Universität Bremen

Masterarbeit

im Fachbereich 3 - Mathematik und Informatik
Universität Bremen

Automatische Fehlersuche bei Montageprozessen auf semantischer Ebene

automatic error detection of assembly process using semantic knowledge

Klaus Prüger

Matrikel-Nummer: 4510399
Studiengang: Informatik

Bremen, den 19. Dezember 2024

Erstgutachter:	Prof. Dr.-Ing. Udo Frese	FB3 | AG MSIS
Zweitgutachter:	Prof. Dr.-Ing. Michael Freitag	FG 20, FB4 | BIBA
Betreuer:	M.Sc. Dirk Schweers	FG 20, FB4 | BIBA

Kurzfassung

Die Montage stellt einen zentralen Punkt der Produktion von Produkten dar. Sich ändernde Kundenbedürfnisse erzeugen eine steigende Produktvielfalt, bei kürzeren Produkt Lebenszyklen. Damit geht ein steigender Kostendruck einher. Das analysieren von Montagedaten, zur Findung von Optimierungen gewinnt an Bedeutung. Ein Teil ist das erkennen von Montagefehlern. Bisher wird die Struktur aber nicht die Semantik der Daten berücksichtigt. Die Ontologie *buPAOv1* (BIBA-Uni-Prueger-Assembly-Ontology- Version-1) setzt die Montagedaten semantisch in Beziehung. Der eigens entwickelte Tool *Data Handler*, überführt die Montagedaten in einen Datensatz von Wissensgraphen. Das GraphSAGE Convolution Embedding Modell lernt Unsupervised die Gemeinsamkeiten und gibt das Knoten Embedding aus. Das Siamese Modell führt Graph Matching für zwei Sample aus, sie werden auf ihre Ähnlichkeit verglichen. Der Score von 0.0 bis 1.0 beschreibt die Ähnlichkeit, von verschieden bis identisch. Diskutiert wird, in wieweit erkannte Ähnlichkeiten Aufschluss über die Fehlererkennung geben.

Schlagwörter Ontologie, Montage, Graph Matching, Deep Learning, Wissensgraph, Unsupervised Learning, Heterogener Graph

Assembly is a central point in the production of products. Changing customer requirements are creating an increasing variety of products with shorter product life cycles. This is accompanied by increasing cost pressure. Analyzing assembly data to find optimizations is becoming increasingly important. One part of this is the detection of assembly errors. Up to now, the structure but not the semantics of the data has been taken into account. The ontology buPAOv1 (BIBA-Uni-Prueger-Assembly-Ontology- Version-1) relates the assembly data semantically. The specially developed Tool Data Handler transfers the assembly data into a data set of knowledge graphs. The GraphSAGE Convolution Embedding Model learns the similarities unsupervised and outputs the node embedding. The Siamese model performs graph matching for two samples, they are compared for their similarity. The score from 0.0 to 1.0 describes the similarity, from different to identical. The extent to which recognized similarities provide information about error detection is discussed.

Inhaltsverzeichnis

1 Einleitung

In den folgenden Kapiteln wird die Idee der Anwendung von Deep Learning und semantischem Wissen zur Untersuchung von Montageprozessen skizziert. Kapitel 1.1 greift die Motivation hinter der Untersuchung von Montageprozessen auf. Kapitel 1.2 bündelt die Idee in ein konkretes Forschungsziel. Die Schritte der Untersuchung beschreibt Kapitel 1.3. Das Kapitelende stellt den Aufbau der Arbeit vor.

1.1 Motivation

Die Montage (auch Montageprozess genannt) eines Produktes beschreibt das Zusammensetzen der Bauteile von Menschen (maschinengestützt) zu einem fertigen Produkt. Die Montage gliedert sich in Montageschritte, ein Schritt ist eine Tätigkeit, wie das Verbinden zweier Bauteile mit einer Schraube.

Die Montage ist ein Teil der Produktion eines Produktes und mit 80 Prozent stellt die Montage den größten Teil der Kosten der Produktion eines Produktes dar. Entsprechend relevant sind die Bemühungen, die Kosten des Montageprozesses zu verringern. Ergriffene Maßnahmen sind, bestehende Montageprozesse automatisch zu optimieren bzw. effizienter zu gestalten, indem Schritte ersetzt, kompakter oder parallelisiert werden. Doch in erster Linie sollen Prozessschritte wiederverwendbar gestaltet werden für andere Produkte (Lohse et al., 2005). Gleichzeitig sinken die Lebenszyklen der Produkte[1], während die Produktvielfalt wächst, um den Kundenbedürfnissen gerecht zu werden. Dies führt zu geringeren Stückzahlen und einem damit einhergehend steigenden Preisdruck. Es erfordert eine steigende Flexibilität der Produktion, getrieben durch die Anforderungen, was die Komplexität in allen Belangen erhöht. Um dem gerecht zu werden, unterstützen Werkerassistenzsysteme die Werker in der Montage, indem sie Montageschritte anleiten und weitere Hilfestellungen in textueller und visueller Form bieten. Sie erhöhen die Zuverlässigkeit, überprüfen die Qualität der Montageschritte und

[1] Produktlebenszyklus ist die Zeitspanne von der Einführung bis Absetzung eines Produktes im Markt.

bieten Schnittstellen zum vereinfachten Melden von aufgetretenen Fehlern. Die für die Qualität erhobenen Messdaten können in Key Performance Indicators, kurz KPIs, einfließen und die Arbeitslast steuern. Die Belastungssteuerung reduziert psychischen Stress. Darüber hinaus kann ein Werkerassistenzsystem bei körperlich schweren Arbeitsschritten entlasten, wie die Handhabung von schweren Bauteilen (Keiser et al., 2023).

Als Produktvarianten werden Produkte mit stark ähnlicher Form bzw. Funktion bezeichnet. Produktvarianten weisen zahlreiche identische Baugruppen sowie Bauteile auf und besitzen meist dieselbe Grundfunktion. Produktvarianten unterscheiden sich in mindestens einer Eigenschaft. Tritt ein Fehler in einem Montageprozess eines Produktes auf, kann dies als Indikator gesehen werden, bei Produkten ähnlicher Bauanleitung zu schauen, ob dieser Fehler auch bei Ihnen auftritt. Die Herausforderung in der übertragenden Erkennung von Fehlern, besteht in der bisher fehlenden Verbindung der Montageprozesse verwandter Produkte. Die strukturelle Ähnlichkeit, also dieselbe Montageabfolge, reicht nur für begrenzte Aussagen in der Übertragbarkeit der Fehler. So lässt sie keine Aussage darüber zu, ob dieselben Bauteile oder Werkzeuge, Werkzeugträger oder allgemeine Peripherie verwendet wurden. Dazu kommt, die Anpassung des Montageprozesses zur Schaffung neuer Produktvarianten erfolgt meist händisch und fehlende Richtlinien haben zur Folge, dass gleiche Montageschritte von unterschiedlichen Varianten von Personen anders benannt werden. Was die Erkennung der Fehlerübertragbarkeit weiter erschwert.

Das Problem mündet heruntergebrochen in einem NP-Problem, das zum Stand der Veröffentlichung dieser Arbeit nicht gelöst werden konnte. In anderen Worten, es existiert kein exakter Algorithmus, der ein NP-Problem in polynomieller Zeit löst. Infolgedessen gibt es approximierte Algorithmen, die sich dem Entscheidungsproblem aus praktischer Sicht annehmen, jedoch nur in eingegrenzten Fällen verwendbare Ergebnisse produzieren können.

Die strukturelle Betrachtungsebene ist hilfreich zur Darstellung des Ablaufs, aber nicht ausreichend für Aussagen über die Übertragbarkeit von Fehlern. Erweitert man die Betrachtungsweise um eine semantische Ebene, setzt man die Bauteile und Montageschritte der Montage verschiedener Produkte in Beziehung zueinander und ermöglicht damit, Ähnlichkeiten zu identifizieren, trotz ihrer Unterschiede.

Neuste Entwicklungen eröffnen vielversprechende Ansätze, die auch bei der Fehlersuche in Montagedaten zum Tragen kommen könnten. Anstelle das NP-Problem generell zu lösen, nähert man sich dem Problem auf eine Genauigkeit, bis für Eingaben aus praktischer Sicht verlässliche Aussagen getroffen werden können. Konkret wird von Daten und Beispielen generalisiert, sodass ein Algorithmus (Deep Learning Modell) von praktischem Nutzen entsteht, der die gegebenen Zusammenhänge der Domäne versteht. Untersuchungen in verschiedens-

ten Disziplinen, unter anderem auf Wissensgraphen, übertreffen lineare Algorithmen beim Finden von Ähnlichkeiten auf Graphen (Ji et al., 2022).

Begründet aus der Motivation heraus ergeben sich folgende Ziele, beschrieben im nächsten Kapitel 1.2.

1.2 Ziel

Da stellt sich die Frage bzw. keimt die Hoffnung auf, ob das semantische Wissen über Montageprozesse mit dem Deep Learning Ansatz bei der Fehlererkennung hilft und eine präventive Aussagekraft für verwandte Produkte zulässt? Es gilt die Vermutung, dass semantisches Wissen über Montageprozesse die Fehlererkennung bei verwandten Produkten ermöglicht. Es gilt also zu widerlegen: Semantisches Wissen über Montageprozesse lässt keine Aussage über Produktionsfehler in der Kette zu.

Es gilt also zu untersuchen, ob der Ansatz auf echten Industriedaten hält. In einem überschaubaren Rahmen soll diese Arbeit praktisch demonstrieren, dass ein Graph Deep Learning Modell, mit der Hilfe eines Wissensgraphen, Ähnlichkeiten in den Montageprozessen verwandter Produkte findet, wobei die Anwendungsfälle aus echten Montageindustriedaten abgeleitet werden.

Es wird erwartet, dass die Ergebnisse eine Abschätzung ermöglichen, ob der Ansatz in der Montageindustrie Bestand hat, also ob dieser weiterverfolgt werden sollte, um in einer praktischen Anwendungsgröße zu skalieren. Dabei sollten potenzielle technische, konzeptionelle oder durch diesen Rahmen verursachte Limitierungen des Ansatzes deutlich werden. Wie die Metriken zur Analyse erhoben und eine geeignete Anwendung ermittelt wird, folgt im nächsten Kapitel 1.3.

1.3 Methodik und Aufbau der Arbeit

Das Vorgehen ist praktisch, lösungsorientiert. Eine konzipierte Anwendung soll Ergebnisse für vorhandene Montagedaten liefern. Die Entwicklung eines praktischen Systems zur Überprüfung des genannten Konzepts gliedert sich in drei Schritte. Erstens, das semantische Wissen über vorhandene Montageprozesse aufbereiten. Zweitens, die Bestimmung eines geeigneten Deep Learning Modells, dass das semantische Wissen der Montageprozesse in Beziehung zueinander setzt. Abschließend die Metriken für die Ähnlichkeiten Erkennung umsetzen.

Allem vorweg steht die wissenschaftliche Recherche. Die literarische Grundlage zeigt mögliche Lösungswege und Konzepte auf. Für alle genannten Komponenten: semantisches Wissen, Deep Learning und Vergleichs-Metriken, werden die Möglichkeiten evaluiert. Etablierte Konzepte werden auf Ihre Anwendbarkeit für die Montagedaten, Domäne und Fragestellung hin untereinander verglichen.
Anschließend folgen die drei Schritte der praktischen Systemumsetzung. Die Aufbereitung der

Montageprozesse bzw. Überführung in semantisches Wissen beinhaltet eine Analyse der Montagedaten. Es gilt, gemeinsame Muster zu erkennen und sie in ein übergeordnetes Konzept abzubilden. Das übergeordnete Konzept beeinflusst maßgeblich die Wahl des vielversprechendsten Deep Learning Modells. Das Modell soll dem Anwendungsfall und den aufbereiteten Montagedaten genügen. Die Architektur des Modells wird an die Bedürfnisse der Datenstruktur angepasst. Die Ausgabe des Modells verwendet die Metrik zur Bestimmung von Ähnlichkeit zwischen zwei Montageprozessen. Entsprechend muss die Metrik mit der Ausgabe des Deep Learning Modells umgehen können und da ein Vergleich gefordert ist, entsprechend eine vergleichende Aussage über die Eingabe treffen.

Abschließend gilt, anhand der Ergebnisse die Leistungsfähigkeit der Anwendung zu ermitteln. Es gliedert sich in zwei Schritte. Die Bewertung des erlernten semantischen Verständnis des Deep Learning Modells und der Nutzen, den das Deep Learning Modell in der Ähnlichkeiten Erkennung erbringt. Dafür werden gezielt Montageprozesse anhand ihrer Eigenschaften gewählt.

Die Arbeit gliedert sich folgendermaßen: Kapitel 2 führt technische Begriffe und Konzepte ein, vergleicht sie hinsichtlich deren Nutzen für die Umsetzung. Basierend auf den Ergebnissen stellt das Kapitel 3 die Montagedaten getriebene Umsetzung vor. Es beinhaltet die Vorstellung der Montagedaten und den Entwurf einer auf den Montagedaten abgestimmten Ontologie, die die Montagedaten in Beziehung setzt. Die Vorstellung des gewählten Modells zum Erlernen des Embedding. Gefolgt von der Einführung der Vergleichs-Metriken zum Graph Matching. Das umgesetzte Konzept wird auf seine Tauglichkeit geprüft, dies passiert in zwei Schritten, beschrieben in Kapitel 4. Erstens wird die Embedding Modellqualität untersucht. Anschließend die Erkenntnisse der erkannten Ähnlichkeiten des Graph Matching für ausgewählte Sample. Die Arbeit schließt mit einem Fazit in Kapitel 5, indem sie die Ergebnisse zusammenfasst und weiterführende Forschungsziele vorstellt.

2 Stand der Forschung

Das Kapitel führt die technischen Grundlagen und Begriffe ein, die aus der Motivation hervorgehen. Ontologie beschreibt das Konzept, Daten in Beziehung zu setzen. Die Graphdatenbank stellt Techniken zur effizienten Verwaltung der semantisch erzeugten Daten dar. Die Möglichkeiten des Vergleichs von Graphen beschreibt Graph Matching. Ein Teil der Graph Matching Konzepte setzt das embedden der Graphen voraus, es reduziert die Graphen auf dessen wesentlichen Eigenschaften und gibt sie in einem effizienten Format aus. Wie dies konkret geschieht, beschreibt das Graph Embedding.

2.1 Ontologie

Ontologie beschreibt die Beziehung von Konzepten als einen Graphen. Die Beziehungen sind gerichtete Kanten. Konzepte repräsentieren Informationen. Konzepte können auch Attribute sein, sie beschreiben Eigenschaften von Konzepten. Konkrete Instanzen der Ontologie werden als Wissensgraphen betitelt. Wissensgraphen instanziieren Konzepte, setzen Beziehungen und konkrete Werte für Attribute ein, nach dem Schema der Ontologie. Verbreitete Darstellungsformen sind grafisch und textuell.
Eine standardisierte textuelle Beschreibung bietet das *Resource Description Framework* (kurz *RDF* (W3C RDF Working Group, 2024)) des World Wide Web Consortium (kurz *W3C*). Es ist ein offener Web Standard, der das Schema über *Internationalized Resource Identifier* kurz *IRI* beschreibt. Kernelemente sind die *RDF Triple*, auch *Triple*, bestehend aus *Subject*, *Predicate* und *Object*. Die kompakte Schreibweise ist *(s, p, o)*. Sie beschreiben eine gerichtete Beziehung zweier Konzepte. Startknoten ist das *Subject*, *Predicate* die Beziehung und *Object* der Zielknoten.
Eine *resources*, auch als *entity* bezeichnet, ist vom Typ *IRI* oder *Literal* und kann „Alles“ sein, darunter ein Wert, physisches Objekt, Text und Konzept. Ein *Subject* repräsentiert eine *resources* vom Typ *IRI*. *Predicate* ebenfalls repräsentiert als *IRI*, beschreibt ein *property*, das als binäre Relation gesehen werden kann. *Object* repräsentiert *resources* vom Typ *IRI* oder *Literal*. *Literal* kann als Attribut angesehen werden, das den Datentyp und Wertebereich vorgibt. Die

IRIs setzen sich aus einem gemeinsamen Teilwort (*Namespace IRI*), gefolgt von einem Wort, das die *resources* repräsentiert, zusammen. Eine Menge von *Tripeln* ergibt einen *RDF Graph*.

