Hochschule für angewandte Wissenschaften Coburg
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik

Bachelorarbeit

Multi-Modal Feature Fusion with Cross-Attention for Tabular and Textual Data

Städler, Sebastian

Abgabe der Arbeit: 05.01.2026

Betreut durch:

Prof. Dr. Roman Rischke, Hochschule Coburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codebeispielverzeichnis

Abkürzungsverzeichnis

BERT	Bidirectional Encoder Representations from Transformers
ELMo	Embeddings from Language Models
NLP	Natural Language Processing
TF-IDF	Term Frequency-Inverse Document Frequency

1 Einführung

Das ist meine Einfuehrung ...

2 Theoretische Grundlagen

2.1 Multimodale Daten

Eine *Modalität* bezeichnet eine eigenständige Form der Informationsrepräsentation mit spezifischer statistischer Struktur und Verarbeitungslogik (z. B. sequenzieller Text vs. tabellarische Attribute). Multimodales Lernen nutzt diese unterschiedlichen und sich ergänzenden Perspektiven auf denselben Sachverhalt, erfordert jedoch spezialisierte Strategien zur gemeinsamen Modellierung [BAM19].

2.1.1 Heterogenität in multimodalen Daten

Die Kombination von Text- und Tabellendaten stellt hierbei eine besondere Herausforderung dar. Während Text linguistisch strukturiert ist und Semantik aus der Reihenfolge von Tokens gewinnt, sind Tabellen attributbasiert und nicht-sequenziell. Diese strukturelle Diskrepanz führt zu einem „Manifold Mismatch“ [BAM19, HKCK20]: Text wird in hochdimensionalen, kontinuierlichen Embedding-Räumen repräsentiert, während Tabulardaten aus heterogenen, unskalierten Merkmalen bestehen. Die geometrische Inkompatibilität dieser Räume verhindert, dass Attention-Mechanismen, die auf Text-Tokens effektiv angewendet werden können, direkt auf rohe Tabularfeatures funktionieren.

Die zentrale Herausforderung besteht somit darin, diese strukturelle Lücke zu überbrücken. Bevor eine Fusion technisch möglich ist, müssen beide Modalitäten in einen kompatiblen, gemeinsamen Vektorraum projiziert werden. Dies erfordert spezifische Repräsentationsstrategien: Während für Text etablierte Tokenisierungsverfahren existieren, müssen für tabellarische Daten erst analoge Methoden der Tokenization angewendet werden, um sie in eine verarbeitbare Sequenzform zu überführen. Die folgenden Abschnitte erläutern die Umsetzung dieser Repräsentationen für Text (Abschnitt 2.1.2) und Tabellen (Abschnitt 2.1.3) im Detail.

2.1.2 Repräsentation von Textdaten

Für die effektive Verarbeitung natürlicher Sprache (Natural Language Processing (NLP)) und die in dieser Arbeit untersuchte Fusion von Text- und Tabellendaten ist die Transformation von Informationen in numerische Vektorrepräsentationen unerlässlich [AU22]. Um das Verständnis der nachfolgenden Architektur-Entscheidungen zu erleichtern, werden zunächst die grundlegenden Konzepte der Tokenisierung und Repräsentation definiert:

- **Token**: Ein Token stellt die kleinste diskrete Einheit einer Eingabesequenz dar, in die Rohdaten zerlegt werden, bevor sie ein Modell verarbeitet. Während dies im Textbereich

Wörter oder Wortteile (Sub-words) sein können [DCLT19], werden in multimodalen Modellen auch Bild-Patches [DBK+21] als Tokens behandelt. Im Modell dienen Tokens als eindeutige Identifikatoren (Token IDs), die eine Verbindung zwischen Rohdaten und ihrer mathematischen Repräsentation (Vektoren) herstellen.

- **Repräsentation**: Die Repräsentation ist die numerische Kodierung (Vektorisierung) von Informationen, die es einem Modell erlaubt, die Semantik, Struktur oder Merkmale von Daten mathematisch zu erfassen [AU22].

- **Embedding**: Ein Embedding ist eine gelernte, dichte Vektorrepräsentation. Man unterscheidet hierbei zwischen statischen Einbettungen (z. B. Word2Vec [MCCD13]), die semantische Grundbeziehungen abbilden, und *kontextualisierten Repräsentationen* (z. B. Bidirectional Encoder Representations from Transformers (BERT) [DCLT19]), bei denen die mathematische Bedeutung eines Elements dynamisch durch seine Beziehung zu umgebenden Datenpunkten bestimmt wird.

Zur Veranschaulichung dient folgende Analogie: Wenn Daten eine Sprache wären, dann entsprechen die *Tokens* den einzelnen Buchstaben oder Wörtern in einem Wörterbuch, während die *Repräsentation* die tiefere Bedeutung ist, die diese Wörter erst in einem konkreten Satz ergeben.

Historisch betrachtet basierten frühe Ansätze auf frequenzbasierten Verfahren wie dem Bag-of-Words-Modell oder der Term Frequency-Inverse Document Frequency (TF-IDF) [SB88]. Diese Methoden repräsentieren Dokumente als Vektoren von Worthäufigkeiten, ignorieren jedoch weitgehend die grammatikalische Struktur und die semantische Bedeutung der Wörter. Zudem führen sie oft zu hochdimensionalen, dünn besetzten Vektoren (sparse vectors).

Einen signifikanten Fortschritt markierte die Einführung von verteilten Wortrepräsentationen (Word Embeddings), insbesondere durch das Word2Vec-Verfahren [MCCD13]. Hierbei werden Wörter in einen dichten, niedrigdimensionalen Vektorraum projiziert, wobei diese Projektionen durch einfache neuronale Netze gelernt werden. Semantisch ähnliche Wörter liegen in diesem Vektorraum nahe beieinander. Ein wesentlicher Nachteil dieser statischen Embeddings besteht jedoch darin, dass jedem Wort unabhängig von seinem Kontext ein fixer Vektor zugewiesen wird. Polyseme Wörter, die je nach Satzkontext unterschiedliche Bedeutungen haben, können so nicht adäquat abgebildet werden.

Um dieses Defizit zu beheben, wurden kontextuelle Embeddings entwickelt. Peters et al. stellten mit Embeddings from Language Models (ELMo) einen Ansatz vor, der tiefere neuronale Netze nutzt, um kontextabhängige Wortvektoren zu generieren [PNI+18]. Den aktuellen Standard setzen jedoch Transformer-basierte Modelle wie BERT, eingeführt von Devlin et al. [DCLT19]. Als Encoder-only-Modell wird BERT auf großen Textmengen vor-trainiert und erzeugt durch ein bidirektionales Training tiefgreifende kontextuelle Repräsentationen. Im Gegensatz zu ELMo

basiert BERT auf der Transformer-Architektur und nutzt intensiv Self-Attention-Mechanismen, deren technische Details in Abschnitt 2.2 erläutert werden.

Für die Verarbeitung ganzer Sätze oder Dokumente, wie sie in dieser Arbeit relevant ist, stoßen reine Token-Embeddings jedoch an Grenzen. Hier kommen Techniken wie Pooling oder die Verwendung spezieller Tokens ins Spiel. Insbesondere das `[CLS]`-Token (Classifier Token) in Modellen wie BERT wird häufig genutzt, um eine aggregierte Repräsentation des gesamten Satzes zu erhalten. Dieses spezielle Token wird jeder Eingabesequenz vorangestellt und sammelt während der Verarbeitung durch die Transformer-Schichten über Self-Attention kontextuelle Informationen des gesamten Inputs. Der finale Vektor des `[CLS]`-Tokens ($T_{[CLS]}$) dient somit als kompakte Repräsentation der gesamten Eingabe und wird in Klassifikationsaufgaben üblicherweise verwendet, um die Vorhersage zu treffen [DCLT19]. Die Nutzung des `[CLS]`-Tokens ermöglicht stabile semantische Kodierungen auf Satzebene, welche die notwendige Voraussetzung schaffen, um Textinformationen effizient mit anderen Modalitäten, wie tabellarischen Merkmalen, in einer multimodalen Architektur zu fusionieren.

2.1.3 Repräsentation tabellarischer Daten

Tabellarische Daten bilden die Grundlage zahlreicher Anwendungen in der Industrie, stellen jedoch für Deep-Learning-Ansätze eine besondere Herausforderung dar [BLS+24]. Um die in Abschnitt 2.1.1 beschriebene Lücke zu schließen, ist eine Transformation der rohen Tabellendaten in eine kompatible Vektor-Repräsentation notwendig.

Herausforderungen der Tabellenstruktur Tabellen zeichnen sich durch eine starke Heterogenität aus. Sie bestehen aus einem Mix von dichten numerischen Features (kontinuierliche Werte wie Preis oder Alter) und sparsen kategorialen Features (diskrete Werte wie Postleitzahl oder Produktkategorie) [BLS+24, BSP23].

Ein weiteres wesentliches Merkmal ist die Permutationsinvarianz der Spalten. Anders als Wörter in einem Satz haben die Spalten einer Tabelle keine natürliche Ordnung. Ein Modell muss daher robust gegenüber der Vertauschung von Feature-Positionen sein, solange die Zuordnung von Wert und Feature-Typ erhalten bleibt [GRKB21]. Die Repräsentation $(Feature_A, Feature_B)$ muss für das Netzwerk semantisch identisch zu $(Feature_B, Feature_A)$ verarbeitet werden.

Grenzen klassischer Vorverarbeitungsmethoden In traditionellen Machine-Learning-Verfahren, wie Gradient Boosted Decision Trees (GBDT), werden kategoriale Daten häufig mittels One-Hot-Encoding (OHE) verarbeitet. Dabei wird eine kategoriale Variable mit der Kardinalität C in einen binären Vektor der Länge C transformiert. Für Deep-Learning-Modelle und insbesondere für die

Integration in Transformer-Architekturen weist dieses Verfahren jedoch gravierende Nachteile auf:

Zum einen führt OHE bei Features mit hoher Kardinalität zu extrem hochdimensionalen und dünn besetzten Vektoren (Sparsity). Dies erschwert es neuronalen Netzen, dichte und aussagekräftige Repräsentationen zu lernen, da der Großteil der Eingabewerte Null ist [HKCK20, GRKB21]. Zum anderen fehlt OHE jegliche semantische Information. Alle Kategorien werden als orthogonal betrachtet, was bedeutet, dass der Abstand zwischen allen Kategorien im Vektorraum identisch ist. Semantische Beziehungen, wie etwa die Ähnlichkeit zwischen den Kategorien „Auto“ und „LKW“ im Vergleich zu „Apfel“, können durch OHE nicht abgebildet werden [HKCK20].

