (State of Technical Support, 2024): The State of Technical Support in 2024: HDI Practices & Salary Presentation, 2024 <https://www.thinkhdi.com/library/research> [Zugriff am 28.12.2025]

Intercom (Service Trends Report, 2024): Intercom Customer Service Trends Report 2024, S. 14 ff. <https://www.intercom.com/blog/customer-service-trends-report-2024/> [Zugriff am 10.01.2026]

Github, Google (google-research_bert, 2020): bert <https://github.com/google-research/bert> [Zugriff am 15.01.2026]

Github, University of Helsinki (Helsinki NLP, 2021): Helsinki-NLP Opus-MT <https://github.com/Helsinki-NLP/Opus-MT?tab=readme-ov-file> [Zugriff am 15.01.2026]

Github, Żak, Karol (Impact Dataset, 2019): Support-tickets-classification. <https://github.com/karolzak/support-tickets-classification#22-dataset> [Zugriff am 15.01.2026]

Google Cloud (Deep Learning auf Google Cloud, 2026): Was ist Deep Learning? <https://cloud.google.com/discover/what-is-deep-learning?hl=de> [Zugriff am 18.01.2026]

Google Developers (Dataset Split, 2025): Datasets: Ursprüngliches Dataset unterteilen <https://developers.google.com/machine-learning/crash-course/overfitting/dividing-datasets?hl=de> [Zugriff am 26.01.2026]

Huggingface (bert-base-uncased, 2023): BERT base model (uncased) <https://huggingface.co/google-bert/bert-base-uncased> [Zugriff am 15.01.2026]

Huggingface (distilbert_distilbert-base-uncased, 2019): DistilBERT base model (uncased)

<https://huggingface.co/distilbert/distilbert-base-uncased>
[Zugriff am 15.01.2026]

Sagar, Dingu (Knowledge Distillation, 2021): Knowledge Distillation in Deep Learning – DistilBERT Explained
<*https://www.youtube.com/watch?v=rNOuDKWtrAE*>
[Zugriff am 25.01.2026]

Topalovic, Drago (Advisera, 2026): All about Incident Classification <https://advisera.com/20000academy/knowledgebase/incident-classification/>
[Zugriff am 25.01.2026]