Zur Verdeutlichung folgt ein Beispiel, das die grafische Darstellung mit einbezieht (Illustriert in Abbildung 1). Die folgende Ontologie modelliert die Beziehung von Menschen. Identifier sind, *Person*, *Erwachsener* und *Kind*. Literal ist, *istJahreAlt* (Datentyp Integer). Beziehungen sind, *unterklasseVon*, *verheiratet*, *geschieden*, *elternteilVon*. Das Literal gehört dem Identifier *Person*, dies erben *Kind* und *Erwachsener* als Unterklassen von *Person*. Ein *Erwachsener* kann *geschiedenVon* oder *verheiratetMit* sein, mit einem anderen *Erwachsener*. Darüber hinaus kann eine *Person* von einem Kind ein Elternteil sein. Der definierte Namespace den die *IRIs* zur Beschreibung der Ontologie verwenden, lautet: http://www.demo.ontologie.com/beziehung-von-menschen/v1#. Die textuell formale Beschreibung im RDF Format befindet sich im Anhang A.1.

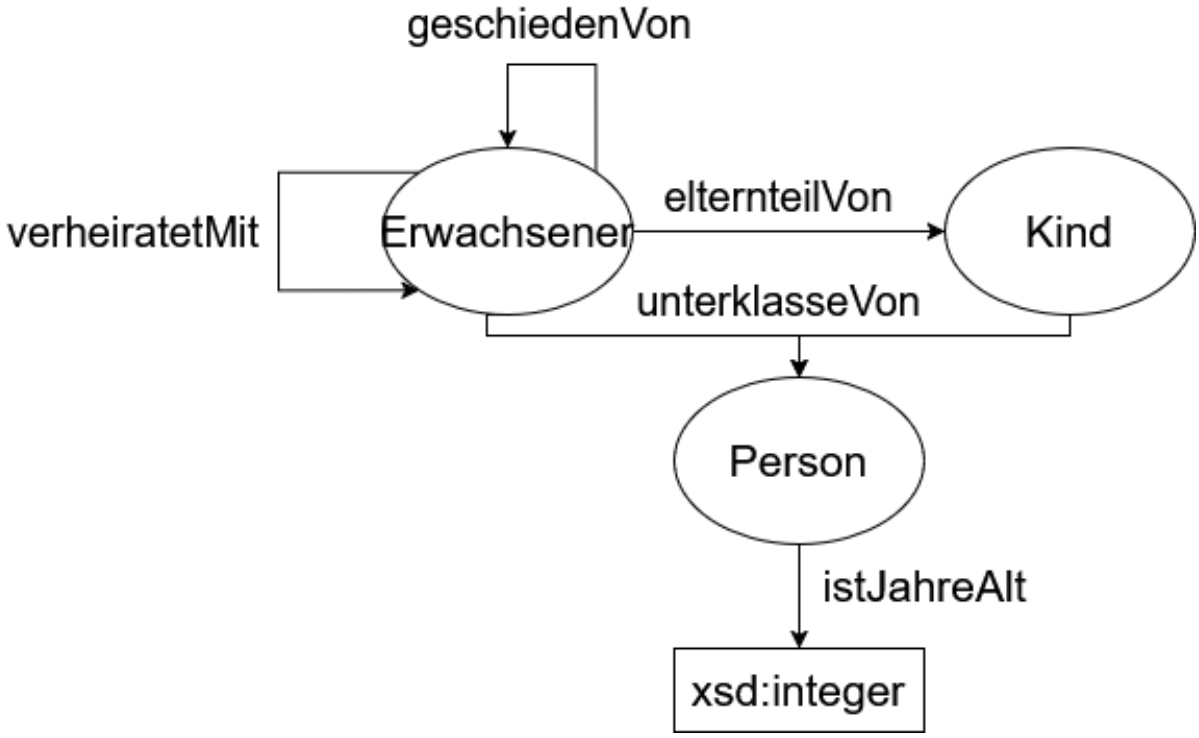


Abb. 1: Die Menschen Ontologie. Sie modelliert die Beziehung von Menschen.

Konkret modelliert wird eine Familie, sie besteht aus drei Kindern, *Bob* (9), *Eliza* (5) und *Eva* (11), und drei Erwachsenen, Tim (45), Jim (40) und Pam (42). *Jim* und *Pam* sind *verheiratetMit* und haben zwei Kinder, *Bob* und *Eliza*. *Tim* und *Pam* sind *geschiedenVon* und haben ein Kind, *Eva*. Illustriert in Abbildung (2). Die textuell formale Beschreibung im RDF Format befindet sich im Anhang A.2. Es gibt noch weitere syntaktische Dialekte: N-Triples, Turtle, TriG, N-Quads und RDFa. Sie sind angepasst an andere Formate oder Anwendungsfälle und stechen durch Ihre Eigenschaften Kompaktheit oder Lesbarkeit heraus. Gleichwertig und ineinander

überführbar (W3C RDF Working Group, 2023).

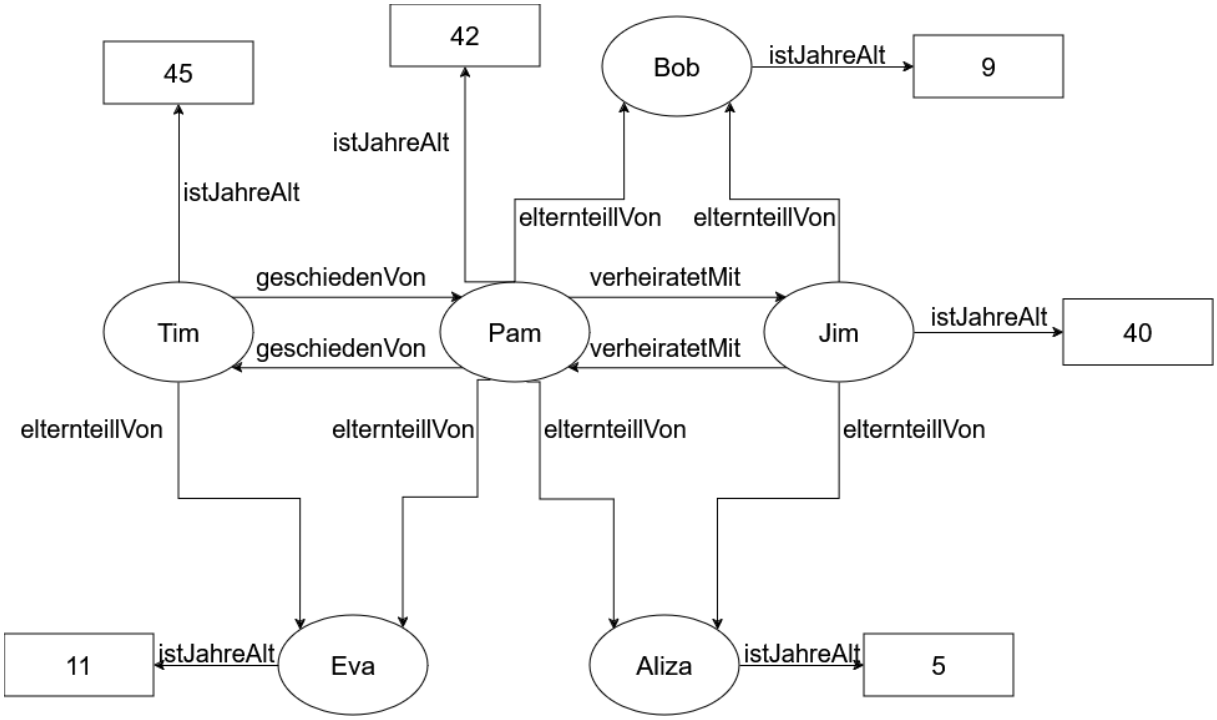


Abb. 2: Wissensgraph Instanz der Beziehung von Menschen Ontologie. Es ist eine vereinfachte Darstellung, die aus Übersichtsgründen auf technische Details verzichtet. Literale werden durch Rechtecke dargestellt, IRIs durch Ovale und Kanten als gerichtete Pfeile zwischen Subject und Object. Leserichtung der Triple ist Subject, ausgehender Pfeil zeigt auf Object.

Produktion Ontologie. Ontologien der Produktion bzw. Montage teilen Konzepte. Im Zentrum steht die Beziehung von Bauteilen. Ihre Wechselwirkung (Kontaktpunkte) und Degree of Freedom. Die Werte liefern CAD (Computer-Aided Design) Daten. Eingebettet sind die Beziehungen in Montagesequenzen. Die Ontologie gestaltet sich hierarchisch, von der Beschreibung grober Konzepte, wie der Montage, hin zu kleineren Elementen, wie Prozessschritte und ihre Bedingungen. Die Ontologie *RGOM* (Yahya et al., 2021) geht über die Montage hinaus und beschreibt den ganzen Produktlebenszyklus, geteilt in verschiedene Ontologien, angepasst an die Anwendung bzw. Domäne. Darunter fällt die Logistik, das Produkt, die Produktion usw. (Liu et al., 2023a) beschreibt den Entwurf einer Ontologie für den Konkreten Anwendungsfall der Produktion von Windturbinen. Sie gliedert sich in drei Ebenen. Begonnen mit Meta Informationen der Montage und Montageabfolge. Gefolgt von der Sequenzbeschreibung der Prozessschritte in der zweiten Ebene. Die dritte Ebene umfasst die Eigenschaften der Bauteile. *AsD Ontologie* (Kim et al., 2006) beschreibt die geometrischen Eigenschaften der Bauteile, ihre Position zueinander und deren Zusammenführung. *MASON Ontologie* (Lemaignan et al., 2006) besteht aus den drei Akteuren Entität, Ressource und Tätigkeit, die sich weiter gliedern. Sie wurde zur Abschätzung der Produktionskosten während der Produkt-Designphase entworfen. *KGRAM Ontologie* (Chen et al., 2020) führt CAD Daten und Montageprozesse zusammen. Sie

gliedert sich in drei Ebenen. Ebene Eins beschreibt den Aufbau der Montagesequenz Struktur. Ebene Zwei gliedert die Beziehung und den Ablauf von Teilschritten der Montageschritte. Und die dritte Ebene verknüpft die Eigenschaften der Montageschritte.
Aus den untersuchten Ontologien ergibt sich ein Konsens zur Gestaltung einer Montage Ontologie. Sie verbindet CAD und Montagesequenzdaten. Die Konzentration liegt in der Modellierung der Beziehung von Bauteilen. Eine hierarchische Staffelung modelliert die Montagesequenzen von groben zu feineren Abläufen. Der Nutzen, wofür die Ontologie eingesetzt wird, führt zu den Unterschieden in den Ontologie-Konzepten. Beispielsweise *RGOM* (Yahya et al., 2021) geht über die Darstellung der Montage hinaus und ergänzt Konzepte, die unter anderem die Logistik mit abbilden. Oder die *MASON Ontologie* (Lemaignan et al., 2006), sie führt Kriterien zur Kostenabschätzung ein. Trotz der Unterschiede erscheint eine Standardisierung der über die Ontologien geschaffenen informellen Konventionen in eine Ontologie, die alle Anwendungsfälle abbildet, sinnvoll. Es erspart die Notwendigkeit, für jeden Anwendungsfall von Neuem eine Ontologie zu entwerfen. Es bietet zumindest ein Fundament, das um spezifische Anliegen erweitert werden könnte. Die Untersuchung, ob das gemeinsam übergreifende Ziel, die Abbildung von Montageprozessen ausreicht, ein Ontologie-Standard zu schaffen, soll kein Gegenstand dieser Arbeit sein? Die Konventionen fließen in den Entwurf der Ontologie für diesen Anwendungsfall ein. CAD Daten sind kein Gegenstand der Montagedaten, entsprechend liegt der Fokus auf der Modellierung der Montageprozesse.

Wie die Visualisierung und Manipulation von Wissensgraphen im großen Stil funktionieren, folgt im nächsten Kapitel 2.2.

2.2 Graphdatenbank

Graphdatenbanken speichern und verwalten Daten in der Form von Knoten, Kanten und Attributen. Diese Struktur ermöglicht effiziente Abfragen über Beziehungen. Technisch werden zwei Konzepte unterschieden, *Labeled-Property Graph* (*LPG*) und *Resource Description Framework* (*RDF*).
LPG. Im Property-Graph-Modell, das von Systemen wie Neo4j (Neo4j, 2023) und Amazon Neptune (im Property-Graph-Modus) (Amazon Web Services, 2023) verwendet wird, bestehen Graphen aus Knoten und Kanten, die Attribute als Key-Value-Paare besitzen können. Dieses Modell bietet Flexibilität durch die Möglichkeit, verschiedene Datentypen und komplexe Beziehungen direkt als Knoten- und Kanteneigenschaften zu speichern. Die Abfragesprachen Cypher und Gremlin sind für Property-Graph-Datenbanken optimiert und ermöglichen die Navigation entlang der Kanten und Filterung anhand der Eigenschaften der Knoten und Kanten (OpenCypher, 2023).
RDF. Das RDF-Modell, ein W3C-Standard (W3C RDF Working Group, 2014, 2024), beschreibt Graphen als Triple (*Subject*, *Predicate* und *Object*) und eignet sich für semantische Graphen. RDF-Datenbanken wie GraphDB von Ontotext (Ontotext, 2023) und Amazon Neptune (im RDF-Modus) (Amazon Web Services, 2023) verwenden SPARQL als Abfragesprache. Die *SPAR-*

QL Abfragesprache, ebenfalls ein W3C-Standard (W3C SPARQL Working Group, 2013), filtert und matched *Triple*, um Verbindungen zwischen *Tripeln* zu finden.

Das *Property-Graph-Modell* bietet durch die Wahl der Sprache und Freiheit in der Gestaltung des Graph-Modells ein an den Anwendungsfall abgestimmtes Modell, das skaliert und effiziente Abfragen zulässt. Es eignet sich für die Echtzeitanalyse. Die fehlende Standardisierung fördert Inkompatibilität. Es erschwert die Integration von Daten aus neuen Datenquellen, sie benötigen eine Anpassung an das System.
Hingegen *RDF* erleichtert durch dessen Standardisierung den Datenaustausch und Integration neuer Daten. Bei komplexen Abfragen bremst das *Triple* Format aus.

Die Gestaltung und Verarbeitung semantischer Graphen sind im Vorgehen ein Vorverarbeitungsschritt. Die Montagedatenmenge ist fest und die Integration neuer Daten erfolgt episodenweise, mit dem Einsatz der Anwendung in neuen Standorten und nicht kontinuierlich. Das *Property-Graph-Modell* bietet mit Skalierbarkeit und effizienten Abfragen wichtige Eigenschaften für die Verwaltung von Millionen von Datenpunkten, steht *RDF* mit der fehlenden Standardisierung in der Erweiterbarkeit nach. Die einmalige Erzeugung von semantischen Graphen entkräftet das Laufzeitargument gegen *RDF*. *RDF* eignet sich aus den genannten Gründen für diesen Anwendungsfall gegenüber dem *Property-Graph-Modell*.

2.3 Graph Matching

Graph Matching identifiziert Ähnlichkeiten bei Graphen, indem Elemente wie Knoten, Kanten und Label zweier Graphen verglichen und auf ihre Gleichheit bewertet werden. Der Anwendungsfall bestimmt den Bewertungsmaßstab der Gleichheit. Mögliche Konzepte sind *Entity Alignment*, *Siamese* und *Subgraph Matching*. Die im Folgenden näher beschrieben werden.

Entity Alignment führt zwei Graphen zusammen, indem ähnliche Knoten eine Paarung bilden (Tam et al., 2021). Das *EAGCN Modell* (Tam et al., 2021) sucht die passenden Entitätspaare zweier Wissensgraphen. Die Wissensgraphen, einer wird Source und der andere Target Graph, teilen einen Embeddingraum und dessen Gewichte. Die effiziente Repräsentation der Wissensgraphen wird durch Graph Convolution Layer erzeugt. Zur Identifizierung der Knotenpaare wird auf das erlernte Knoten Embedding aller Layer zurückgegriffen. Das Training erfolgt Unsupervised, wobei direkte Nachbarknoten zusammen und bezugslose, weit entfernte Knoten auseinandergezogen werden.

Ein **Siamese** Modell (Jiang et al., 2023) vergleicht zwei Sample auf Ähnlichkeiten. Es nutzt ein Deep Embedding Modell zweimal. Das Embedding Modell ist frei wählbar. Die Eingabesample werden separat embedded. Die erzeugten Embedding der Sample werden in einen ge-

meinsamen Embeddingraum überführt und dort auf Ähnlichkeiten verglichen.

Subgraph Matching prüft für einen Graphen, ob er Teil eines anderen Graphen ist. Also, ob sich die Struktur und Eigenschaften in dem Graph wiederfinden (Rex et al., 2020). *FG_{qT}-Match* (Sun et al., 2022) führt Subgraph Isomorphie Matching auf Wissensgraphen aus. Der Query Graph q wird auf den Target Graph G abgebildet. Es wird ein matching-driven flow Graph aufgebaut, aus passenden Kandidaten zwischen den Query und Target Graph. Sie werden schrittweise, anhand ihrer Label und Nachbarstruktur reduziert. Anhand des gehen eines Pfades kann ein Match gefunden werden. Um den Suchraum weiter einzuschränken und das optimale Matching zu finden, gewichtet die multi-label weight Matrix die Knoten nach Relevanz im Subgraph Matching.