Auch einfache Multi-Layer Perceptrons (MLPs) stoßen auf rohen Daten an ihre Grenzen, da sie numerische Eingaben lediglich als skalare Werte betrachten und im ersten Schritt keine feature-spezifische Verarbeitung ermöglichen, wie sie beispielsweise für das Erlernen von Interaktionen zwischen spezifischen Features notwendig wäre [GRKB21].

Tabular Tokenization und Latenter Raum Der Prozess, tabellarische Daten in eine für Transformer verarbeitbare Sequenz zu überführen, wird allgemein als *Tabular Tokenization* bezeichnet. Ziel ist die Projektion der heterogenen Rohdaten in einen gemeinsamen latenten Raum [BLS+24, BSP23].

Das Ziel ist es, eine Tabellenzeile x in eine Sequenz von Vektoren $\mathbf{E}$ zu transformieren, die strukturell der Eingabe eines Sprachmodells gleicht:

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N] \quad \text{mit} \quad \mathbf{e}_i \in \mathbb{R}^H \tag{2.1}$$

Hierbei entspricht H der Dimension des Hidden-States des verwendeten Transformer-Modells (z. B. $H = 768$ bei BERT). Durch diese Angleichung der Dimensionen wird die technische Voraussetzung geschaffen, um später Mechanismen wie Cross-Attention zwischen Text- und Tabellen-Tokens anzuwenden [BSP23]. Im Gegensatz zu OHE erlaubt dieser dichte Vektorraum das Erlernen semantischer Ähnlichkeiten, sodass verwandte Features im Vektorraum geometrisch näher beieinander liegen.

Es ist jedoch essenziell zu unterscheiden: Während bei Text-Tokens die Reihenfolge die semantische Bedeutung bestimmt, handelt es sich bei *Tabular Tokens* um eine künstlich erzeugte Sequenz ohne natürliche Ordnung. Die Sequenzform dient hier rein als technisches Konstrukt für die Attention-Verarbeitung, nicht als Träger syntaktischer Informationen.

Architekturen der Repräsentation Hinsichtlich der Struktur der erzeugten Sequenz lassen sich zwei Hauptstrategien unterscheiden, die den Unterschied zwischen spaltenweiser und zeilenweiser Verarbeitung verdeutlichen [BSP23]:

Der erste Ansatz, bekannt als **Feature Tokenization** (oder Column-wise Embeddings), bildet jedes Feature der Ursprungstabelle isoliert auf genau einen Token-Vektor ab. Dies wird beispielsweise im *TabTransformer* [HKCK20] oder im *SAINT*-Modell [SGS+21] angewendet. Die Sequenzlänge N entspricht hierbei exakt der Anzahl der Spalten. Der Vorteil liegt in der Erhaltung der feingranularen Information jedes einzelnen Features.

Der zweite Ansatz, der in dieser Arbeit auch Anwendung findet, verfolgt eine **globale Projektion** (Row-level Tokenization). Hierbei werden alle Features einer Zeile zunächst konkateniert und gemeinsam durch ein MLP geleitet, um eine Sequenz von latenten Tokens zu erzeugen, die nicht mehr zwangsläufig 1-zu-1 den ursprünglichen Spalten entsprechen [GB21]. Dies ermöglicht eine Entkopplung der Sequenzlänge von der Anzahl der Eingabefeatures und kann die Rechenkomplexität in der nachfolgenden Attention-Schicht reduzieren. Entscheidend ist hierbei, dass das Modell bereits in der Tokenization-Phase globale Zusammenhänge zwischen den Features aggregieren kann, anstatt sie isoliert zu betrachten.

Unabhängig von der gewählten Methode ist das Ergebnis dieser Transformation eine Sequenz dichter Vektoren. Erst durch diesen Schritt wird die „Sprache“ der Tabelle in die „Sprache“ des Transformers übersetzt, was die Basis für die in dieser Arbeit untersuchte multimodale Fusion mittels Cross-Attention bildet.

2.2 Transformer-Architekturen und Attention-Mechanismen

Nachdem im vorangegangenen Kapitel die Grundlagen der Datenrepräsentation für Text und Tabellen erläutert wurden, widmet sich dieser Abschnitt der Modellarchitektur, die den aktuellen Stand der Technik in der Verarbeitung natürlicher Sprache definiert: dem Transformer. Ursprünglich für maschinelle Übersetzungsaufgaben konzipiert, hat sich diese Architektur als leistungsfähiger Standard für die Modellierung komplexer Abhängigkeiten in sequenziellen Daten etabliert. Das Verständnis der internen Mechanismen, insbesondere der Attention, ist eine notwendige Voraussetzung für die in dieser Arbeit entwickelte multimodale Fusion.

2.2.1 Der Transformer-Encoder

Das Transformer-Modell wurde 2017 von Vaswani et al. eingeführt und stellt einen signifikanten Fortschritt im Deep Learning dar, da es auf Rekurrenz und Faltung verzichtet und stattdessen primär auf Attention-Mechanismen basiert [VSP+17]. Die ursprüngliche Architektur besteht aus zwei Hauptkomponenten: einem Encoder, der die Eingabesequenz verarbeitet und in eine abstrakte Repräsentation überführt, und einem Decoder, der basierend darauf eine Ausgabesequenz generiert.

Für die Aufgabenstellung dieser Arbeit, bei der es um die Klassifikation von multimodalen Daten geht, ist primär der Encoder-Teil von Relevanz. Encoder-only-Modelle, wie das in Abschnitt 2.1.2 vorgestellte BERT, nutzen einen Stapel von Transformer-Blöcken, um für jedes Eingabe-Token einen kontextualisierten Vektor zu berechnen [DCLT19]. Im Gegensatz zu statischen Embeddings, bei denen ein Wort stets denselben Vektor erhält, integriert der Transformer-Encoder Informationen aus dem gesamten Kontext der Sequenz in die Repräsentation jedes einzelnen Tokens. Der Encoder fungiert somit als Feature-Extraktor, der in der Lage ist, syntaktische und semantische Beziehungen innerhalb der Daten abzubilden.

2.2.2 Scaled Dot-Product Attention

Das zentrale Element eines jeden Transformer-Blocks ist der Attention-Mechanismus. Er ermöglicht es dem Modell, die Relevanz verschiedener Teile der Eingabe für den aktuellen Verarbeitungsschritt dynamisch zu gewichten. Vaswani et al. formalisieren dies als *Scaled Dot-Product Attention* [VSP+17].

Das Konzept lässt sich abstrakt als ein Abfrage-Prozess beschreiben: Eine Abfrage (*Query* Q) wird mit einer Menge von Schlüsseln (*Keys* K) verglichen, um die korrespondierenden Werte (*Values* V) zu aggregieren. Mathematisch wird dies durch folgende Gleichung ausgedrückt:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2}$$

Hierbei berechnet das Skalarprodukt QK^T die Ähnlichkeit zwischen der Query und den Keys. Der Faktor $\frac{1}{\sqrt{d_k}}$ dient der Skalierung, um numerische Instabilitäten bei großen Dimensionen zu vermeiden [VSP+17]. Die Softmax-Funktion normalisiert diese Ähnlichkeitswerte zu Wahrscheinlichkeiten, die sich zu 1 aufsummieren. Schließlich wird eine gewichtete Summe der Values V gebildet. Dies bedeutet, dass der resultierende Vektor primär Informationen von denjenigen Elementen enthält, die eine hohe Ähnlichkeit zur Query aufweisen.

2.2.3 Self-Attention vs. Cross-Attention

Ein wesentlicher Aspekt moderner Transformer-Architekturen und insbesondere multimodaler Modelle ist die Definition der Quellen für die Vektoren Q, K und V. Hierbei wird zwischen *Self-Attention* und *Cross-Attention* unterschieden.

Self-Attention (Intra-Modal) Im Standard-Encoder, wie er von Vaswani et al. [VSP+17] beschrieben wird, stammen Query, Key und Value aus derselben Eingabequelle (z. B. der Ausgabe des vorherigen Layers). Dies wird als Self-Attention bezeichnet.

- **Funktionsweise:** Jedes Token der Sequenz berechnet seine Aufmerksamkeit in Bezug auf alle anderen Tokens derselben Sequenz.
- **Zweck:** Modellierung von Abhängigkeiten innerhalb einer Modalität. Dies erlaubt dem Modell, kontextuelle Beziehungen, wie etwa Referenzen innerhalb eines Satzes, zu erfassen.

Cross-Attention (Inter-Modal) Für die Fusion unterschiedlicher Modalitäten, wie Text und Tabellendaten, ist die Cross-Attention von zentraler Bedeutung. Dieses Konzept findet sich in diversen multimodalen Architekturen [TB19].

- **Funktionsweise:** Die Vektoren werden aus unterschiedlichen Quellen gespeist. Beispielsweise generiert Modalität A (z. B. Text) die Queries Q, während Modalität B (z. B. Tabelle) die Keys K und Values V bereitstellt.
- **Zweck:** Ermöglichung eines Informationsflusses über Modalitätsgrenzen hinweg. Die Text-Repräsentationen können gezielt Informationen aus den Tabellen-Features integrieren, die für den aktuellen Kontext relevant sind. Dies bildet das theoretische Fundament für flexible Fusions-Architekturen, da es eine dynamische Verknüpfung heterogener Datenräume erlaubt.

2.3 Strategien der multimodalen Fusion

Voraussetzung für jede Fusion ist, dass die unterschiedlichen Modalitäten bereits in einen kompatiblen Vektorraum transformiert wurden (vgl. Abschnitt 2.1). Die Strategien der multimodalen Fusion befassen sich anschließend mit der Frage, *wie* (Mechanismen) und *wann* (Architekturen) diese Informationen technisch kombiniert werden [LT24, BAM19].

2.3.1 Mechanismen der Fusion

Die technische Realisierung der Fusion, also *wie* die Vektoren der unterschiedlichen Modalitäten zusammengeführt werden, kann durch verschiedene mathematische Operationen erfolgen [JGF+24, BAM19]. Ein zentrales Konzept ist hierbei die *Joint Representation*. Das Ziel ist es, die unimodalen Eingangsvektoren in einen gemeinsamen semantischen Unterraum zu projizieren, um eine einzige, multimodale Repräsentation Z zu erzeugen [JGF+24, LT24]. Mathematisch lässt sich dieser Vorgang allgemein formulieren als:

$$Z = f(X_{\text{text}}, X_{\text{tab}}) \tag{2.3}$$

Wobei f eine Funktion darstellt (beispielsweise ein neuronales Netz oder eine deterministische Operation), die aus den unimodalen Repräsentationen X_{text} und X_{tab} die fusionierte multimodale Repräsentation Z berechnet [BAM19].

Zu den gängigsten spezifischen Operationen zählen:

Concatenation (Verkettung) Dies ist die wohl einfachste Form der Fusion, bei der die Merkmalsvektoren lediglich hintereinander gehängt werden:

$$Z = [X_{\text{text}} \parallel X_{\text{tab}}] \tag{2.4}$$

Ein wesentlicher Vorteil ist, dass die Eingangsvektoren X_{text} und X_{tab} nicht dieselbe Dimension besitzen müssen. Die Dimension des Ergebnisvektors Z entspricht der Summe der Einzeldimensionen. In der Literatur wird dieser Ansatz oft als „Early Fusion“ bezeichnet, wenn er direkt auf der Ebene der Eingangsmerkmale angewendet wird [BAM19].

Addition (Summierung) Hierbei werden die Vektoren elementweise addiert:

$$Z = X_{\text{text}} + X_{\text{tab}} \tag{2.5}$$

Dies setzt zwingend voraus, dass beide Vektoren dieselbe Dimension aufweisen, weshalb oft eine vorherige Projektion notwendig ist. Diese Methode ist besonders in Joint-Representation-Ansätzen verbreitet [JGF+24].

Attention-Mechanismen Im Gegensatz zu statischen Operationen wie der Addition oder Verkettung erlauben Attention-Mechanismen eine dynamische Gewichtung der Features. Besonders relevant ist die *Cross-Attention*. Hierbei können Repräsentationen einer Zielmodalität (z. B. Text) gezielt Informationen aus einer Quellmodalität (z. B. Tabelle) abfragen. Typischerweise stammen dabei die *Queries* aus der Zielmodalität, während *Keys* und *Values* aus der Quellmodalität geliefert werden. Dies ermöglicht eine gewichtete Aggregation von Informationen, die auf den aktuellen Kontext konditioniert ist, und erlaubt die Modellierung komplexer Abhängigkeiten über Modalitätsgrenzen hinweg [LT24].

2.3.2 Architekturen und Positionierung der Fusion

Neben der konkreten Fusionsoperation ist entscheidend, *an welcher Stelle* im Modellfluss die Modalitäten zusammengeführt werden. Die Position bestimmt, ob Interaktionen bereits auf niedriger Ebene gelernt werden (früh), ob zunächst getrennte Repräsentationen entstehen (spät) oder ob die Integration mehrfach über die Modelltiefe erfolgt [JGF+24, LT24, BAM19].

Early Fusion (Daten- und Feature-Ebene) Bei der Early Fusion werden die Modalitäten auf Rohdaten- oder Feature-Ebene zu einer gemeinsamen Eingabe kombiniert und anschließend in einem gemeinsamen Modellzweig (*One-Tower*) weiterverarbeitet [JGF+24, LT24, BAM19]. Dadurch können Interaktionen zwischen Merkmalen früh und kontextübergreifend gelernt werden, jedoch steigen die Anforderungen an Ausrichtung und Kompatibilität der Modalitäten; bei stark heterogenen Eingaben kann dies die Modellierung erschweren [BAM19, JGF+24].

Late Fusion (Entscheidungs- und Ausgabe-Ebene) Bei der Late Fusion werden Text und Tabelle zunächst in getrennten Zweigen (*Two-Tower*) verarbeitet und erst am Ende über ihre High-Level-Repräsentationen oder Prädiktionen kombiniert [LT24]. Typische Operatoren sind Mittelwertbildung, Voting oder ein nachgeschaltetes MLP [BAM19]. Der Ansatz ist modular und robust gegenüber fehlenden Modalitäten, modelliert jedoch kaum niedrigstufige Interaktionen; zudem wirken sich starke Störungen einer Modalität direkt auf die Endentscheidung aus [BAM19].

Hybrid und Deep Fusion Hybride Ansätze kombinieren frühe und späte Integrationspunkte, etwa indem getrennte Zweige durch zusätzliche Fusionsmodule gekoppelt werden [JGF+24, BAM19, LT24]. *Deep Fusion* erweitert dieses Prinzip, indem Fusion wiederholt in Zwischenschichten stattfindet, wodurch Interaktionen schichtweise verfeinert werden können [JGF+24]. Transformer-basierte Modelle realisieren dies häufig über Cross-Attention, die Modalitäten iterativ über mehrere Layer hinweg verbindet [LT24].

3 Verwandte Arbeiten

Das Feld des multimodalen Lernens hat durch die Adaption der Transformer-Architektur signifikante Fortschritte erzielt. Während die Kombination von Bild und Text (Vision-Language Models) bereits etablierte Standards für die Fusion mittels Cross-Attention hervorgebracht hat, ist die gemeinsame Verarbeitung von tabellarischen Daten und Text ein jüngeres und weniger standardisiertes Forschungsfeld. Dieses Kapitel ordnet die Arbeit in den aktuellen Forschungsstand ein, differenziert zwischen generellen Fusions-Architekturen und spezifischen Ansätzen für Tabellen und stellt den im Unternehmenskontext relevanten Benchmark vor.

3.1 Cross-Attention in multimodalen Transformer-Architekturen

Die effektivste Methode zur Fusion heterogener Datenströme in modernen neuronalen Netzen ist der Einsatz von Cross-Attention-Mechanismen. Diese ermöglichen es einem Modell, Informationen einer Modalität dynamisch in den Kontext einer anderen zu integrieren, ohne die ursprüngliche Struktur der Daten frühzeitig aufzulösen.

Im Bereich Vision-Language zeigt **LXMERT** [TB19], wie effektiv eine Dual-Stream-Architektur sein kann: Bild- und Textdaten werden zunächst von separaten Encodern verarbeitet, bevor sie in einem dedizierten Cross-Modality-Encoder fusioniert werden. Dies verhindert den Informationsverlust, der bei einer zu frühen einfachen Konkatenation (Early Fusion) auftreten kann. Einen Schritt weiter geht **Flamingo** [ADL$^+$22], ein Modell für Few-Shot-Learning. Anstatt ein neues Modell von Grund auf zu trainieren, injiziert Flamingo visuelle Informationen über *Gated Cross-Attention-Dense*-Schichten in ein eingefrorenes Sprachmodell. Dies belegt die Flexibilität von Cross-Attention als Mechanismus zur späten, aber tiefen Integration von Zusatzinformationen. Auch neuere Modelle wie **PaLI** [CDP$^+$23] bestätigen, dass diese dynamische Interaktion einfacheren (statischen) Fusionsmethoden überlegen ist, wenn komplexe semantische Korrelationen erfasst werden müssen.

3.2 Verarbeitung von Tabellen und Text mit Transformern

Die Übertragung dieser Konzepte auf tabellarische Daten stellt aufgrund der in Abschnitt 2.1.1 beschriebenen fehlenden Sequenzialität eine Herausforderung dar.

Der **TabTransformer** [HKCK20] lieferte den fundamentalen Nachweis, dass Transformer-Encoder effektiv auf kategoriale Daten anwendbar sind, indem Features als individuelle Tokens betrachtet werden (Feature Tokenization). Obwohl er Text ignoriert, etablierte er die Basis für spaltenweise Attention. Für die Verbindung mit Text wählt **TaBERT** [YNYR20] einen Ansatz der Linearisierung: Tabellenzeilen werden in text-ähnliche Sequenzen umgewandelt und gemeinsam

mit natürlicher Sprache durch ein BERT-Modell verarbeitet. Dies entspricht einer Early Fusion, bei der die modalitätsspezifische Struktur teilweise verschwimmt.

Neuere Untersuchungen im **Multimodal-Toolkit** [GB21] und von Bonnier et al. [Bon24] vergleichen verschiedene Architekturen systematisch. Bonnier et al. stellen fest, dass spezialisierte multimodale Architekturen in der Praxis oft nur marginale Verbesserungen gegenüber starken Baselines (wie der bloßen Konkatenation von Text- und Tabellen-Features) erzielen, wenn die unimodalen Repräsentationen nicht sorgfältig abgestimmt sind. Dies unterstreicht die Notwendigkeit, Fusionsmechanismen nicht nur theoretisch, sondern auch empirisch gegen robuste Standards zu validieren.

Industrieller Benchmark: CrossFitter Stacking Im Unternehmenskontext dieser Arbeit dient die interne Modellarchitektur „CrossFitter“ als relevante Baseline. Im Gegensatz zu den oben genannten End-to-End-Architekturen handelt es sich hierbei um einen zweistufigen Stacking-Ansatz, der die Stärken spezialisierter Modelle kombiniert, ohne dass diese in einem gemeinsamen, durchgängig trainierbaren Modell vereint sind.

Das Verfahren nutzt ein **2-Fold-Cross-Prediction-Schema**, um Textinformationen für ein tabellarisches Modell nutzbar zu machen:

1. Die Trainingsdaten werden in zwei Folds (A und B) unterteilt.
2. Ein BERT-Modell wird auf Fold A trainiert und generiert Vorhersagen für Fold B (Out-of-Sample Predictions).
3. Analog wird ein zweites BERT-Modell auf Fold B trainiert, um Vorhersagen für Fold A zu erstellen.