Das *neuroMatch Modell* (Rex et al., 2020) prüft ob ein Teilgraph, Query Graph genannt, Teil eines Target Graph ist. Der Target Graph wird dafür in viele Teilgraphen von der Größe des Query Graphen zerlegt. Anschließend wird ein Teilgraph und der Query Graph embedded und dort verglichen. Das Embedding Modell ist frei wählbar, mit der Einschränkung, dass es die Reihenfolge der Knoten berücksichtigt.
GLSearch (Bai et al., 2021) sucht den größten gemeinsamen Subgraphen zweier Graphen. Voraussetzung, der Subgraph muss zusammenhängend sein und kann Knotenlabel haben, die dann ebenfalls übereinstimmen müssen. Nach der Branch and Bound Algorithmen-Klasse wird ein Suchbaum aufgebaut. Anstelle die Knotenpaare mit Heuristik zu bestimmen, ersetzt er diesen durch ein Deep Q-Network. Der Agent lernt eine bessere Suchstrategie.
Alle zuvor vorgestellten Ansätze betrachten ein oder mehrere Graphen, fester Struktur.
(Sub-)Graph Matching auf sich ständig verändernden Graphen auf struktureller oder semantischer Weise kommt mit der Herausforderung: Zeitnah Vergleiche bzw. Muster von Interesse zu liefern, die nicht obsolet sind (Wang et al., 2022). Der *NNT* Algorithmus (Wang et al., 2022) berücksichtigt strukturelle Änderungen des Graphen. Das Matching findet auf einem statischen Stand des Graphen statt. Ausgehend von Knoten des Graphen werden für sie Bäume konstruiert. Die Bäume speichern die Nachbarschaftsbeziehungen des Wurzelknotens bis zu der vorher angegebenen maximalen Tiefe. Beim Subgraph Matching wird der Baum eines Zielgraphen mit dem Baum des Query Graph verglichen. Bei Updates im Graphen (Einfügen oder Löschen von Kanten) passt der Algorithmus den Baum an, indem betroffene Unterstrukturen über invertierte Indizes lokalisiert und aktualisiert werden. Der *Gradin* Algorithmus (Wang et al., 2022) behandelt semantische Änderungen des Graphen. Die Graph-Eigenschaften werden in einem mehrdimensionalen Gitterindex gespeichert. Änderungen von Attributen oder Labels können effizient durchgeführt werden. Bei Änderungen in Kanten oder Knotenattributen wird die betroffene Region im Gitter aktualisiert, wodurch nur relevante Teile des Graphen neu berechnet werden müssen. Beim Subgraph Matching wird der entsprechende Query Graph in den Gitterindex überführt. Nur die gezielten Regionen um den Query Graph werden auf Übereinstimmung in dem Gitterindex geprüft.

Die Montagedaten des Anwendungsfalls sind statisch. Algorithmen, die Eigenschaften dynamischer Graphen berücksichtigen, werden ausgeschlossen. Ihre zusätzliche Funktionalität steigert dessen Komplexität, bietet in diesem Fall aber keinen Vorteil. Weitere Subgraph Matching Ansätze wie *FG_{qT}-Match* (Sun et al., 2022) eignen sich zur Erkennung von Ähnlichkeiten. Die Beschränkung auf einen Teilvergleich lässt einen Großteil des Graphen jedoch außen vor. Besonders *GLSearch* konzentriert sich auf das Finden des größten gemeinsamen Subgraphen. Die Ansätze sind zu sehr konzentriert auf die Graphenstruktur und auf das Finden von exakter Gleichheit. *neuroMatch* (Rex et al., 2020), *Entity Alignment* und *Siamese* hingegen sind von den festen Bedingungen gelöst. Sie ersetzen exakte Gleichheit durch Finden von Ähnlichkeit, anhand eines Schwellenwertes. Die Sichtweise ermöglicht, eine semantische Verbindung zwischen Knoten zu schaffen, die zuvor als ungleich eingestuft wurden. Das *neuroMatch Modell* (Rex et al., 2020) und *Siamese Modell* bieten weitere Freiheit in der Wahl des Graph Embedding Modell. *Entity Alignment* hingegen nicht und fällt deshalb aus der Betrachtung. Der Grundlegende Unterschied zwischen dem *neuroMatch Modell* (Rex et al., 2020) und *Siamese* ist die Aussagekraft. *Siamese* beziehen die gesamten Graphen ein, während *neuroMatch* (Rex et al., 2020) Aussagen für Subgraphen trifft. Die Beschränkung auf Subgraphen wird in diesem Kontext als Nachteil eingestuft, der mögliche semantische Ähnlichkeiten übersehen könnte. Entsprechend fällt die Wahl auf *Siamese*. Sie ist eine Hülle für den Vergleich von Embeddings. Es bietet den ganzen Graphen den Freiraum, die erlernte Semantik zu vergleichen, ohne eigene Anforderungen zu stellen. Das *Siamese* Modell benötigt ein Graph Embedding Modell, welche Architekturen sich eignen, folgt im nächsten Kapitel 2.4.

2.4 Graph Embedding

Embedding umfasst ein Deep Learning Modell, welches eine effiziente Repräsentation der Sample erlernt. Das Modell reduziert die Sample auf dessen wesentlichen Eigenschaften und gibt sie komprimiert zurück. Anwendungsfälle sind die Übersetzung von Texten in eine andere Sprache, die Klassifikation von Sampeln oder der Vergleich von Sampeln. Graph Embedding beschreibt Embedding für Deep Graph Modelle, die Graphen effizient repräsentieren. Vertretende Deep Graph Modell Konzepte sind Convolution, Auto-Encoder und Translational.

Die verbreitetste Funktionsweise der Deep Graph Modelle, zum Erlernen eines Embedding, ist das *Message Passing*, es beschreibt den Nachrichtenaustausch von benachbarten Knoten des Graphen. Ein Knoten erhält sein Embedding, indem es Informationen der Nachbarknoten mit einbezieht. *Aggregation* ist der Schritt der Kombination der erhaltenen Nachbarinformationen. Ein *Aggregator* bezieht die Informationen direkter Nachbarknoten für das Embedding eines Knoten ein. *Hopp* gibt die Pfadtiefe an, wie viele *Aggregatoren* hintereinander geschaltet, das *Message Passing* für das Embedding eines Knoten mit einbezieht (Hamilton et al., 2018). Grafik Zwei der Abbildung 3 illustriert den Vorgang beispielhaft. Der rote Knoten erhält sein Embedding durch zwei *Hopps*, aggregator_1 und aggregator_2. Aggregator_1 zieht die Informa-

tionen der direkten Nachbarknoten (blau markiert) in das Embedding des roten Knoten mit ein. Aggregator_2 die entsprechend grün markierten Nachbarknoten der blauen Knoten.

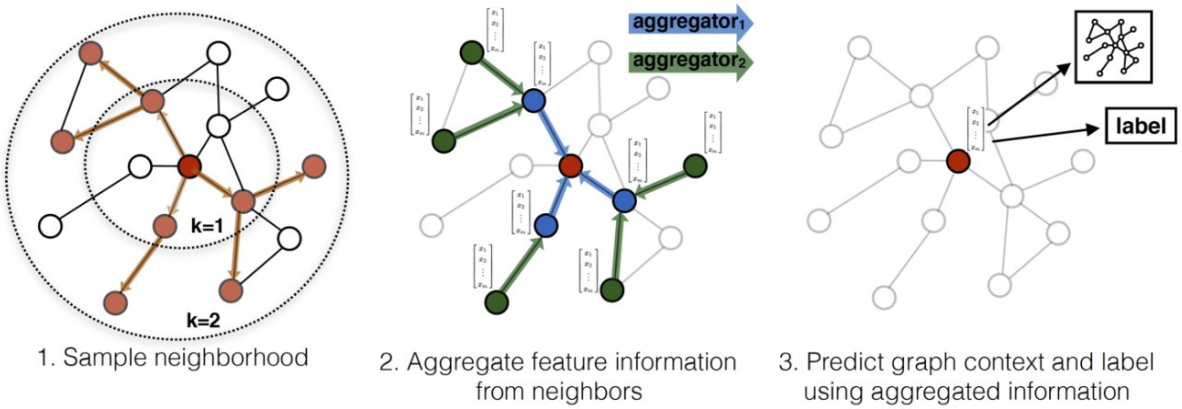


Abb. 3: GraphSAGE: Eine Methode zur induktiven Repräsentationsgenerierung (Entnommen aus (Hamilton et al., 2018)). Zeigt das Sampling und die Aggregation des GraphSAGE Modells. Grafik Eins zeigt das Sampling eines Graphen der Tiefe Zwei. Gesampelt werden die orange markierten Nachbarknoten, ausgehend vom rot markierten Knoten. Die mittlere Grafik demonstriert die Aggregation. Der rote Knoten erhält seine Repräsentation durch Nachbarinformationen der Tiefe Zwei. Aggregator_1 bezieht die Informationen der direkten Nachbarn des roten Knoten mit ein und Aggregator_2 die Informationen der übernächsten Nachbarn. Jeder Aggregator ist ein weiterer *Hopp*. Grafik Drei beschreibt mögliche Anwendungen für das Knoten Embedding, die Vorhersage des Knotenlabel oder des Graphkontext.

Convolution Embedding. *GraphSAGE Modell* (Hamilton et al., 2018) hat drei mögliche Aggregationsfunktionen, *mean*, *pool* und *lstm*. *Hopps* sind auf die Tiefe Eins begrenzt und damit ist die Tiefe der betrachteten Nachbarknoten an die Anzahl der Layer gebunden. Das Training verringert die Distanz zwischen Nachbarknoten im Embeddingraum und vergrößert sie zwischen weit entfernten Knoten. Statt eines direkten Knoten Embedding erlernen die Aggregationsfunktionen das Aggregieren der Feature der Nachbarknoten. *CompGCN Modell* (Vashishth et al., 2020) embedded Knoten und Kanten in einem Embeddingraum. Die Graphen sind gerichtet, wobei invertierte Kanten eingefügt werden. Kanten werden abhängig von dessen Typ gewichtet. Es werden „originale“, invertierte und self-loop Kantentypen unterschieden.

Translational Embedding. Anders als die zuvor angesprochenen Konzepte, embedden *TransE Modelle* (Yan et al., 2022) keine Graphen, sondern Triple. Entitäten und die Relation werden als Vektoren in einem Embeddingraum abgebildet. Statt des *Message Passing* beschreibt eine *Scoring Funktion* die Beziehung der Entitäten und deren Relation. Gelernt wird Unsupervised, die Distanz positiver Sample wird minimiert und die von negativen Samplen erhöht. Negative Sample werden durch das zufällige Ersetzen einer Entität des Triples gebildet. Das *CapsE Modell* (Nguyen et al., 2019) ist eine hybride Weiterentwicklung. Vorab wird der Eingabetri-

ple in eine Matrix überführt. Dann werden die Features durch Convolution Layers extrahiert und erzeugen sogenannte „Feature Maps“, diese werden anschließend in „Kapseln“ zusammengefasst. Die „Kapseln“ werden zu einem Vektor zusammengefasst und der Ausgabekapsel übergeben, diese erzeugt durch eine nicht lineare Aktivierungsfunktion eine Vektorausgabe. Die Vektorlänge stellt die Plausibilität des Triples dar.

Auto-Encoder Embedding. Das *HGATE Modell* (Wang et al., 2023b) lernt die Rekonstruktion der Knoten Features und Kanten von heterogenen Graphen. Die Aggregation der direkten Nachbarknoten Feature erfolgt über *Meta-Paths*. Sie verwendet Attention, die die Nachbarknoten Feature gewichtet. Anschließend wird die Relevanz zwischen den *Meta-Path* bestimmt. Der Decoder invertiert den Prozess. Erstens werden die Knoten Feature durch die Nachbarknoten über die Meta-Path rekonstruiert. Zweitens, die Knoten Feature durch die *Meta-Path* Informationen ergänzen. Das *SAGES Modell* (Wang et al., 2023a) setzt auf Wissens gestütztes Subgraph Sampling. Aus einem ungerichteten Graph werden Subgraphen gesampelt, der Prozess berücksichtigt die Knoten Feature und die Graphstruktur, dies erhöht die Modellqualität. Das Encoding Modell ist konzeptionell frei wählbar. Der Decoder rekonstruiert die Knoten Feature und Graphen Struktur. Das ergänzende Training um negative Sample bewirkt eine stärkere Zusammenführung des Embedding ähnlicher Knotentypen. *SelfLinkG Modell* (Liu et al., 2023b) schafft eine gemeinsame Repräsentation zweier Wissensgraphen. Statt eines Encoder-Decoders wird ein Encoder trainiert. Knoten Feature werden durch ein pre-trained LLM embedded. Nachbarknoten Feature werden durch Attention Layer gewichtet in das Knoten Feature Embedding mit einbezogen. Darüber hinaus werden Oberbegriffe aus synonymen Nachbarn gebildet und embedded, um die begriffliche Zugehörigkeit Wissensgraphen übergreifend zu stärken. Zur Trainingsstabilisierung werden die Gewichte eines zweiten identischen Encoders, durch einen Momentum Faktor, gleichmäßiger angepasst.

Ein geeignetes Embedding Modell soll in erster Linie ein Embedding liefern, das eine Graph-Ähnlichkeiten-Bestimmung erleichtert, es erlernt eine kompakte Repräsentation, die den Detailgrad der Features erhält. Dafür muss das Modell die semantische Schärfe der Montagedaten berücksichtigen, wie Knoten-Feature, Knoten- und Kantentypen und gerichtete Kanten für das *Message Passing*. Fehlende Klassifizierungslabel in den Montagedaten fordern eine Unsupervised Trainings-Ausrichtung des Embedding Modells. Grundsätzlich zählt die Fehlerquellenminimierung, Modellanpassungen an den Anwendungsfall fördern diese wiederum.

Drei geeignete Modelle sind *CapsE*, *HGATE* und *GraphSAGE*. Jedes dieser Modelle basiert auf einem anderen Konzept, ein Vergleich der Konzepte für den Anwendungsfall bietet sich an. *CapsE* trainiert Unsupervised und bildet die semantische Schärfe der Montagedaten durch Triple ab. Da Sample im Triple Format vorliegen, ermöglicht dem Modell eine nahtlose Nutzung. Die Betrachtung eines Triple zur Zeit ist eine starke Einschränkung, die eine Anpassung des Modells für den Anwendungsfall bedarf. Triple Paare bilden und vergleichen setzt ein manuelles Mapping der Paare voraus. Alternativ das Ermöglichen des Embedding aller Triple

in einem Embeddingraum durch Ergänzen der Modellarchitektur, um beispielsweise Attention Layer, die eine Beziehung der Triple herstellen. Die notwendige Anpassung ist von hohem Einsatz und birgt ein hohes Risiko der Abschätzung, ob sich die Anpassung eignet. *HGATE* trainiert Unsupervised die Rekonstruktion der Knoten Feature und Kanten. Die Encoder-Ausgabe bietet ein erlerntes Embedding für die Graphen Ähnlichkeiten Erkennung. Und das Sample Format sind gerichtet Graphen. Das Modell bedarf keiner Anpassungen und erfüllt damit alle Voraussetzungen. *GraphSAGE* erlernt Unsupervised ein Knoten Embedding, auf ungerichteten Graphen. Eine Erweiterung des *GraphSAGE* Modells, um die Richtung der Kanten zu berücksichtigen, ist zu befürworten. Es würde die Semantik der Graphen genauer abbilden. Zusammengefasst bedarf *CapsE* von allen drei Modellen als einziges eine als notwendig eingestufte Anpassung, *HGATE* berücksichtigt am besten die Semantik. *GraphSAGE* hingegen sticht durch dessen Implementierung in bekannten Deep Learning Bibliotheken heraus. Die Implementierung und damit das minimierte Risiko, eigene Fehler einzubauen, sprachen für die Wahl des *GraphSAGE* Modells über die semantisch schärfere Repräsentation des *HGATE* Modells.