Die resultierenden Wahrscheinlichkeitswerte (Scores) des Textmodells werden anschließend als zusätzliches Feature gemeinsam mit den ursprünglichen strukturierten Daten in ein Gradient-Boosting-Modell (LGBM) eingespeist. Dieser Ansatz ist in der Praxis äußerst robust und performant, besitzt jedoch eine theoretische Limitierung: Es findet keine direkte Interaktion der Features während des Trainings statt. Das Textmodell erhält kein Feedback von den Tabellendaten und kann seine Repräsentation nicht an den Kontext der strukturierten Attribute anpassen.

3.3 Forschungslücke und Einordnung

Die Analyse der verwandten Arbeiten zeigt eine deutliche Zweiteilung der aktuellen Forschungslandschaft:

- Auf der einen Seite existieren hochentwickelte **Cross-Attention-Architekturen** für Bild und Text (Flamingo), die eine tiefe Interaktion der Modalitäten ermöglichen.
- Auf der anderen Seite dominieren im Bereich Tabelle und Text oft noch Ansätze der Linearisierung (TaBERT) oder das in der Industrie bewährte **Stacking** (CrossFitter).

Es besteht eine Forschungslücke in der Übertragung der erfolgreichen Cross-Attention-Konzepte aus der Vision-Language-Domäne auf das Tabular-Text-Problem. Während TabTransformer zeigt, wie man Tabellen „tokenisiert“, fehlt oft der Schritt, diese Tabellen-Tokens dynamisch mit Text-Tokens interagieren zu lassen. Die vorliegende Arbeit untersucht daher eine Architektur der *Hybrid Fusion* mittels Cross-Attention. Ziel ist es zu prüfen, ob die direkte, differenzierbare Interaktion der Modalitäten einen messbaren Mehrwert gegenüber dem robusten, aber statischen Stacking-Ansatz des CrossFitters bietet.

4 Methodik

Dieses Kapitel beschreibt Datenbasis, Vorverarbeitung und experimentelles Setup zur Entwicklung und Evaluation der in dieser Arbeit vorgestellten CrossBERT-Architektur.

4.1 Datenbasis und Vorverarbeitung

Es wird ein hybrider Ansatz verfolgt: proprietäre Echtdaten aus der Versicherungswirtschaft werden mit öffentlichen Benchmarks kombiniert. Alle Datensätze werden über eine einheitliche Pipeline (`LightningDataModule`) in konsistente Tensor-Repräsentationen überführt, inklusive Splitting, tabellarischer Transformationen und Texttokenisierung.

4.1.1 Datenbasis

Der primäre Datensatz stammt aus einem deutschen Versicherungsunternehmen und wurde für die Identifikation von Regresspotenzialen kuratiert. Es handelt sich um ein Multi-Class-Setting mit den Zielklassen *Kein Regress* (0), *Regress* (1) und *Versicherungsschutzentzug* (2). Er kombiniert unstrukturierte Schadenbeschreibungen mit tabellarischen Metadaten (Tabelle 1 zeigt einen exemplarischen Datenpunkt). Methodisch relevant sind ein *Selection Bias* (Labels entstehen historisch nur für manuell geprüfte Fälle) sowie ein *Missing Label Problem*. Für ungelabelte Fälle wird initial Label 0 angenommen und die dadurch entstehenden Unsicherheiten werden als potenziell verrauschte Labels berücksichtigt.

Tab. 1: Exemplarischer Datensatz der internen Regressdatenbasis (anonymisiert).

Metadaten	ID: 68443XXXX, Label: 1 (Regress), Status: Validiert (True)
Textmodalität	*Schadenhergang:* VN: Kreuzung: Vn hatte grünen Pfeil der aufgeleuchtet sei. Es kam zur Kollision Vorgang strittig. *Dokumente:* ausführliche HSA, strittige Ampel, PUM, EA-Geb-re io. *Notizen:* PWS empfiehlt Beauftragung eines SV; ADAC bereits auf dem Weg; wst wartet auf fg...
Tabellarische Merkmale	`log_zlg_fzg`: 9,839, `kh_other_partially_paid`: 1, `untersparte`: KF.VK, `risikoschluessel`: UNF, `svorg_schadenursache`: Unfall ein weiteres Kfz, `had_ever_erinnerungsregress`: 1

Zur externen Validierung werden zwei Kaggle-Benchmarks genutzt:

1. **Women's E-Commerce Clothing Reviews** [Bro18]: Binäre Sentiment-Klassifikation mit Text und Metadaten.
2. **PetFinder.my Adoption Prediction** [Pet19]: Multi-Class-Klassifikation mit Beschreibungen und Metadaten.

Damit wird die Übertragbarkeit der Architektur außerhalb der Versicherungsdomäne überprüft.

4.1.2 Datenvorverarbeitung

Die Vorverarbeitung trennt strikt Trainings-, Validierungs- und Testdaten, um Data Leakage zu vermeiden. Das Splitting erfolgt stratifiziert (bei Multi-Class) und nutzt (sofern verfügbar) feste Test-Indizes. Zusätzlich wird aus dem Trainingssplit ein Validierungsanteil (`validation_fraction`) abgeleitet. Optional wird die Klassenverteilung im Training über Undersampling der Majoritätsklasse auf ein vorgegebenes Verhältnis $N(0)/N(1)$ (`train_imbalance_0_to_1`) eingestellt. Die zentralen Schritte sind:

- **Tabellarische Daten:** One-Hot-Encoding für kategoriale Merkmale (fehlende Werte als Kategorie `None`, unbekannte Kategorien werden ignoriert); Min-Max-Skalierung numerischer Variablen auf $[0, 1]$. Die Encoder/Scaler werden ausschließlich auf Trainingsdaten trainiert (*fit*) und auf Validierung/Test angewandt (*transform*).
- **Textdaten:** Tokenisierung mit `bert-base-uncased` (inkl. `[CLS]` und `[SEP]`); Padding/Truncation auf die modellabhängige maximale Kontextlänge (für BERT: 512 Tokens).
- **Splitting und Balancing:** Stratified Split unter Beibehaltung der Klassenverteilung (insbesondere relevant bei Multi-Class, z. B. PetFinder). Im Training wird optional ein Random Undersampling der Majoritätsklasse eingesetzt, gesteuert über `train_imbalance_0_to_1`.

4.2 Baseline-Modelle

Um den Mehrwert der entwickelten CrossBERT-Architektur präzise zu quantifizieren, erfolgt ein Vergleich mit zwei bestehenden multimodalen Modellen für die Fusion von tabellarischen Daten und Textdaten: SumBERT (Early Fusion) und ConcatBERT (Late Fusion). Der vollständige Quellcode beider Implementierungen befindet sich in Anhang A1.1.

4.2.1 SumBERT (Early Fusion)

SumBERT realisiert eine frühe Fusion, bei der die Tabelleninformationen bereits vor dem Encoder in den Modellkontext integriert werden. Hierfür werden die normalisierten und kodierten Tabellendaten zunächst mittels einer linearen Projektionsschicht (`nn.Linear`) in den Vektorraum des Sprachmodells projiziert ($H = 768$). Dieser projizierte Vektor wird anschließend elementweise zum Embedding des `[CLS]`-Tokens addiert, noch bevor die Sequenz den ersten Transformer-Layer erreicht (siehe Abbildung 1).

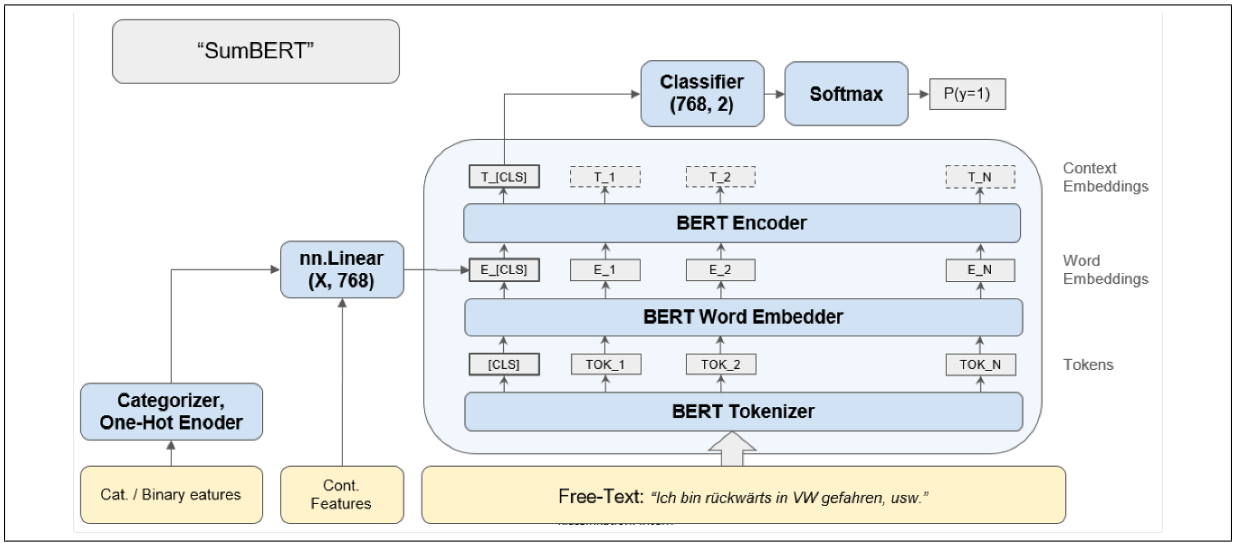


Abb. 1: Architektur von SumBERT: Addition der Tabellen-Repräsentation zum CLS-Token-Embedding vor dem Encoder.

Formal lässt sich die Berechnung des fusionierten Eingabevektors wie folgt beschreiben:

$$E_{\text{fused}}^{[CLS]} = E_{\text{text}}^{[CLS]} + \text{MLP}_{\text{tab}}(X_{\text{tab}}) \tag{4.1}$$

Durch diese additive Verknüpfung am Eingang fungiert das `[CLS]`-Token als globaler Träger multimodaler Informationen, die anschließend über den Self-Attention-Mechanismus im Transformer-Encoder in alle tieferen Schichten des Modells propagiert werden.