3 Umsetzung

Die Montagedaten und das Ziel formten maßgeblich die konzeptionelle Umsetzung. Das Konzept (Abbildung 4) beschreibt die Transformation der Montagedaten, sodass sie anschließend zielgerichtet analysiert und Informationen über die Fragestellung offenbaren.
Die Montagedaten werden im ersten Schritt in Wissensgraphen transformiert, dies geschieht nach einer vorher auf Basis der Montagedaten erzeugten Ontologie. Die Ontologie beschreibt auf abstrakte Weise die Beziehungen und Features der Montagedaten. Die Wissensgraphen liegen als textuelle Dateien im RDF-Format vor. Im nächsten Schritt werden die Daten an das Graph Modell angepasst, ein mit den Montagedaten trainiertes Word2Vector Modell embedded die Features in Vektoren. Das Word2Vector Modell erlernte eine Vektorrepräsentation der Wörter. Anschließend lernt das Graph Modell eine gemeinsame Repräsentation der Wissensgraphen, bevor es dazu verwendet wird, die Ähnlichkeiten zweier Wissensgraphen vorherzusagen.
Die Abbildung 4 illustriert den genannten Ablauf, sie folgt der Datenverarbeitungsrichtung. Gelesen von links oben nach rechts unten. Die Vorstellung der Montagedaten erfolgt in Kapitel 3.1. Die Datenkonvertierung, Kapitel 3.2, erzeugt den Datensatz, sie benötigt dafür die Ontologie. Der Entwurf der Ontologie, beschrieben in Kapitel 3.3, richtet sich nach den Charakteristiken der Montagedaten. Nicht in der Illustration aufgeführt ist die Graphdatenbank. Dessen angedachte Rolle in der Datensatzerzeugung beschreibt Kapitel 3.4. Das Datensatz Kapitel 3.6 beschreibt die Überführung des Datensatzes in ein für das Graph Modell passende Format. Die technische Umsetzung des Graph Modell beschreibt Kapitel 3.5. Graph Matching, Kapitel 3.7, embedded zwei Sample *x_1* und *x_2* separat mit demselben Graph Modell. Dessen Embeddings werden in einem Embeddingraum auf Ähnlichkeiten verglichen und anschließend wird ein Ähnlichkeiten-Score ausgegeben.

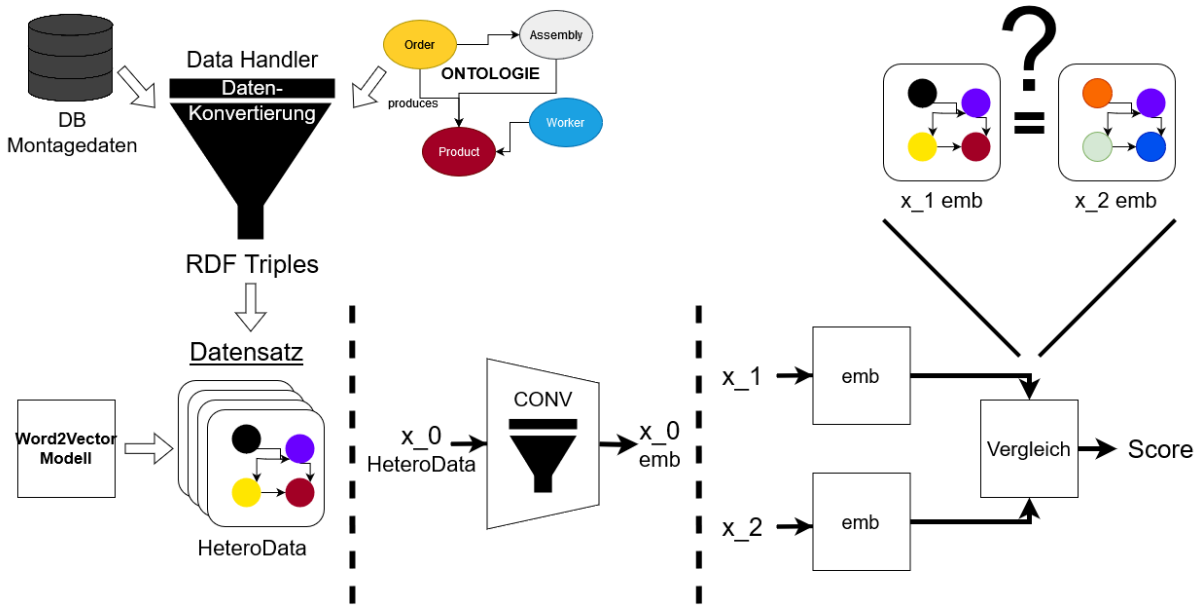


Abb. 4: Konzept der Umsetzung. Betrachtung beginnend von oben links nach rechts unten. Der *Data Handler* konvertiert die Montagedaten in Wissensgraphen und verwendet die Konzepte der Ontologie. Die Wissensgraphen liegen im *RDF Triples* Format vor. Die Wissensgraphen werden in ein für das Graph Modell passendes Format gebracht und die Features der Wissensgraphen werden mit dem Word2Vector Modell embedded. Das Graph Modell lernt eine Repräsentation der Wissensgraphen. Abschließend werden die Embeddings zweier Wissensgraphen auf Ähnlichkeit verglichen.

3.1 Montagedaten

Die Prozessdaten entstammen aus der Montage von Hydraulikaggregaten, die durch ein Werkerassistenzsystem aufgenommen wurden. Die Prozessdaten werden in Folge als Montagedaten bezeichnet. Ein beispielhaft montiertes Produkt ist eine Kraftstoffpumpe (siehe Abbildung 5). Bilder der Montage von Hydraulikaggregaten stehen aus rechtlichen Gründen nicht zur Verfügung und dürfen nicht veröffentlicht werden. Exemplarisch als Anschauungsobjekt wird die Kraftstoffpumpe verwendet. Die gesamte Montage erfolgt an einer oder verteilt auf mehrere Arbeitsstationen. Die beteiligten Arbeitsstationen werden sequenziell geschaltet und an jeder Arbeitsstation werden mehrere Montageschritte durchgeführt. Die Kraftstoffpumpe wird auf deren Montageplatte über ein Montageband zur nächsten Arbeitsstation befördert. Abbildung 6 zeigt den Beginn der Montage der Kraftstoffpumpe an einer Arbeitsstation. Das Pumpenuntergehäuse liegt auf der Montageplatte (1). Der Akkuschrauber und die Winkelschraubendreher deuten das für die Montage notwendige Werkzeug an (2). Im industriellen Fall sind stationäre Schrauber mit eingestellter Fassung und Drehmoment vorhanden. Die Ablagen führen die für die Montage benötigten Bauteile (3). Der Monitor unterstützt die Montage mit dem Zeigen der Montageanleitung (4).

Die wesentlichen 20 Montageschritte der Kraftstoffpumpe sind in einem vereinfachten Ablaufdiagramm 21 beschrieben (Anhang A.4). In der ursprünglichen Montageanleitung illustrieren beigefügte Bilder den beschriebenen Ablauf der Schritte. Sie kann aus rechtlichen Gründen nicht gezeigt werden. Die Montageschritte 15 bis 19 sind in der Bildfolge 22 (Im Anhang A.5) demonstriert, die in dem Montageschritt beteiligten Bauteile sind rot hervorgehoben. Sie zeigen die Montage des Filters. Der vollendete Filter ist in Abbildung 5 rot hervorgehoben. In Schritt 15 wurde der Filterdeckel aus dem Fach entnommen und aufgelegt. In Schritt 16 werden vier M5x10 Schrauben angesetzt und anschließend im Folgeschritt der Filterdeckel verschraubt. In Schritt 18 und 19 wird die Ablassschraube angesetzt und verschraubt.

Die Bandbreite der Montagedaten, bestehend aus 25 Einträgen, sie umfasst die beteiligten Personen, die Aufgaben und die Messwerte. Tabelle 7[2] im Anhang A.3, beschreibt alle 25 Einträge. Die relevanten Folgen im Text. „ID“ wird in Folge nicht gebraucht und ignoriert, es zählt auch nicht als Eigenschaft der Aufträge, sondern ist lediglich eine Aufzählung aller Zeilen. So kommen auch die zuvor genannten 25 Einträge zustande (ohne „ID“ gezählt).
Die Abgrenzung der Montagedaten erfolgt per Auftrag bzw. Seriennummer des jeweiligen Produkts. Die Betrachtung erfolgt reihenweise, die Zugehörigkeit der Messdaten zu einem Auftrag erfolgt über die Auftragsnummer „MA_NR“. 1113 Aufträge umfassen die $15,6$ Millionen (genau $15.642.083$) Montagedaten Zeilen. Der Zeitstempel, der den Zeitpunkt einer Messung dokumentiert, gibt Aufschluss über die Abfolge der ausgeführten Aufgaben einer Montage. „STATUS“ und„STATUSTEXT“ zeichnen simultan, „STATUS“ als ID und „STATUSTEXT“ als

[2]Die Spalte „Default“ führt einen Repräsentanten jeder Spalten auf.

Beschreibung, die Bewertung der Durchführung einer Aufgabe aus. Misserfolg („STATUS“=1 und „STATUSTEXT“=NIO, Nicht in Ordnung) kann zu einer Wiederholung der Aufgabe führen. Es lässt jedoch keine Aussage darüber zu, ob ein Produkt fehlerfrei gebaut wurde, es bewertet lediglich die Durchführung eines Prozessschrittes. Deshalb kann der gleiche Prozessschritt mehrfach in der Montage eines Produkts vorkommen.

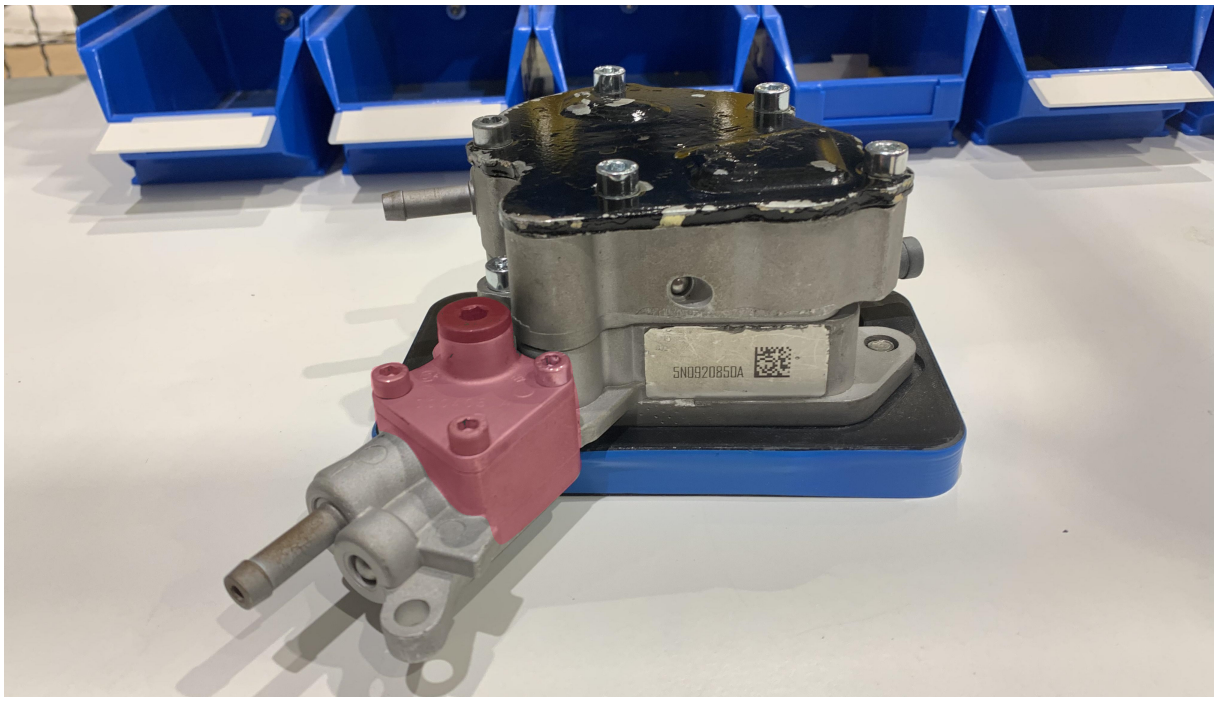


Abb. 5: Eine montierte Kraftstoffpumpe auf der Montageplatte. Der Filter ist in Rot hervorgehoben.

Die Montagedaten kommen ohne Label oder Fehlerklassen, was ein unmittelbares Supervised Training erschwert. Die Ermittlung eigener Label oder Fehlerklassen benötigt Zeit und ein tiefes Domäne-Verständnis in der Produktion, diese Faktoren waren schwer abschätzbar und führten zu der Entscheidung, die Montagedaten Ähnlichkeit durch Unsupervised Training zu ermitteln. Die Datenmenge ist fest und die Beschaffung von weiteren Montagedaten ist erschwert. Zur Erhebung neuer Montagedaten müssten weitere Produkte gefertigt oder vorab neue Produkte und deren Montage entworfen werden, vor der eigentlichen Fertigung. Dies stellt einen enormen Aufwand dar. Alternativ gäbe es die Möglichkeit, aus dem bestehenden Datenbestand synthetische Aufträge zu sampeln und damit vorbeugend die Datenvielfalt zu erhöhen, durch Out-Of-Distribution (kurz OOD) (Li et al., 2022). Doch dieser Ansatz schien für den Anwendungsfall unpassend, sie gefährdet die Datenintegrität und es könnte das Erlernen falscher Beziehungen auf falschen Annahmen fördern.
Die begrenzte Menge an Aufträgen gepaart mit eingeschränkter Diversität und fehlenden Werten führt zu einer Reihe von Risiken, die sich offenbaren. Es besteht ein erhöhtes Risiko des Overfitting, durch die starke (strukturelle und inhaltliche) Ähnlichkeit (begünstigt durch die Domäne/Anwendung) der Montagedaten. Gleichfalls läuft das Modell Gefahr, dem Shortcut Learning zu verfallen, durch fehlende Werte, die im Unsupervised Training dem Modell den

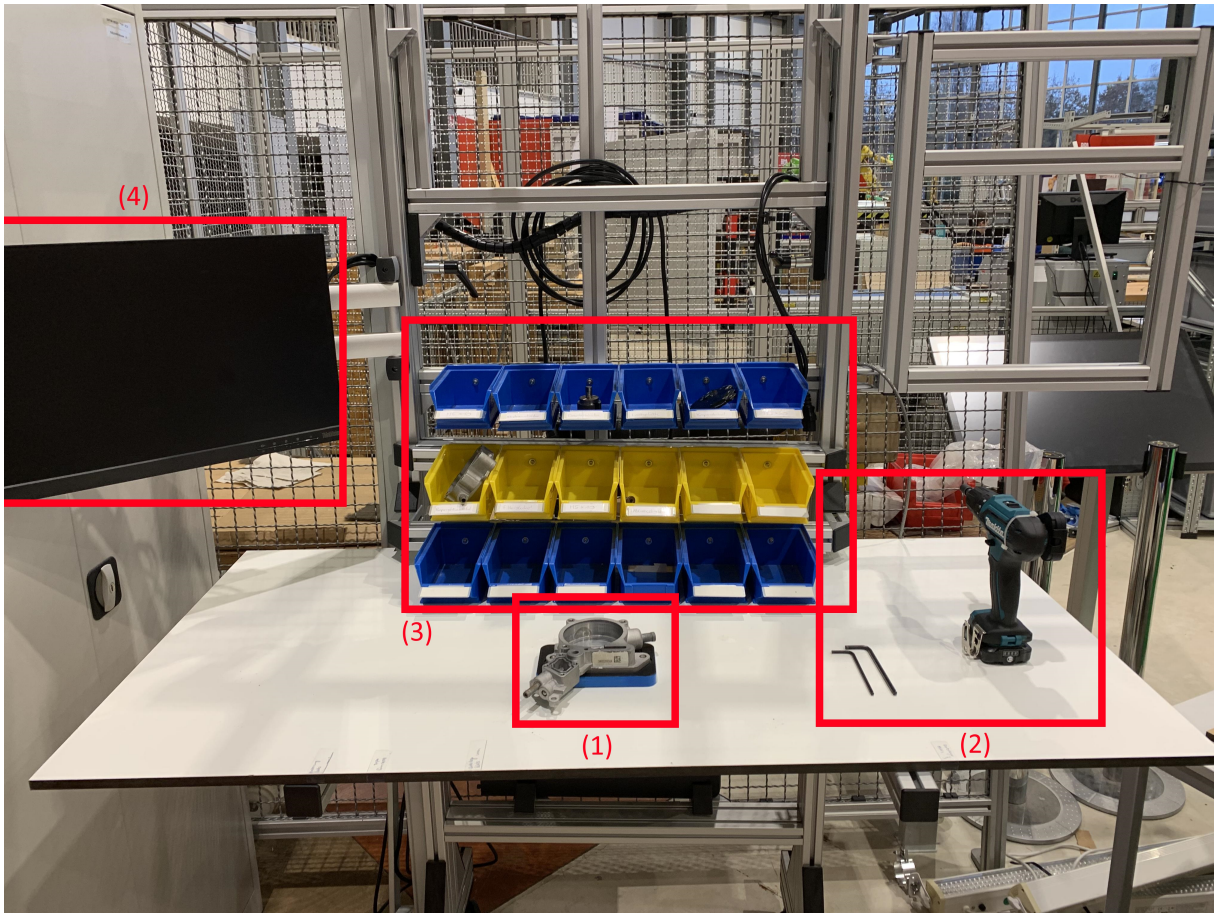


Abb. 6: Arbeitstation: Zeigt den Beginn der Montage einer Kraftstoffpumpe. (1) zeigt das Pumpenuntergehäuse auf einer Montageplatte. (2) Akkuschrauber und Winkelschraubendreher stehen demonstrativ für die in der Montage benötigte Werkzeuge. (3) Ablagen führen für die Montage benötigte Bauteile. (4) zeigt die Montageanleitung.