4.2.2 ConcatBERT (Late Fusion)

Im Gegensatz dazu verfolgt ConcatBERT einen Ansatz der späten Fusion. Text- und Tabellendaten werden hierbei zunächst separat verarbeitet. Der Text durchläuft den vollständigen BERT-Encoder, um eine kontextualisierte Repräsentation des `[CLS]`-Tokens zu generieren. Erst im Anschluss an den Encoding-Prozess wird dieser Vektor mit den rohen bzw. leicht vorverarbeiteten Tabellen-Features konkateniert (siehe Abbildung 2).

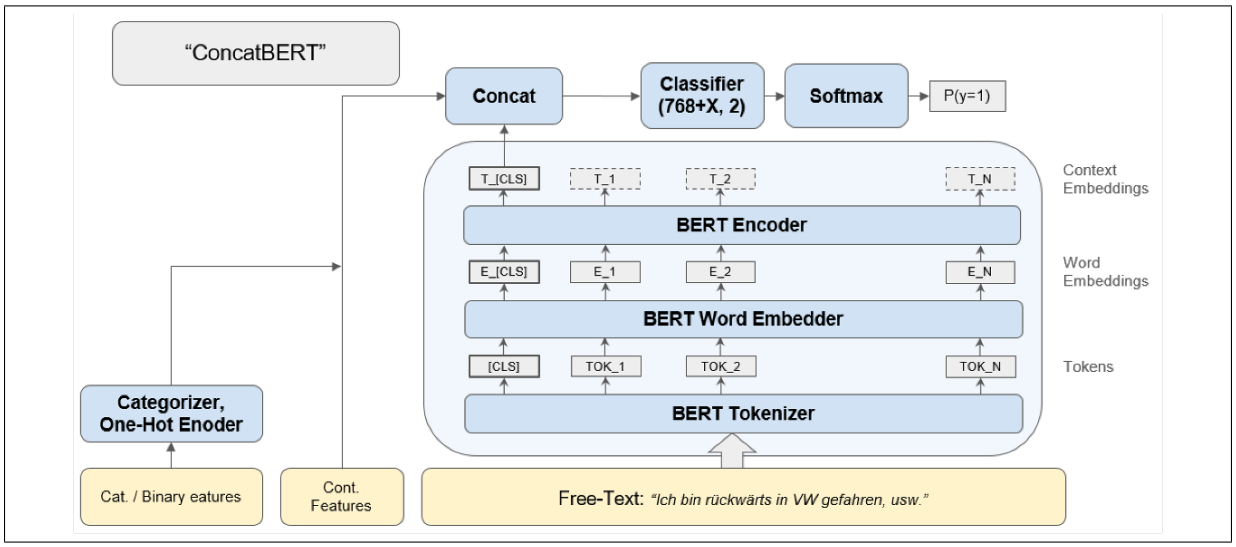


Abb. 2: Architektur von ConcatBERT: Konkatenation der Tabellen-Features mit dem Output des BERT-Encoders.

Der resultierende Vektor dient als Eingabe für den Klassifikations-Head:

$$h_{\text{fusion}} = [\text{BERT}(E_{\text{text}})_{[CLS]} \parallel X_{\text{tab}}] \tag{4.2}$$

Diese Architektur ist rechnerisch effizient und leicht zu implementieren, verhindert jedoch eine tiefe Interaktion zwischen den Modalitäten während der Feature-Extraktion, da die Tabellendaten den Self-Attention-Mechanismus des Transformers nicht durchlaufen.

4.3 CrossBERT-Architektur

Die CrossBERT-Architektur erweitert einen BERT-Klassifikator um Cross-Attention zwischen Text und Tabellendaten, um eine kontextabhängige, tiefe Interaktion der Modalitäten zu ermöglichen (vgl. Abschnitt 2.2). Im Unterschied zu SumBERT und ConcatBERT (Abschnitt 4.2) wird die tabellarische Modalität nicht als statischer Zusatzvektor behandelt, sondern als kurze Token-Sequenz in denselben Hidden-Space projiziert und vom Text gezielt „abgefragt“. Der vollständige Quellcode ist in Anhang A1.2 (Listing 2) dokumentiert.

4.3.1 Architekturüberblick

Abbildung 3 zeigt den Gesamtaufbau. Text wird mit `deepset/gbert-base` in Embeddings überführt und durch den Encoder kontextualisiert ($H = 768$). Tabellarische Features werden parallel durch einen *TabularTokenizer* in N Tab Tokens kodiert. Optional werden ein oder zwei Cross-Attention-Blöcke eingesetzt, die ausschließlich den `[CLS]`-Token mit Tab Tokens fusionieren: (i) *early* vor dem Encoder (Eingangs-Embeddings) und/oder (ii) *late* nach dem Encoder (finaler `[CLS]`-State). Der Klassifikationskopf arbeitet anschließend auf dem fusionierten `[CLS]`-Vektor.

4.3.2 TabularTokenizer

Die tabellarische Eingabe liegt nach der Vorverarbeitung als Vektor $X_{\text{tab}} \in \mathbb{R}^{B\times F}$ vor (normalisierte numerische Variablen und One-Hot-kodierte kategoriale Merkmale). Analog zur in Abschnitt 2.1.3 beschriebenen globalen Projektion werden daraus N latente Tokens $E_{\text{tab}} \in \mathbb{R}^{B\times N\times H}$ erzeugt (Abbildung 4a):

$$E_{\text{tab}} = [\mathbf{e}_1, \ldots, \mathbf{e}_N], \quad \mathbf{e}_i \in \mathbb{R}^{B\times H} \tag{4.3}$$

Praktisch wird dies durch N gelernte parallele MLP-Projektionsköpfe realisiert, die denselben Feature-Vektor in unterschiedliche Teilrepräsentationen abbilden. Intuitiv kodiert jeder Tab Token einen anderen Blick auf dieselben Tabulardaten und macht so eine gezielte Interaktion mit dem Text über Cross-Attention möglich. Die Tokenanzahl N ist ein Hyperparameter (`num_tab_token`); die Wahl eines kleinen N entkoppelt die Sequenzlänge von F und hält die Fusion rechnerisch effizient.

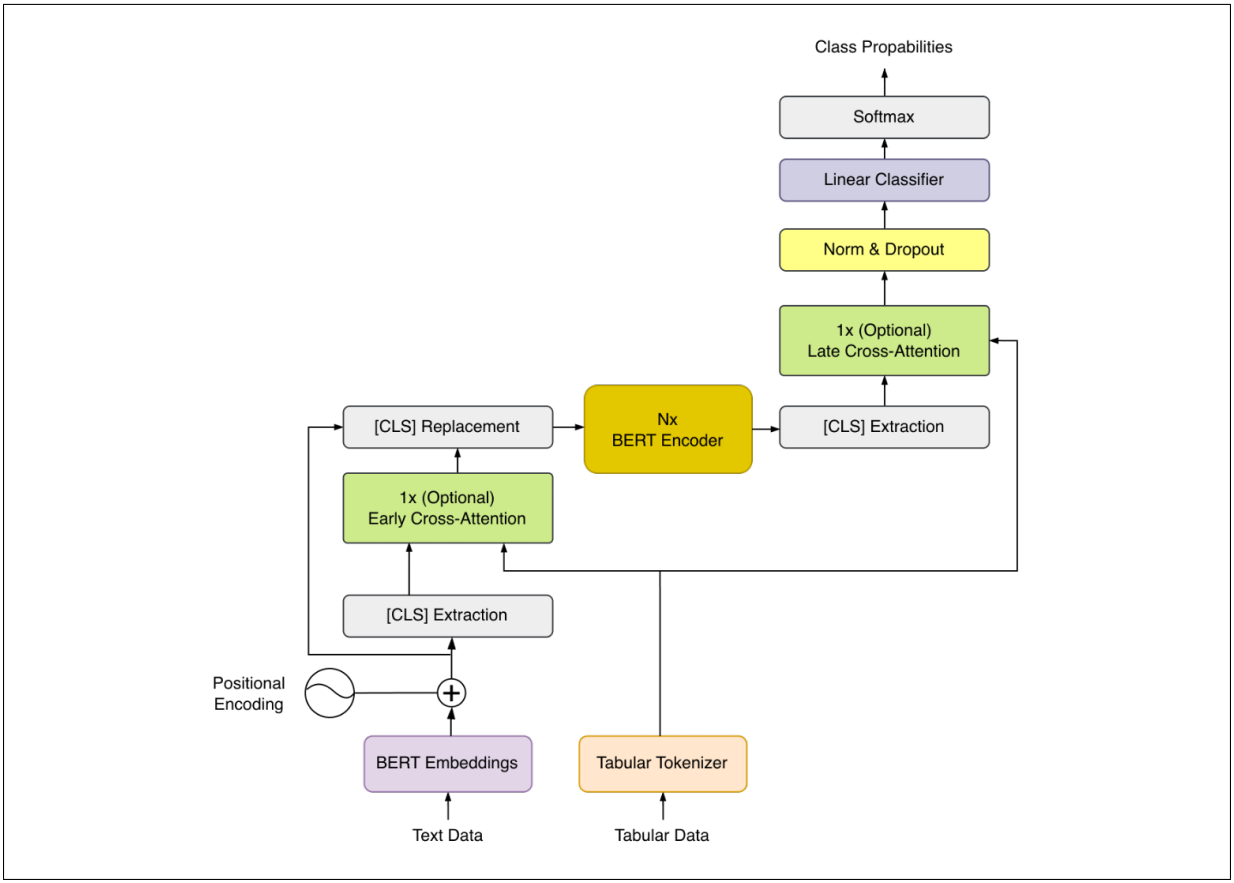


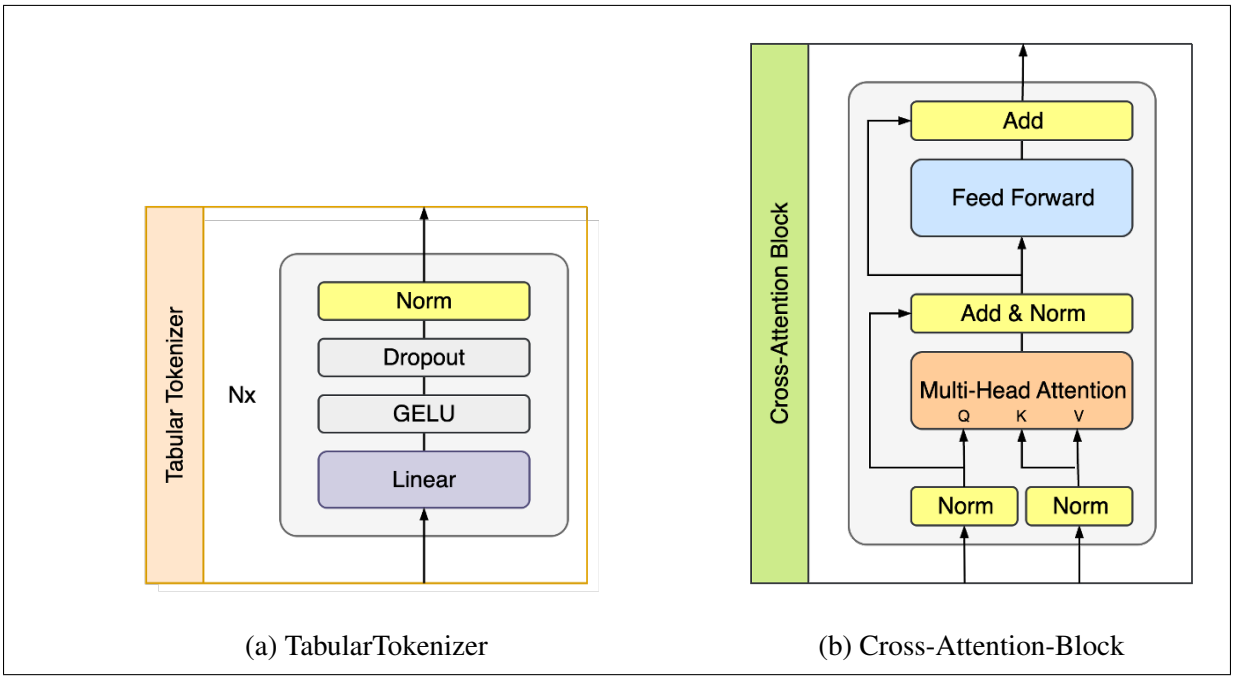
Abb. 3: Architektur von CrossBERT mit optionaler Early- und Late-Cross-Attention über Tab Tokens.

4.3.3 Cross-Attention-Block

Die Fusion erfolgt über Cross-Attention, bei der der `[CLS]`-Token als Query dient und Tab Tokens als Keys/Values (vgl. Abschnitt 2.2). Für $\mathbf{c} \in \mathbb{R}^{B\times 1\times H}$ gilt:

$$Q = \mathbf{c}W_Q, \quad K = E_{\text{tab}}W_K, \quad V = E_{\text{tab}}W_V \tag{4.4}$$

Die resultierenden Attention-Gewichte steuern, welche Teile der Tabulardaten für die aktuelle Texteingabe relevant sind. Dadurch wird die `[CLS]`-Repräsentation nach der Attention um tabellarischen Kontext angereichert. CrossBERT verwendet Multi-Head Attention und ergänzt diese um Residualpfade sowie ein Feed-Forward-Netzwerk (Abbildung 4b). Optional kann eine Tab-Maske genutzt werden, um einzelne Tab Tokens in der Attention auszublenden.



(a) TabularTokenizer (b) Cross-Attention-Block

Abb. 4: Komponenten der CrossBERT-Architektur: Links die Projektion der Features in Tab Tokens, rechts deren Integration via Cross-Attention.

4.3.4 Fusionsvarianten: Early, Late und Hybrid

Die Positionierung des Cross-Attention-Blocks operationalisiert die in den theoretischen Grundlagen diskutierten Fusionszeitpunkte:

- **Early Fusion:** Cross-Attention wird auf den `[CLS]`-Embedding-Vektor vor dem Encoder angewandt, sodass tabellarische Information die gesamte nachfolgende Kontextualisierung beeinflussen kann.
- **Late Fusion:** Cross-Attention wird erst auf den finalen Encoder-`[CLS]`-State angewandt, wodurch tabellarische Information als kontextabhängige Nachschärfung der Dokumentrepräsentation integriert wird.
- **Hybrid Fusion:** Early und Late Fusion werden kombiniert, um sowohl die Encoder-Features als auch die finale Entscheidungsschicht mit Tabellensignalen zu versorgen.

Falls keine tabellarischen Features vorliegen oder Cross-Attention deaktiviert ist, verhält sich CrossBERT identisch zu einem unimodalen BERT-Klassifikator und bleibt damit direkt vergleichbar.

4.4 Trainings-Setup und Evaluationsmetriken

Für eine wissenschaftliche Vergleichbarkeit werden alle Experimente unter identischen Rahmenbedingungen durchgeführt. Die Pipeline ist über eine zentralisierte Konfiguration parametrisiert und setzt alle relevanten Hyperparameter (Modell, Optimierung, Splitting, Callbacks) konsistent über Baselines und CrossBERT hinweg.

Modell und Lernstrategie. Als Textencoder wird `deepset/gbert-base` verwendet [CSM20]. GBERT ist ein vortrainiertes deutsches Sprachmodell auf Basis von BERT [DCLT19] und dient als Initialisierung für das end-to-end Fine-Tuning im jeweiligen Klassifikations-Setup. Die Klassifikation erfolgt überwacht mit Cross-Entropy-Loss. Die Architekturvariante (Baseline vs. CrossBERT) beeinflusst dabei ausschließlich die Art der multimodalen Fusion (vgl. Abschnitt 4.3).

Optimierung und Trainingsprotokoll. Die zentralen Trainingshyperparameter sind in Tabelle 2 zusammengefasst. Zur Stabilisierung des Fine-Tunings wird eine Lernratenstrategie aus linearem Warmup und anschließendem Cosine Scheduling eingesetzt. Das Early Stopping überwacht `val/average_precision`. Die Validierung wird in festen Intervallen innerhalb einer Epoche ausgeführt (`val_check_interval`=500). Für die *Gating*-Ablation nach Flamingo ([ADL+22], siehe Abschnitt 4.5) werden ausschließlich dafür zusätzlich (i) der BERT-Encoder für die ersten zwei Epochen eingefroren (`freeze_bert_epochs`=2), (ii) die Gating-Parameter (`alpha_`) mit einer um den Faktor 5 erhöhten Lernrate ohne Weight Decay optimiert (`lr_alpha_multiplier`=5,0) und (iii) die Epochen-Grenzen auf min/max 2/6 gesetzt.

Tab. 2: Zentrale Trainingshyperparameter (Basiskonfiguration ohne Gating).

Parameter	**Wert**
Seed	777
Optimizer	AdamW
Lernrate	$1 \cdot 10^{-5}$
LR-Scheduler	Linear Warmup (0,1) + Cosine
Batchgröße (Train / Eval)	64 / 128
Epochen (min / max)	1 / 4
Early Stopping	Patience 5, Monitor `val/average_precision`
Validierungsanteil	`validation_fraction`=0,2

Validierung, Hardware und Mixed Precision. Die Validierung erfolgt über einen festen Train/Val/Test-Split gemäß der Datenpipeline (vgl. Abschnitt 4.1). Alle Experimente werden auf einer einzelnen NVIDIA Tesla V100-PCIE-32GB ausgeführt. Zur Beschleunigung wird

Mixed Precision eingesetzt, wodurch sich Rechenzeit und Speicherbedarf bei vergleichbarer Modellqualität reduzieren lassen.

Evaluationsmetriken. Als Kernmetriken werden Average Precision (AP) und AUC-ROC berichtet. AP ist für unausgeglichene Klassenverteilungen besonders aussagekräftig, da sie die Precision-Recall-Kurve über alle Entscheidungsschwellen zusammenfasst. AUC-ROC misst die Fähigkeit des Modells, positive gegenüber negativen Beispielen korrekt zu ranken, unabhängig von einem festen Schwellenwert. In Multi-Class-Settings werden beide Kennzahlen one-vs-rest pro Klasse berechnet. Zusätzlich werden die Klassen binarisiert (positiv vs. Klasse 0), um eine konsistente Vergleichbarkeit über Datensätze hinweg zu gewährleisten.

4.5 Ablationsstudien

Um den Einfluss einzelner Architektur-Entscheidungen systematisch zu verstehen, werden gezielte Ablationsstudien durchgeführt.

Tab. 3: Übersicht der geplanten Ablationsstudien

Ablation	**Variable (Wertebereich)**	**Ziel**
Cross-Attention	Position: Early / Late / Hybrid	Einfluss des Fusionsorts (vor/nach Encoder vs. kombiniert)
Tab Tokens	n_{tab} (Hybrid): 0 / 1 / 2 / 4 / 8 / 16	Quantitativ prüfen, wie stark n_{tab} die Leistung von CrossBERT (Hybrid) beeinflusst
Gating Layer	Flamingo-style [ADL$^+$22]: Mit / Ohne	Adaptive Gewichtung Text↔Tab für stabilere Scores und ökonomische Kennzahlen
Tokenizer	TabTokenizer: V1 / V2 (per-Feature-Tokenisierung)	Prüfen, ob feinere Tokenisierung Fusion und Interpretierbarkeit verbessert

Im Fokus stehen (i) die Positionierung der Cross-Attention (Early/Late/Hybrid), (ii) die Sensitivität gegenüber der Anzahl der Tab Tokens n_{tab} im Hybrid-Setup, (iii) ein Gating-Mechanismus nach Flamingo zur stabileren Fusion sowie (iv) ein *TabTokenizer V2* mit per-Feature-Tokenisierung zur potenziell besseren Interpretierbarkeit. Für die *Gating*-Ablation wurden die im Flamingo-Ansatz beschriebenen trainingsseitigen Anpassungen gezielt übernommen (Encoder-Freezing und erhöhte Lernrate für die Gate-Parameter; vgl. Abschnitt 4.4), da andernfalls die zusätzlichen Freiheitsgrade des Gates im Fine-Tuning empirisch instabil waren.