Eindruck von Gemeinsamkeit erwecken. So fallen verständlicherweise die an den Prozessschritten beteiligten *WERKER* Namen dem Datenschutz zum Opfer und schwächen damit zugleich die Datenqualität, in Bezug auf die Ähnlichkeitserkennung. Es schwächt die Unterscheidbarkeit, da unterschiedliche Daten einen gemeinsamen Bezugspunkt bekommen. Alle Aufgaben werden damit von demselben *WERKER* ausgeführt. Graph Modelle lernen Knoten Embedding häufig über Informationen ihrer Nachbarn und bei selben Nachbarn gleichen sich die Embedding unterschiedlicher Knoten weiter an und schwächen die Unterscheidungskraft des Embedding. Eine ausgleichende Möglichkeit ist das eigene Füllen dieser leeren Werte, dies bürgt jedoch das Risiko, dass getroffene Annahmen falsche Beziehungen schaffen, die nicht korrekt sind. Begründet aus dem Anspruch heraus, mit echten Daten zu arbeiten, wurde die Ergänzung fallen gelassen. Das Beispiel verdeutlicht, dass die Montagedaten direkt aus einer Produktion kommen und nicht auf Deep Learning Prozesse ausgelegt sind.
Die eingeschränkte Diversität kann auch als Stärke verstanden werden, trotz dessen bieten die Montagedaten genügend Variation für den Anwendungsfall. Overfitting kann in dem begrenzten Anwendungsfall gewollt bzw. nicht störend sein, der Korpus an Möglichkeiten ist statisch und die Prozesse unterscheiden sich in Details wie beispielsweise die Schraubengröße und dessen Reihenfolge im Montageprozess. Drastische Unterschiede in der Montage von Produkten muss die physische Produktion widerspiegeln und dies möchte man wie bereits in der Motivation erwähnt vermeiden.
Die Daten stammen aus der Produktionsdomäne und die damit erzeugte Anwendung findet dort Ihre Anwendung. Die Laborbedingungen sind ähnlich den Anwendungsbedingungen, und ein Einsatz kann standortübergreifend ohne gravierende Anpassungen geschehen. Die Montagedaten werden bereits umfangreich erhoben. Eine industrielle Standardisierung würde nicht nur die Datenmenge, sondern auch die direkte Anwendbarkeit erhöhen. Eine ergänzende Protokollierung entstandener Fehler, fehlerhafter Prozesse und getesteter Varianten würde das Labeln auch mit geringem Domäne Wissen und Training beschleunigen.

Das beendet den Überblick über die Montagedaten. Sie in kleinere, verständlichere und semantisch reichhaltigere vergleichbare sample zu konvertieren, darum soll es im folgenden gehen. Kapitel 3.2 widmet sich der konkreten Umsetzung dieser Konvertierung.

3.2 Datenkonvertierung

Die Datenkonvertierung erzeugt für jeden Auftrag einen Wissensgraph, indem es die Montagedaten in die Struktur der Ontologie überführt.
Für eine einfachere Handhabung der Montagedaten in der Datenkonvertierung sind sie in einer relationalen Datenbank abgelegt. Die Montagedaten liegen dabei unverändert in einer Tabelle vor. Die Spalten entsprechen den aufgezeichneten bzw. notierten Werten und eine Zeile kann über dessen *MA_NR*, dem Auftrag zugewiesen werden.

Das Ziel, der Vergleich, veranlasst eine Verarbeitung der Montagedaten. Ein Graph Modell benötigte Sample derselben Struktur zum Erlernen eines Embedding. Eine direkte Datenkonvertierung der Montagedaten in eine weitestgehend ähnliche Ontologie Struktur ist eine Möglichkeit und spielt die Stärke der relationalen Datenbanken, das effiziente Verarbeiten großer Datenmengen aus. Der Grund der fehlenden Modularität sprach dagegen. Jedoch hätten einzelne Vorverarbeitungsschritte auf dieser Ebene durchzuführen, die Überführung der Montagedaten in Wissensgraphen beschleunigen können. Auf der anderen Seite ist die Datensatz-Größe begrenzt und es handelt sich um einen einmaligen Vorverarbeitungsschritt, der auf einen zweiten Rechner ausgelagert werden kann, daher ist Zeit kein hartes Kriterium für die Datenkonvertierung.
Die in Folge beschriebene Anwendung führt die Datenkonvertierung durch.

3.2.1 Data Handler

Die entwickelte Datenkonvertierungsanwendung heißt *Data Handler*. Die Bedienung der Anwendung erfolgt über das Terminal, für weitere Details siehe die Dokumentation vom Data Handler. Die Funktionalität umfasst das Abfragen von PostgreSQL Queries und die Möglichkeit anschließend die Rückgabe der zuletzt ausgeführte PostgreSQL Query als pickle Datei zu exportieren. Die Rückgabe liegt als Pandas DataFrame vor und kann entsprechend von einer beliebigen Python Anwendung importiert werden.
Die Hauptfunktionalität ist die Konvertierung von Montagedaten in Wissensgraphen, mit dem anschließenden Export in RDF Dateien. Im ersten Schritt werden alle Entitäten einmalig vorab erzeugt, die sich die Aufträge teilen, darunter fallen die *Worker*, *Workstations* und *Tools* (Farblich hervorgehoben in der Ontologie 8). Der zweite Schritt erzeugt für jeden Auftrag deren spezifischen Entitäten und formt den Wissensgraph nach der Ontologie, indem es die Beziehungen zwischen den Entitäten setzt. Wie viele Wissensgraphen in einer RDF Datei gespeichert werden wird beim Export Start als Argument übergeben.
Version eins des Data Handler erfüllte alle funktionalen Anforderungen. Jedoch führte die ineffiziente Bewältigung der Aufgabe zu einem hohen Hauptspeicheraufkommen, was die Kapazität der verwendeten Hardware überstieg und eine Auslagerung der temporären Daten auf die Festplatte ließ die Laufzeit ins Unermessliche steigen. Version zwei löst das Problem, in-

dem es die Schleifen durch effizientere Verarbeitung der Montagedaten ersetzt.

Das Komponentendiagramm (Abbildung 7) zeigt die an der Konvertierung beteiligten Komponenten. Der *User* interagiert mit dem *Terminal*. Die *Terminal Komponente* nimmt Eingaben entgegen und gibt Statusinformationen zum Konvertierungsfortschritt als Log-Nachrichten zurück. Die *Logic Komponente* ist das Zentrum, sie verwaltet die Ein- und Ausgaben, und stellt die Verbindung zu der Datenbank der Montagedaten über die *DB Connector Komponente* her. Die *DB Connector Komponente* verwaltet die Datenbank Verbindung. Query bzw. Datenbank Abfragen und Responses erfolgen über die *DAO Komponente*. Die *Logic* und die *RDF Handler Komponente* machen Gebrauch von der *DAO Komponente*. *Logic* Fragt die Auftragsnummern (*MA_NR*) ab und übergibt sie dem *RDF Handler* zur weiteren Verarbeitung. Die *RDF Handler Komponente* fragt die Montagezeilen der Auftragsnummern(*MA_NR*) ab und konvertiert die Daten wie im vorherigen Paragraph beschrieben.

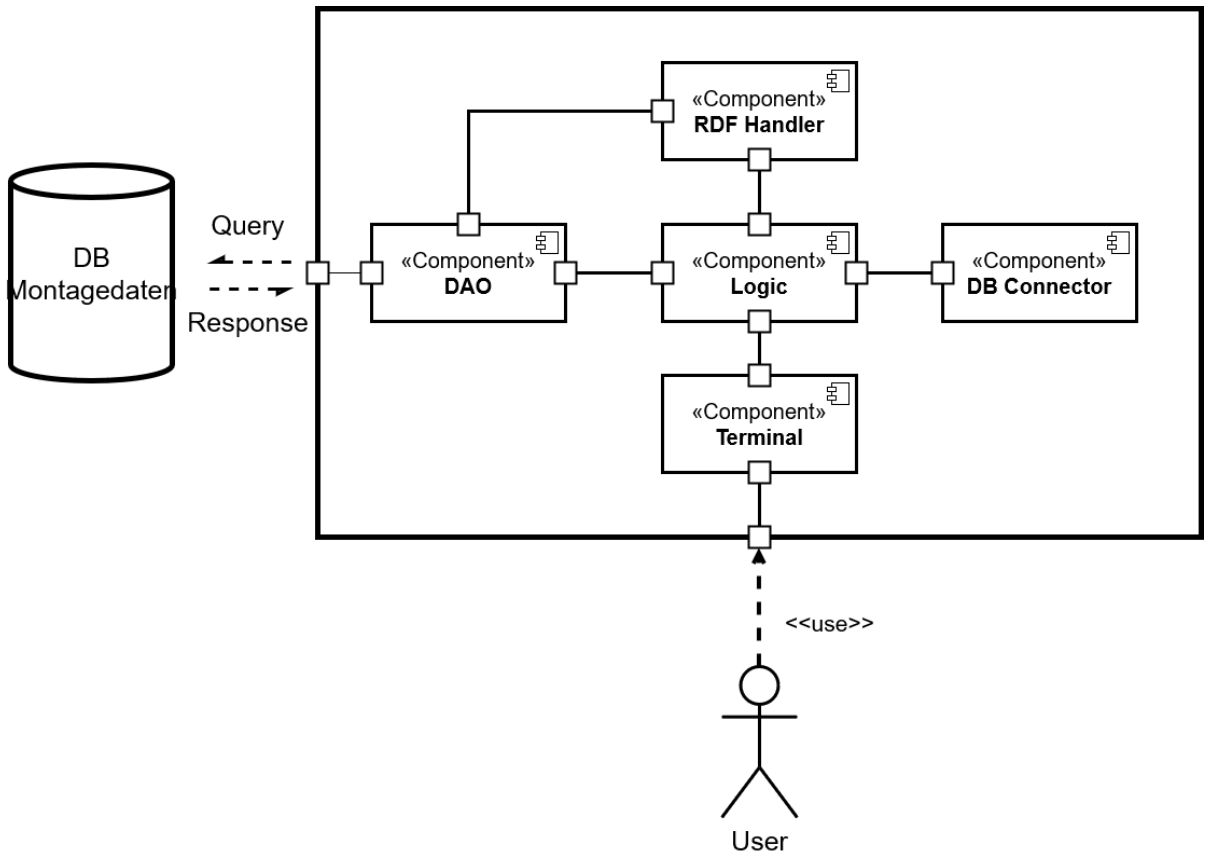


Abb. 7: Komponentendiagramm des Data Handler. Zeigt die Komponenten und deren Beziehungen für die Konvertierung der Montagedaten in Wissensgraphen. Der *User* interagiert mit dem Terminal. Das Terminal nimmt Eingaben entgegen und stellt Ausgaben dar. Die Logic verwaltet Ein- und Ausgaben und stellt die Datenbankverbindung über die DB Connector her. Der DB Connector verwaltet die Datenbankverbindung. Die Queries und Responses erfolgen über die DAO-Komponente. Die Logic und RDF Handler Komponenten stellen Queries an die Datenbank. Die RDF Handler Komponente führt die Konvertierung der Montagedaten in Wissensgraphen durch.

Das folgende Kapitel 3.3 behandelt die semantische Struktur, in die der Data Handler die Montagedaten konvertiert. Dabei werden sowohl die entwickelte Ontologie als auch der Entwurfsprozess beschrieben.

3.3 Ontologie Entwurf

Die Ontologie stellt eine zentrale Rolle des Vorhabens der Ähnlichkeiten Erkennung dar. Sie ist das konzeptionelle Abbild der Semantik der Montagedaten. Die Ontologie folgt drei Grundsätzen: Einfachheit, Domäne und Daten. Einfachheit, die Ontologie ist auf das Wesentliche reduziert, damit es leicht verständlich und erweiterbar ist. Domäne, sie folgt dem strukturellen Konsens der Domäne, eine Drei-Ebenen-Hierarchie. Daten, die Ontologie spiegelt die Montagedaten wider, um ihre Semantik hervorzuheben.
Ausgehend von der Literaturrecherche, dessen Ziel neben dem Verständnis das Finden einer passenden Ontologie für den Anwendungsfall war, führte zu dem Entwurf der Grundsätze und Gestaltung einer eigenen Ontologie (Ergebnisse der Recherche im Stand der Forschung Kapitel 2.1). Die Bereitstellung der Ontologien war sporadisch bis gar nicht vorhanden und meist fehlerbehaftet. Darüber hinaus sind die Ontologien vorzugsweise an CAD (Computer-Aided Design) und BOM (Bill of Material / Stückliste) Daten ausgerichtet, wohin gehend die Montagedaten auf Aufträge und Prozesse ausgelegt sind.
Der Kern der entworfenen Ontologie namens „buPAOv1“
(BIBA-Uni-Prueger-Assembly-Ontology-Version-1) ist die *Order*. Neben dem Abbilden der Semantik der Montagedaten muss die Ontologie auch die Funktion der Vergleichbarkeit erfüllen, da sie als Sample für das Deep Learning Modell Verwendung finden. Die Ontologie in seiner Gesamtheit befindet sich im Anhang A als letzte Seite. In diesem Kapitel ist eine reduzierte Version dargestellt (siehe Abbildung 8), sie zeigt die Entitäten und dessen gerichtete Beziehungen, die Montagedaten erhalten angehangen an die Entitäten als Literale über sie ihre semantische Zuordnung.