5 Ergebnisse

6 Diskussion und Fazit

Literaturverzeichnis

[ADL+22] ALAYRAC, Jean-Baptiste ; DONAHUE, Jeff ; LUC, Pauline ; MIECH, Antoine ; BARR, Iain ; HASSON, Yana ; LENC, Karel ; MENSCH, Arthur ; MILLICAN, Katie ; REYNOLDS, Malcolm ; RING, Roman ; RUTHERFORD, Eliza ; CABI, Serkan ; HAN, Tengda ; GONG, Zhitao ; SAMANGOOEI, Sina ; MONTEIRO, Marianne ; MENICK, Jacob ; BORGEAUD, Sebastian ; BROCK, Andrew ; NEMATZADEH, Aida ; SHARIFZADEH, Sahand ; BINKOWSKI, Mikolaj ; BARREIRA, Ricardo ; VINYALS, Oriol ; ZISSERMAN, Andrew ; SIMONYAN, Karen: Flamingo: a Visual Language Model for Few-Shot Learning. In: *Advances in Neural Information Processing Systems* Bd. 35, 2022, S. 23716–23736

[AU22] ABUBAKAR, H. D. ; UMAR, M.: Sentiment Classification: Review of Text Vectorization Methods: Bag of Words, Tf-Idf, Word2vec and Doc2vec. In: *SLUJST* 4 (2022), Aug, Nr. 1 & 2, S. 27–33. *http://dx.doi.org/10.56471/slujst.v4i.266*. – DOI 10.56471/slujst.v4i.266

[BAM19] BALTRUSAITIS, T. ; AHUJA, C. ; MORENCY, L.-P.: Multimodal Machine Learning: A Survey and Taxonomy. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2019), Feb, Nr. 2, S. 423–443. *http://dx.doi.org/10.1109/TPAMI.2018.2798607*. – DOI 10.1109/TPAMI.2018.2798607

[BLS+24] BORISOV, V. ; LEEMANN, T. ; SESSLER, K. ; HAUG, J. ; PAWELCZYK, M. ; KASNECI, G.: Deep Neural Networks and Tabular Data: A Survey. In: *IEEE Trans. Neural Netw. Learning Syst.* 35 (2024), Jun, Nr. 6, S. 7499–7519. *http://dx.doi.org/10.1109/TNNLS.2022.3229161*. – DOI 10.1109/TNNLS.2022.3229161

[Bon24] BONNIER, T.: Revisiting Multimodal Transformers for Tabular Data with Text Fields. In: KU, L.-W. (Hrsg.) ; MARTINS, A. (Hrsg.) ; SRIKUMAR, V. (Hrsg.): *Findings of the Association for Computational Linguistics: ACL 2024*. Bangkok, Thailand : Association for Computational Linguistics, Aug 2024, S. 1481–1500

[Bro18] BROOKS, Nick: *Women's E-Commerce Clothing Reviews*. *https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews*. Version: 2018. – Zugriff über Kaggle

[BSP23] BADARO, G. ; SAEED, M. ; PAPOTTI, P.: Transformers for Tabular Data Representation: A Survey of Models and Applications. In: *Transactions of the Association for Computational Linguistics* 11 (2023), S. 227–249. *http://dx.doi.org/10.1162/tacl_a_00544*. – DOI 10.1162/tacl_a_00544

[CDP+23] CHEN, Xi ; DJOLONGA, Josip ; PADLEWSKI, Piotr ; BASILICO, Basil ; FEDUS, William ; HOULSBY, Neil: PaLI: A Jointly-Scaled Multilingual Language-Image Model. (2023), Jun. *http://dx.doi.org/10.48550/arXiv.2209.06794*. – DOI 10.48550/arXiv.2209.06794

[CSM20] CHAN, B. ; SCHWETER, S. ; MÖLLER, T.: German's Next Language Model. (2020), Dec. *http://dx.doi.org/10.48550/arXiv.2010.10906*. – DOI 10.48550/arXiv.2010.10906

[DBK+21] DOSOVITSKIY, Alexey ; BEYER, Lucas ; KOLESNIKOV, Alexander ; WEISSENBORN, Dirk ; ZHAI, Xiaohua ; UNTERTHINER, Thomas ; DEHGHANI, Mostafa ; MINDERER, Matthias ; HEIGOLD, Georg ; GELLY, Sylvain ; USZKOREIT, Jakob ; HOULSBY, Neil: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: *International Conference on Learning Representations*, 2021

[DCLT19] DEVLIN, Jacob ; CHANG, Ming-Wei ; LEE, Kenton ; TOUTANOVA, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019), May. *http://dx.doi.org/10.48550/arXiv.1810.04805*. – DOI 10.48550/arXiv.1810.04805

[GB21] GU, K. ; BUDHKAR, A.: A Package for Learning on Tabular and Text Data with Transformers. In: ZADEH, A. (Hrsg.) ; MORENCY, L.-P. (Hrsg.) ; LIANG, P. P. (Hrsg.) ; ROSS, C. (Hrsg.) ; SALAKHUTDINOV, R. (Hrsg.) ; PORIA, S. (Hrsg.) ; CAMBRIA, E. (Hrsg.) ; SHI, K. (Hrsg.): *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*. Mexico City, Mexico : Association for Computational Linguistics, Jun 2021, S. 69–73

[GRKB21] GORISHNIY, Y. ; RUBACHEV, I. ; KHRULKOV, V. ; BABENKO, A.: Revisiting Deep Learning Models for Tabular Data. In: *Advances in Neural Information Processing Systems* (2021)

[HKCK20] HUANG, X. ; KHETAN, A. ; CVITKOVIC, M. ; KARNIN, Z.: TabTransformer: Tabular Data Modeling Using Contextual Embeddings. (2020), Dec. *http://dx.doi.org/10.48550/arXiv.2012.06678*. – DOI 10.48550/arXiv.2012.06678

[JGF+24] JIAO, T. ; GUO, C. ; FENG, X. ; CHEN, Y. ; SONG, J.: A Comprehensive Survey on Deep Learning Multi-Modal Fusion: Methods, Technologies and Applications. In: *CMC-Computers, Materials & Continua* 80 (2024), Nr. 1, S. 1–35. *http://dx.doi.org/10.32604/cmc.2024.053204*. – DOI 10.32604/cmc.2024.053204

[LT24] LI, S. ; TANG, H.: Multimodal Alignment and Fusion: A Survey. (2024). *http://dx.doi.org/10.48550/arXiv.2411.17040*. – DOI 10.48550/arXiv.2411.17040. – arXiv:2411.17040

[MCCD13] MIKOLOV, Tomas ; CHEN, Kai ; CORRADO, Greg ; DEAN, Jeffrey: Efficient Estimation of Word Representations in Vector Space. (2013), Sep. *http://dx.doi.org/10.48550/arXiv.1301.3781*. – DOI 10.48550/arXiv.1301.3781

[Pet19] PETFINDER.MY: *PetFinder.my Adoption Prediction*. *https://www.kaggle.com/c/petfinder-adoption-prediction*. Version: 2019. – Kaggle Competition

[PNI+18] PETERS, Matthew E. ; NEUMANN, Mark ; IYYER, Mohit ; GARDNER, Matt ; CLARK, Christopher ; LEE, Kenton ; ZETTLEMOYER, Luke: Deep Contextualized Word Representations. In: WALKER, Marilyn (Hrsg.) ; JI, Heng (Hrsg.) ; STENT, Amanda (Hrsg.): *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana : Association for Computational Linguistics, Jun 2018, S. 2227–2237

[SB88] SALTON, Gerard ; BUCKLEY, Chris: Term-weighting approaches in automatic text retrieval. In: *Information Processing & Management* 24 (1988), Nr. 5, S. 513–523. *http://dx.doi.org/10.1016/0306-4573(88)90021-0*. – DOI 10.1016/0306–4573(88)90021–0

[SGS+21] SOMEPALLI, G. ; GOLDBLUM, M. ; SCHWARZSCHILD, A. ; BRUSS, C. B. ; GOLDSTEIN, T.: SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. In: *arXiv preprint arXiv:2106.01342* (2021). *https://arxiv.org/abs/2106.01342v1*

[TB19] TAN, H. ; BANSAL, M.: LXMERT: Learning Cross-Modality Encoder Representations from Transformers. (2019), Dec. *http://dx.doi.org/10.48550/arXiv.1908.07490*. – DOI 10.48550/arXiv.1908.07490

[VSP+17] VASWANI, Ashish ; SHAZEER, Noam ; PARMAR, Niki ; USZKOREIT, Jakob ; JONES, Llion ; GOMEZ, Aidan N. ; KAISER, Łukasz ; POLOSUKHIN, Illia: Attention is All you Need. In: *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017

[YNYR20] YIN, Pengcheng ; NEUBIG, Graham ; YIH, Wen-tau ; RIEDEL, Sebastian: TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. (2020), May. *http://dx.doi.org/10.48550/arXiv.2005.08314*. – DOI 10.48550/arXiv.2005.08314