Der *Order* produziert ein bestimmtes *Product*, dieses wiederum führt eine *ProductRecord* mit. Gleichzeitig unterhält der *Order* die *Assembly* des *Products* in einem zweiten Strang. Die *Assembly* unterhält alle *Workstations*, die für die *Assembly* des *Products* benötigt werden. Die *Workstations* führen entsprechende *Tools*, deren *Status* die Arbeitsbereitschaft signalisiert. Die *Assembly* teilt sich in eine Sequenz von *Processsteps*, die jeweils ein oder mehreren *Tasks* angehören. Ausgeführt werden die *Tasks* der *Processsteps* von einem *Worker* an der zugewiesenen *Workstation*, durch Anwendung des *Tool*, nach der Beschreibung des *Task*. Die Ausführung des *Processstep* wird durch ein *Status* Protokoll dokumentiert, zugleich wird die *Task*-Durchführung von der *Workstation* gemessen. *Measure* und die Unterklassen von *Value*, *UG*, *IST*, *OG* und *Soll* unterhalten die Messwerte und Rahmenbedingung der *Task* Ausführung.
Die Literal Namen sind an die Spaltenbezeichnungen (Einträge) der Montagedaten angelehnt und bestimmen die Zuordnung der Spaltenwerte zu den Literalen. Die Werte entsprechen bis auf wenige Ausnahmen den Werten den ursprünglichen Montagedaten der Zellen. Der *Pro-*

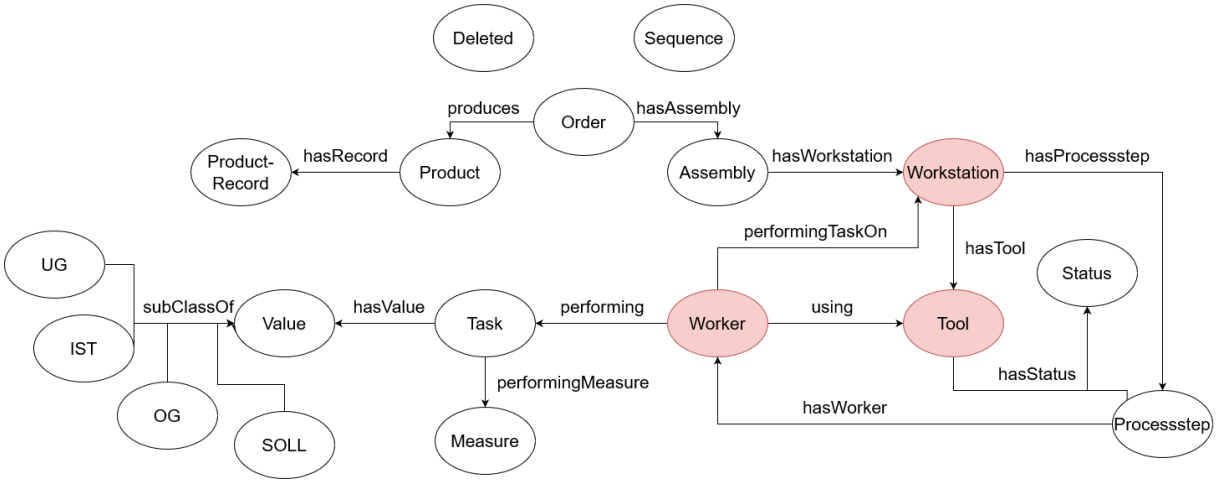


Abb. 8: buPAOv1 Ontologie ohne Literale. In rot hervorgehobene Entitäten werden einmalig vorab für alle Wissensgraphen erzeugt.

cessstep hat neben einer ID (*hasProcessID*), die ihn innerhalb seiner *Order* eindeutig identifiziert und die Ausführungsreihenfolge angibt, einen Namen. *hasProcessstepName* besteht aus dem festen Präfix *Processstep*, gefolgt von dem Präfix des *TAFNAME*. *Task* führt das Literal *hasTaskDescription*, das die ganze Aufgabenbeschreibung des *TAFNAME* unterhält. Der *Processstep* Name und die *Task* Beschreibung lässt eine Zuordnung der Entitäten über den gemeinsamen *TAFNAME* Anteil zu. *Worker* entnimmt die beteiligte Person an einem *Processstep* der *WERKER* Spalte und führt sie als *hasWorkerName*, da die Namen aus Gründen des Datenschutzes jedoch leer sind, ist es hinfällig und es gibt einen anonymisierten *Worker* für alle *Orders*. *Worker* führt auch eine Arbeitsschicht auf, zu der ein *Processstep* ausgeführt wurde, da diese Daten nicht vorliegen ist der Wert ebenfalls leer. Das Literal vervollständigt das Ontologie-Konzept. Ebenfalls der Vollständigkeit, ohne vorhandene Daten, führt jedes *Tool* eine *hasStatus* Beziehung zu einer *Status* Entität. *hasStatusID* führt die Einsatzbereitschaft des *Tool* als ID und *hasStatusDescription* die dazugehörige textuelle Beschreibung. *Processstep* führt dieselbe Beziehung zu *Status*, entsprechende Werte liegen in diesem Fall, aus der gleichnamigen Spalte *STATUS* für *hasStatusID* und *STATUSTEXT* für *hasStatusDescription*. *Tool* selbst hat einen Namen (*hasToolName*) und Anschluss (*hasPort*). Ein *Tool* kann mehr als eine Variante haben. Sie unterscheiden sich durch den Anschluss und werden unter demselben Namen verzeichnet. Um die Eindeutigkeit der *Tools* zu gewährleisten, bilden der Toolname und dessen Port Wert konkateniert die eindeutige ID (*hasToolID*). Die Ontologie führt zur konzeptionellen Vollständigkeit, die *Tool* Literale *hasToolDescription* und *nextMaintenance*. Eine *Workstation* beschreibt sich eindeutig durch dessen Bezeichnung, die Spalte *BEREICH* führt alle auf und wird auf das Literal *hasWorkstationName* gesetzt. *UG*, *IST*, *OG* und *SOLL* unterhalten ihren Wert aus ihrer jeweils gleichnamigen Spalte als *hasValue* Literal. Die Spalte *EINHEIT* liefert die zugehörige Einheit, abgebildet auf das *hasUnit* Literal. *Measure* unterhält die *MESSZEIT* als Zeitstempel (*hasTimestamp*), *MESSART* als *hasType*, den konkreten Messwert (*hasMeasureValue*) von *PROGRAMM* und die *BEDEUTUNG*, ermöglicht die Einordnung des Messwertes in den Montage-

prozess (*hasMeaning*). Die *Assembly* eines *Product* findet auf einer Montagelinie (*hasAssemblyLine*) statt, die und der Zeitpunkt (*hasTimestamp*) an dem das *Product* gefertigt wird, führt *Assembly* als Literale. Der Zeitpunkt existiert nicht in den Montagedaten, die Montagelinie unter der Spalte *BAND*. Die *Order* Entität führt die Stränge der Montage und Informatione des zu fertigen Produktes zusammen. Jeder *Order* hat eine eindeutige ID zur Identifizierung und übergreifende Metainformationen für den Auftrag, in diesem Fall die Bauanleitung und Bauteilliste. Die ID stammt aus der Spalte *MA_NR* und wird an das Literal *hasMAID* übergeben. Die Bauanleitung und Bauteilliste Literale *hasVisualisationID* und *hasPartList* bleiben mangels Daten leer. Das gefertigte *Product* hat folgende Metainformationen, die Seriennummer (Spalte *AEID*) als eindeutige ID (*hasProductID*), die Produktvariante bildet *VARIANTE_NUMMER* und *VARIANTE_NAME* in *hasVariantID* und *hasVariantName* ab. Produktname (*hasProductName*) und Produktbeschreibung (*hasLabel*). Letzten beiden bleiben leer, für sie sind keine Montagedaten vorhanden. Ein *Product* durchlebt eine Lebensspanne, dies bildet die Ontologie mit der *ProductRecord* Entität ab. Sie unterhält den Zustand als ID (*hasPRID*) und Beschreibung (*hasPRDescription*), die Montagedaten füllen *hasPRID* mit Werten aus *PLAID*. Der Vollständigkeit alle Montagedaten abzubilden, *Deleted* unterhält die Werte der Spalte *GELÖSCHT* als *hasDeletedID* und *Sequence* die *SEQUENZNR* als *hasSequenceID*. Die Tabelle 8 im Anhang A.6 führt die Zuordnung der Montagedaten Spalten zu Literalen auf.
Die beschriebene Ontologie durchlief einige Iterationen, deren Haupttreiber technische Details waren. So benötigte die Ontologie die ergänzende Beziehung namens *hasPart* (Illustriert in Abbildung 10). Eine gerichtete Beziehung von der *Order* Entität jeweils zu den Entitäten *Processstep*, *Task*, *Tool*, *Deleted* und *Sequence*. Sie dienen der Graphdatenbank (Kapitel 3.4) im Sampling. Alle *hasPart* Beziehungen wurden anschließend abgesetzt und die Ontologie durch die Kanten *belongsTo* und *hasOrder* ergänzt. Beide Beziehungen sind eingehende der *Order* Entität, *belongsTo* ausgehend der *Product* und *hasOrder* der *Assembly* Entität (Illustriert in Abbildung 13). Damit haben alle Entitäten mindestens eine eingehende Kante, dies setzt das Modell Embedding für das Training voraus. *Deleted* und *Sequence* fallen als isolierte Knoten somit weg.
Technisch nutzbar wird die Ontologie durch die Modellierung in der Anwendung Protegé und Speicherung im RDF Format. Der eindeutige Namespace der Ontologie lautet: http://www.biba.uni-bremen.de/kprueger/assembly/onto/v1#. Eine beispielhafte Wissensgraph-Instanz ist in Abbildung 9 für die *Order940* mit der *MA_NR* 65421123 dargestellt. Sie zeigt die Montage des Produkts mit der gleichnamigen Auftragsnummer 6542123, bestehend aus einem unbekannten Prozessschritt. Ein Prozessschritt erhält den Namen *Processstep [UNKNOWN]*, wenn die *TAFNAME* Zelle, die die Beschreibung der Aufgabe unterhält, leer ist. Die Entitäten und Literal Werte sind Knoten, Entitäten hellblau rund und Literale hellgrün rechteckig. Die Beziehungen sind gerichtete Kanten zwischen den Entitäten, gezogen nach der Ontologie. Die Identifier stehen auf den Kanten. Die Zugehörigkeit der Literal Werte zu den Entitäten erfolgt über gerichtete Kanten von den Entitäten zu den Literal Knoten. Die Literal Label stehen auf den Kanten. Die Entitäten *Deleted* und *Sequence* werden aus Übersichtsgründen nicht mit an-

gezeigt. Ebenso wenig dessen Literale. Die technisch notwendigen Beziehungen *belongsTo* und *hasOrder* existieren in der *RDF* Form nicht und entsprechend ebenfalls nicht in dem Graphen aufgeführt.

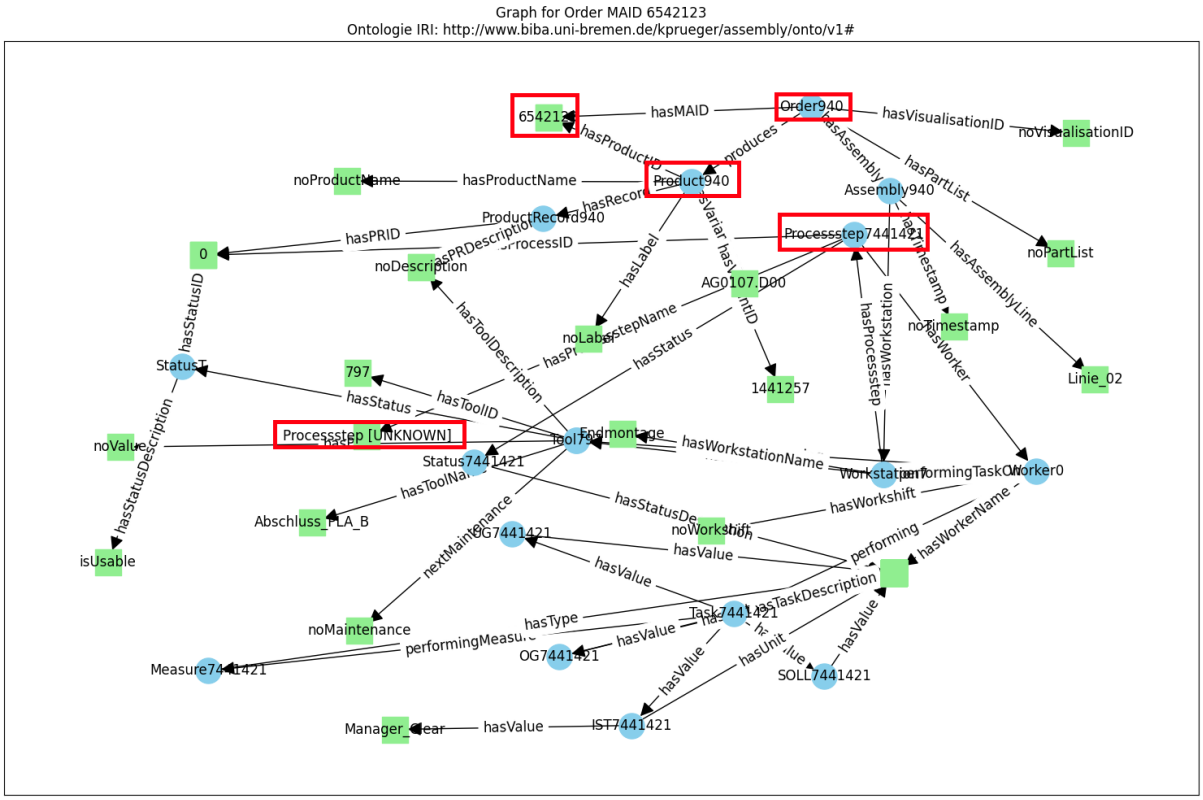


Abb. 9: Wissensgraph Instanz für die *Order940* mit der *MA_NR* 6542123. Es zeigt die Montage des Produktes *Product940* mit der Seriennummer 6542123. Es wird ein unbekannter Prozessschritt ausgeführt. Die Literale sind hellgrüne Rechtecke und die Entitäten hellblaue Kreise. Ihre Werte oder Identifier stehen auf den Knoten. Die Beziehungen sind gerichtete Kanten. Die Entitäten *Deleted* und *Sequence* sind aus Übersichtsgründen nicht aufgeführt. Die Kanten *belongsTo* und *hasOrder* existieren in diesem Format nicht und sind ebenfalls nicht aufgeführt.

3.4 GraphDB

Die Graphdatenbank „GraphDB“ verwaltet und visualisiert die Wissensgraphen effizient. Die Visualisierung hilft die Korrektheit der Beziehungen und Zuordnung der Montagedaten zu prüfen. Die Hauptfunktion umfasst das Abfragen des Wissensgraph mit der SPARQL-Sprache. Das Ziel ist der Entwurf einer SPARQL Query, der für einen Montageauftrag über dessen Auftragsnummer *MA_NR* den Wissensgraph sampelt.
GraphDB erfüllte die Anforderungen an eine Graphdatenbank für den Anwendungsfall. Diese umfassen eine frei zugängliche Version, minimaler Speicherverbrauch, selber Hosten einer eigenen Standalone Instanz, zukünftige Weiterentwicklung, eine ausführliche Dokumentation und verfügbar als Docker Image.

Nach dem Initialisieren des Repository der Graphdatenbank bietet GraphDB drei grundsätzliche Möglichkeiten des Imports von Wissensgraph-Daten. Hochladen der Dateien per REST-API-Abfrage, Verwendung des Upload-Tools oder Hochladen über die Benutzeroberfläche. Keine der Möglichkeiten bot einen entscheidenden Geschwindigkeitsvorteil, daher entschied das Kriterium Benutzerkomfort. Die Dateien wurden über die Benutzeroberfläche hochgeladen. Gleichermaßen wird auch die RDF-Datei der Ontologie, die mit Protegé erzeugt wurde, hochgeladen und offenbart dem Repository die Struktur und Datentypen der Wissensdaten.
Der Entwurf der SPARQL Query und damit die verbundene Möglichkeit der Erzeugung des Datensatzes für das Graph Modell ist der letzte Verarbeitungsschritt vor dem Graph Matching. Wie im Kapitel 3.3 bereits angeschnitten, erleichtert die Einführung der Beziehung *hasPart* in die Ontologie, die Zugehörigkeit der Entitäten *Tool*, *Processstep*, *Deleted*, *Sequence* und *Task* zum jeweiligen *Order* durch jeweils eine direkte Kante. Eine minimale Anpassung entfernte die *hasPart* Kante zwischen den Order und Task Entitäten und ersetzt sie durch eine gleichnamige Kante von dem *Processstep* zur dazugehörigen Task-Entität (Illustriert in Abbildung 10). Die Kante erreicht, dass weniger Bedingungen benötigt werden, die wiederum die Komplexität und damit den Zeitaufwand der SPARQL Query Abfrage verringert. Das Filtern über die Menge aller *Processsteps* und *Tasks* des Datensatzes entfällt. *Order* hat eine *hasPart* Beziehung zu allen dazugehörigen Entitäten *Tool*, *Processstep*, *Deleted* und *Sequence*. Über die Beziehung werden sie schlicht abgefragt. Ohne die Beziehung müsste mit Bedingungen sichergestellt werden, das beispielsweise nur die *Tools* genommen werden, die den *Workstations* über *hasWorkstation* angehören, die wiederum dem *Assembly* über *hasAssembly* der aktuelle abgefragten *Order* angehören. Die Bedingungen zu prüfen benötigt mehr Rechenzeit, als direkte Abfragen über*hasPart*. Diese Vorarbeit dient der Prävention der angesprochenen Schwäche von *RDF*, das nicht guten skalieren bei komplexen Abfragen.

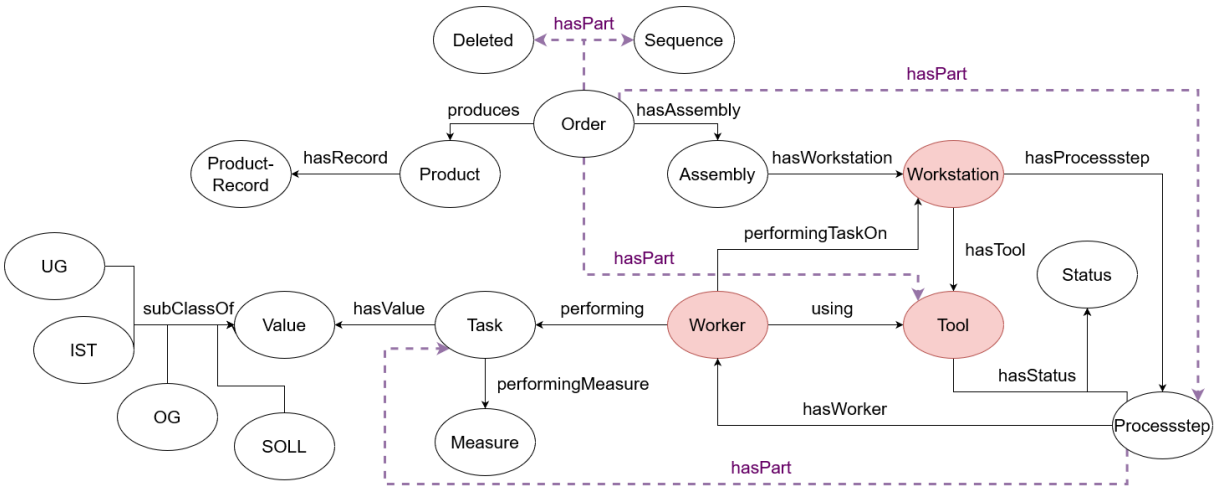


Abb. 10: buPAOv1 Ontologie ohne Literale. Erweitert um die *hasPart* Beziehung.

Ausgehend von einer *Order*, erhalten durch die Übergabe der *MA_NR* an die Query, werden die Entitäten über die Beziehungen gesampelt. Anschließend die Literale aller gesampelten Entitäten über dessen Literal Beziehungen. *Order* ermittelt über die Ontologie Beziehung *pro-*

duces die zugehörige *Product* Entität und der wiederum über seine *ProductRecord*. *Order* nutzt die eingeführte *hasPart* Beziehung für *Processstep*, *Task*, *Deleted*, *Sequence* und *Tool*. Die *Processstep* und *Tool* Entitäten sampeln dann dessen *Status* Entitäten. Es folgen *Assembly*, *Workstations* und die *Worker* die an den *Processsteps* und *Workstations* beteiligt sind. Jeder *Task* sampelt dessen Zugehörige *Measure*, *OG*, *IST*, *SOLL* und *UG*. Abschließend werden alle *hasPart* Hilfsbeziehungen entfernt, sie dienten zum vereinfachten sampling, haben aber keine semantische Bedeutung.

Der Codeblock 3.1 zeigt an dem Ausschnitt der SPARQL Query die wesentlichen Konzepte für die Abfrage der Wissensgraphen, anhand der *Order* und *Product* Entitäten und dessen Beziehung *produces*.
CONSTRUCT legt die Variablen *?order* und *?product* für die Entitäten *Order* und *Product*, und dessen Literale, sowie Ihre Beziehungen an. Das Konstrukt folgt dem Triple-Format, *?[Entität Variable]*, *:[Beziehung]* und *?[Entität/Literal] Variable]*. *WHERE* sucht in den Triple Daten passende Entitäten und Literal-Werte über die Beziehungen. *„{maid}“* ist ein Platzhalter für die Auftragsnummer. Sie muss vor der Abfrage gesetzt werden. Von Ihr geht die Abfrage aus, sie bestimmt für welchen Auftrag der Wissensgraph gesampelt wird. Über sie wird die eindeutige *Order* Entität gefunden, und über die Beziehungen wie *produces* dann alle weiteren Entitäten. Die Literal-Abfragen befinden sind in *OPTINAL*-Blöcken, so wird sichergestellt, dass im Fall, dass kein Literal-Wert für eine Entität gefunden wird, die Abfrage ordnungsgemäß weiter laufen kann.
Die Sampling-Abfrage erfolgt per angebotener REST-API Schnittstelle der GraphDB, übergeben wird die SPARQL Query und die Response ist ein Wissensgraph im XML-Triple Format. Es gelang jedoch nicht eine Response für eine Query zu erhalten. Auch die für die Sampling-Abfrage optimierte Ontologie ließ keine SPARQL Query erzeugen, die in terminierender Laufzeit die Wissensgraphen sampelt. Der REST Punkt der GraphDB gab nach 30 Minuten keine Response für die SPARQL-Query. Die Query Funktionalität wurde zuvor in der GraphDB Weboberfläche bestätigt. Sie terminierte nach 15 Sekunden. Ebenso wurde der REST-API Endpunkt mit einer reduzierten SPARQL Query getestet. Sie gab die gewünschte Response. Es gelang jedoch nicht eine Response für die vollständige Query zu erhalten. Daher erhielt der Data Handler eine Erweiterung um die Funktion des direkten sampeln der Wissensgraphen aus der Konvertierung.

```
PREFIX : <http://www.biba.uni-bremen.de/kprueger/assembly/onto/v1#>

CONSTRUCT {
  # Order Attributes
  ?order :hasMAID ?maid ;
         :hasVisualisationID ?visID ;
         :hasPartList ?partList ;
         :produces ?product ;
         :hasAssembly ?assembly .

  # Product Attributes
  ?product :hasProductID ?productID ;
           :hasVariantID ?variantID ;
           :hasVariantName ?variantName ;
           :hasProductName ?productName ;
           :hasLabel ?label ;
           :hasRecord ?productRecord .

  ...
} WHERE {
  # Match Order with the specific MAID
  ?order :hasMAID "{maid}" .
  OPTIONAL { ?order :hasMAID ?maid . }
  OPTIONAL { ?order :hasVisualisationID ?visID . }
  OPTIONAL { ?order :hasPartList ?partList . }

  # Match the Product related to the Order via produces relationship
  ?order :produces ?product .

  # Retrieve literals for Product
  OPTIONAL { ?product :hasProductID ?productID . }
  OPTIONAL { ?product :hasVariantID ?variantID . }
  OPTIONAL { ?product :hasVariantName ?variantName . }
  OPTIONAL { ?product :hasProductName ?productName . }
  OPTIONAL { ?product :hasLabel ?label . }

  ...
}
```

Programm 3.1: Ausschnitt der *final_sparql.sparql* SPARQL Query. *CONSTRUCT* legt die Entitäten *order* und *product*, und dessen Literale, sowie Ihre Beziehungen an. *WHERE* sucht in den Triple-Daten passende Entitäten und Beziehungen. *"maid"* wird durch die Auftragsnummer ersetzt, von dieser geht die Abfrage aus. Die Literal-Abfragen befinden sich in *OPTINAL*-Blöcken, so wird sichergestellt, dass im Fall, dass kein Literal für eine Entität gefunden wird, die Abfrage ordnungsgemäß weiter laufen kann.

3.5 Graph Embedding Implementierung

Graph Embedding oder Embedding umfasst ein Deep Learning Modell, welches eine effiziente Repräsentation der Wissensgraphen erlernt, damit anschließend im Embeddingraum zwei Wissensgraphen verglichen werden können.

Die Umsetzung des Deep Learning Modells nach dem GraphSAGE Paper (Hamilton et al., 2018) wird in Folge als „Custom GraphSAGE Modell“ oder kurz „Modell“ betitelt. Das Konzept wurde bereits in Kapitel 2.4 dargelegt. Die Modelleingabe ist ein Auftrag im heterogener Graph (Wissensgraph) Format und die Ausgabe eine Vektorrepräsentation des Graphen, wobei jeder Knoten als ein Vektor repräsentiert wird. Die ausführliche Beschreibung der Sample erfolgt in Kapitel 3.6. Aus vier SAGEConv (SAGE Convolution) Layern besteht das Modell, die jeweils die Features der Nachbarknoten über die Kanten aggregieren. Verwendet wird der *mean* Aggregator, der über die aggregierten Feature mittelt (Hamilton et al., 2018). Ein Knoten erhält sein Embedding aus den Feature von Nachbarknoten bis zur Tiefe vier, durch die vier SAGEConv Layer. Auf die ersten drei Layer folgt die Aktivierungsfunktion ReLU, der letzte SAGEConv Layer dient als Ausgabe-Layer ohne Aktivierung. Das Modell ist heterogen, das bedeutet, jeder Knoten gehört einen Knotentyp an, der die Featurelänge bestimmt, dies soll dem Modell die Möglichkeit eröffnen, eine reichhaltigere Repräsentation der einzelnen Knoten zu erlernen. Die Featurelänge aller Knoten ist nicht normiert auf eine Länge sondern variiert zwischen den Knotentypen, so können gezielt Feature der Knoten im Training berücksichtigt werden. Anders homogene Graphen, alle Knoten sind vom selben Typ und haben die selbe Featurelänge, entsprechend fließen weniger vielfältige Charakteristiken in das Training ein.
Die technische Umsetzung erfolgte in der Sprache Python. Die Implementierung stützt sich auf die Bibliotheken *pytorch* und *pytorch geometric*. *pytorch geometric* bot eine Implementierung des GraphSAGE Modells und dessen Layer, sowie die Unterstützung für Heterogenität, sowohl für Modelle als ebenso für die Sample bzw. Datensätze. Genutzt wurden die Layer, es sind die angesprochenen vier SAGEConv Layer, aus der Bibliothek in der eigenen Implementierung des „Custom GraphSAGE Modell“.
Das Training folgte dem Unsupervised Ansatz des Papers, die „graph-based loss function“

$$\mathcal{L}_u = -\log \sigma(\mathbf{z}_u^\top \mathbf{z}_v) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log \sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n}),$$

Hamilton et al. (2018, Siehe Sek. 3.2). Es zeigt die Loss Berechnung für einen Knoten *u*. *v* ist ein Nachbarknoten von *u*. σ ist die Sigmoid Funktion. Q die Anzahl an negativen Knoten und P_n stellt die negative Knotenverteilung dar. Das Ziel des Loss ist das nahe Zusammenliegen positiver Knotenpaare im Embeddingraum, während die Distanz zu den negativen Knoten vergrößert wird. Das Loss wurde für das „Custom GraphSAGE Modell“ an den Umgang mit heterogenen Samplen, die als Batches übergeben werden, folgendermaßen angepasst. Für jeden Kantentyp wird für das positive Loss die Start- und Zielknoten summiert. Die Begrenzung der Sigmoid Normalisierung der Summen, auf $1e-12$ bis $1-(1e-12)$ stabilisiert das Training. Es

verhindert das Auftreten kleiner Gradienten, während des Trainingsprozesses. Anschließend wird über alle gemittelt. Für das negative Loss erfolgt dieselbe Berechnung, entsprechend für die Startknoten und negativen Zielknoten. Anschließend erfolgt die Normierung des gesamten Loss (Loss für ein Batch), indem durch die Anzahl der Kantentypen geteilt wird. Die beschriebenen Anpassungen des Loss münden in Folgender Funktion:

$$
\begin{aligned}
\epsilon &= 10^{-12} \\
\text{pos\_loss} &= -\frac{1}{N}\sum_{i=1}^{N}\log\left(\text{clamp}\left(\sigma\left(\sum_j \text{pos\_src}_j \cdot \text{pos\_dst}_j\right), \min=\epsilon, \max=1-\epsilon\right)\right) \\
\text{neg\_loss} &= -\frac{1}{N}\sum_{i=1}^{N}\log\left(\text{clamp}\left(1-\sigma\left(\sum_j \text{pos\_src}_j \cdot \text{neg\_dst}_j\right), \min=\epsilon, \max=1-\epsilon\right)\right) \\
\text{loss}_k &= \text{pos\_loss} + \text{neg\_loss}
\end{aligned}
$$

$loss_k$ ist das Loss für ein Batch von einem Kantentyp. *clamp* stellt mit *min* und *max* die Begrenzung der Sigmoid Aktivierung dar. *pos_src* und *pos_dst* sind die Start- und Zielknoten der betrachteten Kante und *neg_dst* die negativen Zielknoten.
Die entscheidende und in diesem Modell für jeden Batch vor der Berechnung des Loss auszuführende Funktion ist das negative Sampling. Für jeden Knoten des Batches wird ein Zielknoten aus dessen Sample pseudo zufällig bestimmt. Dieser Zielknoten, auch als negativer Zielknoten betitelt, ist weder vom selben Typ noch ein direkter Nachbar des Startknotens.
Ein Knoten verwendet für jede seiner ausgehenden Kanten denselben negativen Zielknoten in einer Epoche. Bei mehreren Kanten, wie sie die *Worker* Knoten haben, für alle Kanten derselbe negative Zielknoten. Diese Schwäche gleicht das Training über die Epochen hinweg aus. Die Notwendigkeit, pro Kante des Batch einen negativen Zielknoten zu wählen, ist damit hinfällig. Die beliebige Wahl eines negativen Zielknotens aus der Knotenmenge des Batches ist nicht zielführend. Sample übergreifende Überschneidung ist nicht gestattet, es sind getrennte Graphen und die Repräsentation der Knoten wird innerhalb der Beziehungen eines Graphen erlernt. Die Menge der negativen Zielknoten ist abzüglich direkter Nachbarn und dessen eigenen Knotentyp weiter eingeschränkt. Knoten desselben Knotentyps kommen nicht als negative Zielknoten infrage, trotz der Beschaffenheit, keine direkten Nachbarn zu sein, da Knoten des selben Knotentyps. Darüber hinaus erhalten unterschiedliche Startknoten desselben Knotentyps denselben negativen Zielknoten pro Batch, dies beruht auf der Dysbalance der Knotenverteilung über alle Knotentypen. Knotentypen wie *Order*, *Product*, *Assembly*, *Workstation* und *Worker*, haben einen oder wenige Knoten, während *Processstep* und *Task* viele Knoten repräsentieren, die jeder einen negativen Zielknoten benötigt und damit statistisch wahrscheinlicher denselben Knoten erhalten.
Das Loss ist der Hauptanhaltspunkt zur Einschätzung des Trainingsfortschrittes des Modells. Im Supervised Training hilft die Accuracy Metrik bei der Einschätzung, die Label ermöglichen eine Bestimmung der Diskrepanz der Modellvorhersage gegenüber dem erwarteten Ergebnis für einzelne Sample. Im Unsupervised Fall ist die zu trainierende Aufgabe weniger präzise,

fehlende Label verhindern eine Bestimmung einer Accuracy Metrik, es bedarf einer händischen Prüfung der erworbenen Modellfähigkeit. Die Visualisierung des Graphen- bzw. Knotenembedding des Modells nach dem t-SNE Verfahren soll eine Einschätzung der Ausgabequalität ermöglichen. t-SNE reduziert die Feature des Embedding auf zwei oder drei Dimensionen und ermöglicht eine entsprechende Darstellung, die begutachtet werden kann. In einem Plot werden alle embedded Knoten eines Graphen dargestellt, wobei die Knotentypen als Klassenzuordnung dienen. Die Hoffnung ist, einen Eindruck davon zu erhalten, wie das Modell die Knoten Nähe innerhalb eines Knotentyps, sowie die Beziehungen der Knotentypen untereinander gestaltet. Abbildung 11 zeigt die Knoten des *buPAOv1_MORE_EFFICIENT_v2_12*. Konkret die Embeddings der Knotentypen: *ist*, *measure*, *og*, *processstep*, *soll*, *status*, *task*, *tool* und *og*. Knoten eines Knotentyps haben dieselbe Farbe. Ein zweiter Plot soll weitere Aufschlüsse über die Beziehungen und relevanten Features innerhalb eines Knotentyps geben, indem alle embedded Knoten eines Knotentyps dargestellt werden. Offenbart der Plot einen eindeutigen Shift zu einem oder mehreren Features, ist die Fragestellung, die der Plot helfen soll, zu beantworten. Abbildung 12 zeigt das Embedding der *processstep* Knoten, des *buPAOv1_MORE_EFFICIENT_v2_12*.