A1 Anhang

A1.1 Quellcode der Baseline-Modelle

Der folgende Quellcode zeigt die PyTorch-Implementierung der verwendeten Baseline-Modelle *ConcatBERT* und *SumBERT*. Beide Klassen erben von `BertForSequenceClassification` und passen den Forward-Pass entsprechend der jeweiligen Fusionsstrategie an.

```
class ConcatBert(BertForSequenceClassification):
    """Custom BERT that handles numerical and one-hot encoded categorical
       data by concatenating it to the output embedding
    of the CLS token.

    Attributes:
    ----------
    classifier: ClassificationHead
        A classification module that is identical to the one in
           BertForSequenceClassification, except for added input dimensions
            from the numerical data.

    Methods:
    ---------
    forward(...):
        Takes in input ids, attention mask, and a tensor of extra data to
           compute the binary output. Input ids and attention mask are
           first fed through the base bert model.
        The resulting CLS context embedding is then concatenated to the
           extra data tensor and fed through the classificationHead to get
           the final output.
    """

    def __init__(self, cfg: BertConfig, extra_data_dim: int, hidden_dim:
       int):
        super().__init__(cfg)

        class ClassificationHead(nn.Module):
            """Head for sentence-level classification tasks."""

            def __init__(self, cfg, extra_data_dim, hidden_dim):
                super().__init__()
                total_dims = cfg.hidden_size + extra_data_dim
                assert (
                    hidden_dim <= total_dims
                ), "The hidden dim should be less or equal than the input
                   embedding"
```

```
            self.dense = nn.Linear(total_dims, hidden_dim)
            self.bn1 = nn.BatchNorm1d(hidden_dim)
            classifier_dropout = (
                cfg.classifier_dropout
                if cfg.classifier_dropout is not None
                else cfg.hidden_dropout_prob
            )
            self.dropout = nn.Dropout(classifier_dropout)
            self.out_proj = nn.Linear(hidden_dim, cfg.num_labels)

        def forward(self, features, **kwargs):
            x = self.dropout(features)
            x = self.dense(x)
            x = self.bn1(x)
            x = torch.relu(x)
            x = self.dropout(x)
            x = self.out_proj(x)
            return x

    self.classifier = ClassificationHead(cfg, extra_data_dim,
        hidden_dim)
    # https://tfs/web/DefaultCollection/GIT_Projects/_git/hf4-regress-
        exploration/pullRequest/81216#1737385342
    self.post_init()

def forward(
    self,
    input_ids: torch.Tensor | None = None,
    attention_mask: torch.Tensor | None = None,
    token_type_ids: torch.Tensor | None = None,
    position_ids: torch.Tensor | None = None,
    head_mask: torch.Tensor | None = None,
    inputs_embeds: torch.Tensor | None = None,
    labels: torch.Tensor | None = None,
    output_attentions: bool | None = None,
    output_hidden_states: bool | None = None,
    return_dict: bool | None = None,
    extra_data: torch.Tensor | None = None,
) -> tuple | SequenceClassifierOutput:

    return_dict = (
        return_dict if return_dict is not None else self.config.
            use_return_dict
    )

    outputs = self.bert(
```

```
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        # Using the pooler output as document embedding. This corresponds
        # to a post-processed version of the <CLS> token embedding.
        cls_embedding = outputs.pooler_output

        if extra_data is not None:
            output = torch.cat((cls_embedding, extra_data), dim=-1)
        else:
            raise ValueError("Must supply extra data!")
        logits = self.classifier(output)
        if not return_dict:
            output = (logits,) + outputs[2:]
            return output

        return SequenceClassifierOutput(
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


class SumBert(BertForSequenceClassification):
    """
    SumBert integrates additional feature-embeddings with a BERT base model
        used for a classification task.

    Parameters
    ----------
    feature_layer:
      Linear layer used to add the numerical and categorical data to the
         word embedding of the CLS token.

    Methods
    ---------
    forward(...):
```

```
        Gets word embeddings from input ids through bert. The adds the
            output of its feature layer to the CLS word embedding to insert
            numerical / categorical data.
        Subsequently uses BertModels forward function to get outputs.
    """

    def __init__(self, cfg: BertConfig, extra_data_dim: int):
        super().__init__(cfg)
        self.feature_layer = nn.Linear(extra_data_dim, 768)

    def forward(
        self,
        input_ids: torch.Tensor | None = None,
        attention_mask: torch.Tensor | None = None,
        token_type_ids: torch.Tensor | None = None,
        position_ids: torch.Tensor | None = None,
        head_mask: torch.Tensor | None = None,
        inputs_embeds: torch.Tensor | None = None,
        labels: torch.Tensor | None = None,
        output_attentions: bool | None = None,
        output_hidden_states: bool | None = None,
        return_dict: bool | None = None,
        extra_data: torch.Tensor | None = None,
    ):
        # Avoid performing computation under torch.no_grad() to ensure
        # word embeddings can be updated during training.
        text_embedding = self.bert.embeddings.word_embeddings(input_ids)
        text_embedding[:, 0] += self.feature_layer(extra_data)
        outputs = self.bert(
            inputs_embeds=text_embedding,
            attention_mask=attention_mask,
        )
        pooled_output = outputs.pooler_output
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return SequenceClassifierOutput(
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

Code 1: PyTorch-Implementierung von ConcatBERT und SumBERT

A1.2 Quellcode der CrossBERT-Architektur

Der folgende Quellcode enthält die PyTorch-Implementierung der in Abschnitt 4.3 beschriebenen CrossBERT-Architektur, einschließlich *TabularTokenizer* und *CrossAttentionBlock*.

```
class CrossBert(BertForSequenceClassification):
    """BERT classifier with Early and Late cross-attention over tabular tokens."""

    def __init__(self, cfg: BertConfig, extra_data_dim: int, cross_attention_positions: Dict[
        str, bool],
                 num_tab_tokens: int = 4, cross_attention_heads: Optional[int] = None):
        super().__init__(cfg)
        self.tabular_tokenizer = TabularTokenizer(num_features=extra_data_dim, hidden_size=cfg
            .hidden_size,
                                                  num_tab_tokens=num_tab_tokens, dropout=cfg.
                                                      hidden_dropout_prob)
        self.cross_attn_early = CrossAttentionBlock(embed_dim=768, num_heads=8) if
            cross_attention_positions.get("early") else None
        self.cross_attn_late = CrossAttentionBlock(embed_dim=768, num_heads=8) if
            cross_attention_positions.get("late") else None
        self.post_fusion_ln = nn.LayerNorm(cfg.hidden_size)
        self.post_fusion_dropout = nn.Dropout(cfg.hidden_dropout_prob)

    def forward(self, input_ids=None, attention_mask=None, extra_data=None, tab_mask=None, **
        kwargs):
        tab_tokens = self.tabular_tokenizer(extra_data) if extra_data is not None else None
        embeddings = self.bert.embeddings(input_ids=input_ids)

        if self.cross_attn_early and tab_tokens is not None:
            cls_early = self.cross_attn_early(embeddings[:, :1, :], tab_tokens, tab_mask=
                tab_mask)
            embeddings = torch.cat([cls_early, embeddings[:, 1:, :]], dim=1)

        encoder_outputs = self.bert.encoder(hidden_states=embeddings,
                                            attention_mask=self.bert.
                                                get_extended_attention_mask(attention_mask,
                                                attention_mask.shape, attention_mask.device))
        cls_token = encoder_outputs.last_hidden_state[:, :1, :]

        if self.cross_attn_late and tab_tokens is not None:
            cls_token = self.cross_attn_late(cls_token, tab_tokens, tab_mask=tab_mask)

        fused_cls = self.post_fusion_dropout(self.post_fusion_ln(cls_token)).squeeze(1)
        return SequenceClassifierOutput(logits=self.classifier(fused_cls))

class CrossAttentionBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.attention = nn.MultiheadAttention(embed_dim, num_heads, dropout=dropout,
            batch_first=True)
        self.layer_norm_1 = nn.LayerNorm(embed_dim)
        self.ffn = nn.Sequential(nn.Linear(embed_dim, dim_feedforward), nn.GELU(), nn.Dropout(
            dropout), nn.Linear(dim_feedforward, embed_dim))
        self.layer_norm_2 = nn.LayerNorm(embed_dim)
        self.dropout_attn, self.dropout_ffn = nn.Dropout(dropout), nn.Dropout(dropout)

    def forward(self, cls_token, tab_tokens, tab_mask=None):
```

```
        q, k = self.layer_norm_1(cls_token), self.layer_norm_1(tab_tokens)
        attn_out, _ = self.attention(q, k, k, key_padding_mask=(tab_mask == 0) if tab_mask is
            not None else None)
        x = cls_token + self.dropout_attn(attn_out)
        return x + self.dropout_ffn(self.ffn(self.layer_norm_2(x)))

class TabularTokenizer(nn.Module):
    def __init__(self, num_features, hidden_size, num_tab_tokens=8, dropout=0.1):
        super().__init__()
        self.projections = nn.ModuleList([nn.Sequential(nn.Linear(num_features, hidden_size),
            nn.GELU(),
                                                        nn.Dropout(dropout), nn.LayerNorm(
                                                            hidden_size)) for _ in range(
                                                            num_tab_tokens)])

    def forward(self, tabular_input):
        return torch.cat([proj(tabular_input).unsqueeze(1) for proj in self.projections], dim
            =1)
```

Code 2: PyTorch-Implementierung von CrossBERT

HOCHSCHULE COBURG

Persönliche Angaben / *Personal details*

Städler, Sebastian
Familienname, Vorname / *Surnames, given names*

19.03.2002
Geburtsdatum / *Date of birth*

Informatik
Studiengang / *Course of study*

00383022
Matrikelnummer / *Student registration number*

Eigenständigkeitserklärung

Declaration

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt habe. Ich habe keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt. Die Arbeit wurde weder in Gänze noch in Teilen von einer Künstlichen Intelligenz (KI) erstellt, es sei denn, die zur Erstellung genutzte KI wurde von der zuständigen Prüfungskommission oder der bzw. dem zuständigen Prüfenden ausdrücklich zugelassen. Wörtliche oder sinngemäße Zitate habe ich als solche gekennzeichnet.
Es ist mir bekannt, dass im Rahmen der Beurteilung meiner Arbeit Plagiatserkennungssoftware zum Einsatz kommen kann.
Es ist mir bewusst, dass Verstöße gegen Prüfungsvorschriften zur Bewertung meiner Arbeit mit „nicht ausreichend“ und in schweren Fällen auch zum Verlust sämtlicher Wiederholungsversuche führen können.

I hereby certify that I have written this thesis independently and have not submitted it elsewhere for examination purposes. I have not used any sources or aids other than those indicated The work has not been created in whole or in part by an artificial intelligence (AI), unless the AI used to create the work has been expressly approved by the responsible examination board or examiner. I have marked verbatim quotations or quotations in the spirit of the text as such.

I am aware that plagiarism detection software may be used in the assessment of my work.

I am aware that violations of examination regulations can lead to my work being graded as "unsatisfactory" and, in serious cases, to the loss of all repeat attempts.

Unterschrift Studierende/Studierender / *Signature student*

96450 Coburg, den 05.01.2026
Ort, Datum / *Place, date*