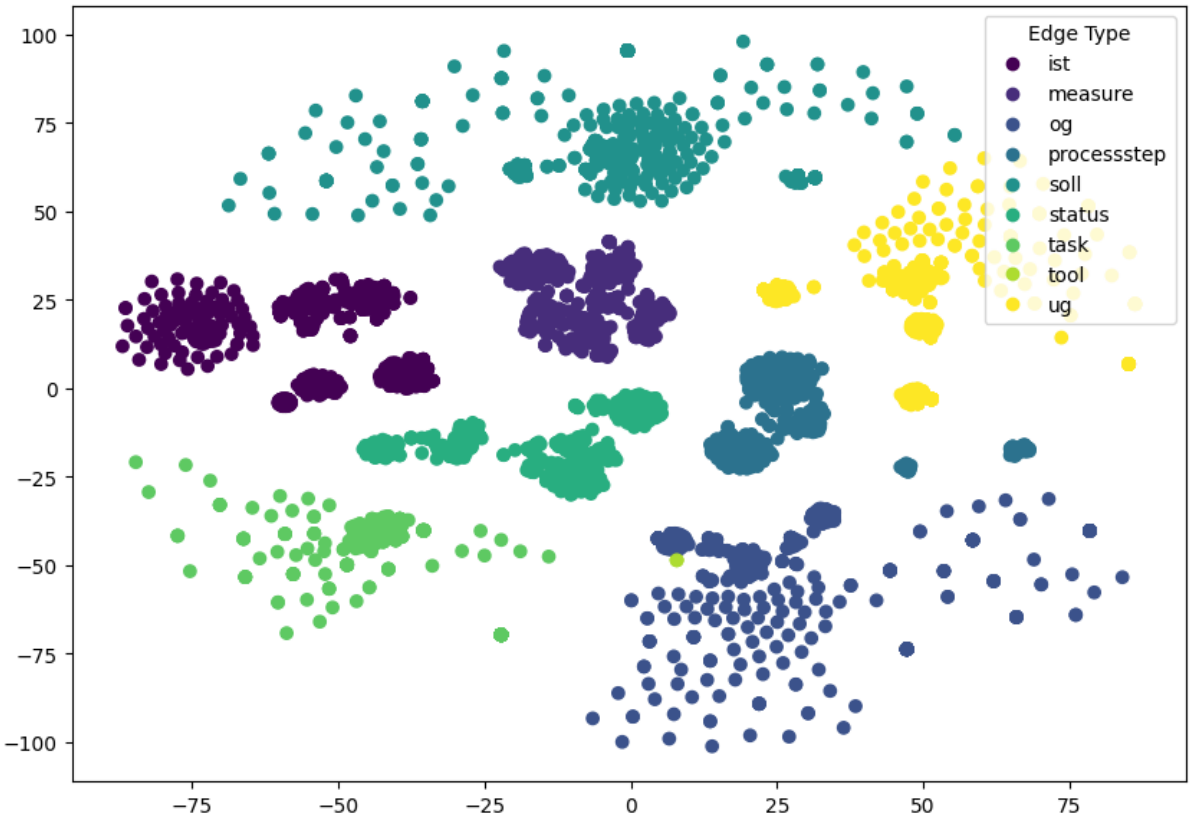


Abb. 11: Zeigt die Embeddings der Knotentypen: *ist*, *measure*, *og*, *processstep*, *soll*, *status*, *task*, *tool* und *og*, reduziert auf Zwei Dimensionen, nach dem t-SNE Verfahren. Konkret die Knoten des *Sample12*.

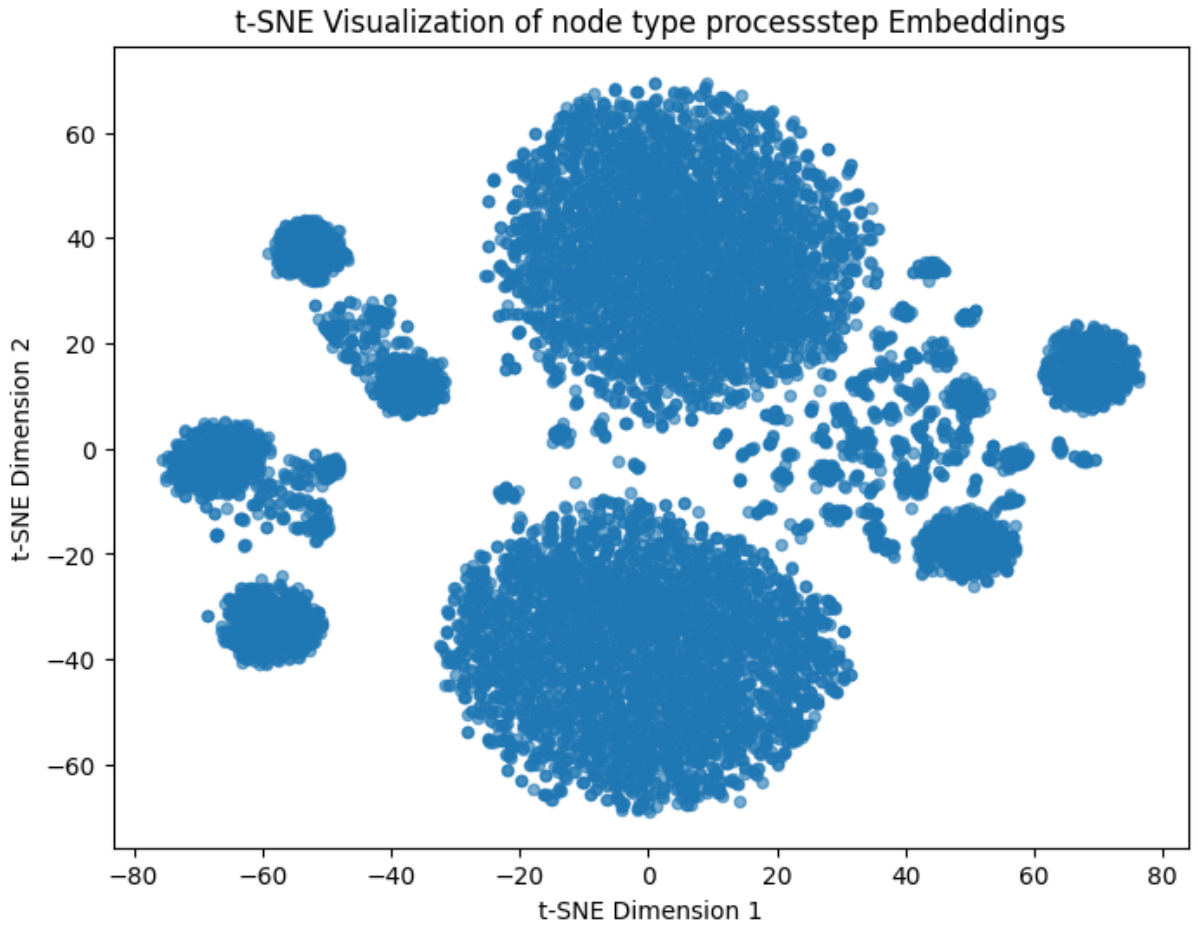


Abb. 12: Zeigt das Embedding der *processstep* Knoten, reduziert auf zwei Dimensionen, nach dem t-SNE Verfahren. Konkret die Knoten des *Sample12*.

Die Visualisierung hilft, die Qualität des Modells nach dem Training zu bewerten. Um den Trainingsfortschritt während des Trainings besser beurteilen zu können, wird neben dem Loss nach jeder Epoche ein durchschnittlicher Silhouette Score für alle Knotentypen berechnet. Anstelle einer Accuracy Berechnung. Es soll einen Eindruck vermitteln, wie gut das Modell die Generalisierung der Knotentypen im Durchschnitt gelernt hat. Über die Knotenmenge jedes Knotentyps wird ein Silhouette Score bestimmt. Anschließend werden alle Scores summiert und durch dessen Anzahl geteilt.
Der Silhouette Score misst die Konsistenz innerhalb von Clustern und die Separierbarkeit zwischen Clustern, wobei höhere Werte auf eine gute Clusterstruktur hinweisen und niedrigere Werte auf mögliche Überschneidungen oder schlechte Zuordnungen (Rousseeuw, 1987).
Ein Silhouette Score nahe 1 deutet darauf hin, dass die Knoten klar und gut getrennt in Clustern angeordnet sind. Ein Wert nahe 0 zeigt, dass die Knoten an den Rändern von Clustern liegen und die Zuordnung unsicher ist. Negative Werte weisen auf fehlerhafte Clusterzuweisungen hin, bei denen Knoten in falsche Cluster eingeordnet werden.
Die Annahme, ähnliche Knoten liegen im Embedding nah aneinander und lassen sich grob in Klassen unterteilen. Über die Epochen lässt sich abzeichnen, ob ähnliche Knoten weiter zusammengeführt werden, also (kompaktere) Klassen bilden. Ein stetig wachsender bzw. großer positiver Silhouette Score zeigt dies. Ein stagnierender oder gar sinkender Silhouette Score, das Lösen der Klassenbildung.
Diese Metrik hat starke Einschränkungen, wie das Setzen einer festen Klassenanzahl, damit vorab mit k-Means die Knoten geclustert werden können. k-Means bestimmt jeweils für die Knoten eines Knotentyps die Klassen Zentren und gibt den Knoten eine Klassenzuordnung. Dazu kommt aus Laufzeit- und Speicherbegrenzung wird die Bestimmung des Score über eine Teilmenge der Knoten ausgeführt, was wohl bemerkt die Aussagekraft der Metrik weiter schwächt.

Welche technischen Anpassungen das Modell für die Sample voraussetzt, folgt im kommenden Kapitel 3.6.

3.6 Datensatz

Die in Kapitel 3.2.1 beschriebenen Sample werden in einem letzten Vorverarbeitungsschritt an das verlangte Eingabeformat des *Custom GraphSAGE Modells* angepasst. Neben der strukturellen Änderung, der Anordnung von Knoten und Kanten, wird die Überführung der Knoten Feature in Vektoren vollzogen. Dem Laden des Datensatzes, mit Sample laden, Datensatz splitten und dem Bereitstellen der Sample als Batches, folgt der üblichen Konvention.

Der Datensatz hat 1113 Sample. Sie liegen als einzelne Pickle Dateien vor und kommen gesamt auf $33,8$ GB. Jedes Sample ist ein Wissensgraph im *HeteroData* Format, der einem Auftrag zuzuordnen ist.

Die *pytorch geometric* Bibliothek bietet neben der Funktionalität, der Anpassung eines Graph Modell in ein heterogenes Graph Modell, mit *HeteroData* auch die Klasse, Sample heterogen darzustellen. *HeteroData* ist angelehnt an ein Python Dict, es bildet die Heterogenität der Graphen ab. Jeder Knotentyp ist ein Key und unterhält die Knoten Feature des Typs als einen Tensor. Die Kanten werden als Triple-Key, bestehend aus Startknoten, Kantenbezeichnung und Zielknoten abgebildet. Die Triple-Keys unterhalten Index Paare der Start- und Zielknoten, die diese Kantenbeziehung im Graph führen (PyTorch-Geometric-Team, 2023).
Technische Limitierungen stießen die Notwendigkeit an, neben der zuvor beschriebenen Überführung des Datenformats, die ursprüngliche Ontologie anzupassen (Angepasste Ontologie 13). Die SAGEConv Layer können nicht mit isolierten Knoten umgehen, da sie zum Lernen des Knoten Embedding die Nachbarknoten mit einbezieht und mindestens eine eingehende Kante benötigt, über die sie Informationen bezieht. Aus demselben Grund wurde die Ontologie um zwei zusätzliche eingehende Kanten *belongsTo* und *hasOrder* erweitert, da Order keine eingehende Kante aufwies. Die isolierten Knoten *Deleted* und *Sequence* wurden aus den Graphen entfernt, da der Gewinn an Komplexität durch ihre Einbindung den semantischen Nutzen deutlich überstieg. Anstelle der manuellen Anpassung bestand die Möglichkeit, die Graphen in ungerichtete Graphen umzuwandeln, wobei fehlende gerichtete Kanten automatisch ergänzt worden wären. Der damit einhergehende Aufwand des Modelltrainings durch die steigenden Graphen Größen und Abnahme der semantischen Bedeutung der Beziehungen sprachen dagegen.
Wie aus der zuvor beschriebenen technischen Umsetzung bereits deutlich wurde, bilden heterogene Graphen die Semantik besser als homogene Graphen ab, die alle Knoten auf einen Typen reduzieren. Das heterogene Training gibt dem Modell den Raum, die Feinheiten der einzelnen Knotentypen im Training zu berücksichtigen. Ein entscheidender, aber bisher weitestgehend außer Acht gelassener Punkt, sind die Knoten Feature. Wie diese in Vektoren übersetzt und was deren Besonderheiten im Kontext von heterogenen Graphen sind, folgt im Unterkapitel 3.6.1.

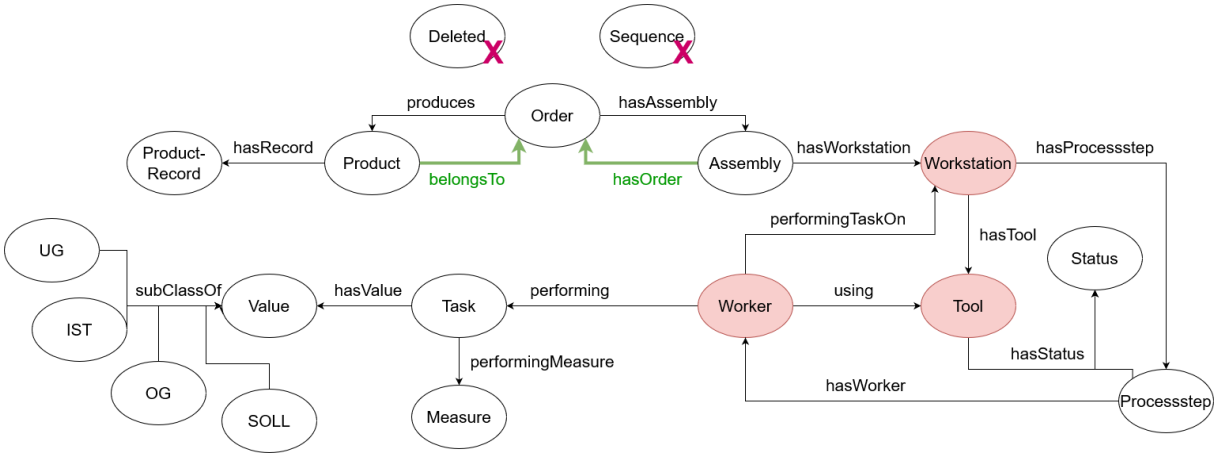


Abb. 13: buPAOv1 Ontologie ohne Literale. Ergänzt um die *belongsTo* und *hasOrder* Kante und Entfernt der *Deleted* und *Sequence* Entitäten.

3.6.1 Word2Vector Vektorisierung

Mit den Begriffen der Montagedaten wurde ein Word2Vector Modell trainiert. Die Feature jedes Knoten werden vektorisiert und anschließend konkateniert.

Die Vektorisierung ist inspiriert vom GraphSAGE Paper. Das *Gensin word2vec emb* Modell lernt eine Menge von Sätzen (Korpus), als Vektoren einer festen Länge zu repräsentieren (Hamilton et al., 2018).
Der gegenwärtige Korpus umfasst 7645419 Wörter. Sie bestehen aus den einzigartigen Wörtern der Montagedaten, gepaart mit natürlichen Zahlen (Null enthalten) und technisch hilfreichen Begriffen. Nutzen und Zusammenhang letztere Begriffe kamen in Kapitel 3.3 zur Sprache. Trainiert wurde ein *Gensin word2vector emb* Modell mit den Wörtern des Korpus. Die trainierte Vektorlänge ist 32. Die Länge ist ein Kompromiss aus moderatem Speicherverbrauch und ausreichend Dimensionen, die eine feine semantische Darstellung der Beziehungen zulassen. Grundsätzlich ist ein eigenes Word2Vector Modell der Notwendigkeit der Domäne Fachbegriffe geschuldet. Für die Integration neuer Begriffe ist ein Nachtrainieren des *word2vector* Modells notwendig.
Der Feature-Vektor eines Knotens entsteht durch die Konkatenation all seiner Feature, wobei jedes Feature ein Literal ist. Die Gesamtlänge des Feature-Vektors ergibt sich aus der Anzahl der Feature multipliziert mit der Länge des Wortvektors im *word2vec* Modell. Beispielsweise hat der Knoten *Order* mit drei Feature (*hasMAID*, *hasVisualisationID*, *hasPartList*) und einer Wortvektorlänge von 32 eine Feature-Länge von $3 \times 32 = 96$. Die Heterogenität erlaubt es Knoten, alle ihre Feature individuell zu embedden, die das *Custom GraphSAGE Modell* bei dem Training berücksichtigt und eine tiefere semantische Repräsentation ermöglicht.

3.7 Graph Matching Implementierung

Graph Matching beschreibt den Vergleich zweier Wissensgraph in einem reduzierten Embeddingraum auf Ähnlichkeit.

Das Siamese Modell erhält die zwei Wissensgraphen, die verglichen werden sollen als Eingabe, diese werden separat vom *Custom GraphSAGE Modell* embedded. Anschließend wird für jeden Knotentyp die Knotendistanz bestimmt. Ausgehend von dem kleineren Embedding (geringere Anzahl an Knoten) wird für jeden Knoten der ähnlichste Knoten des anderen Knoten Embedding bestimmt. Der ähnlichste Knoten ist der mit der geringsten Distanz im Embeddingraum. Für das bestimmte Knotenpaar wird die Ähnlichkeit mithilfe der Metriken bestimmt. Zur Distanzbestimmung wird die Kosinus-Ähnlichkeit und Euklidische Distanz verwendet. Die Knotendistanz gibt Aufschluss über dessen Ähnlichkeit. Der Wert wird auf einem Skalar von 0.0 - 1.0 normiert. Bei 1.0 spricht man von identischen Knoten und bei 0.0 haben die Knoten keine Ähnlichkeit. Abschließend wird der Ähnlichkeit der gesamten Graphen durch Mittlung aller Paar Distanzen bestimmt.
Die normierte Ähnlichkeit für ein Knotenpaar berechnet sich folgendermaßen. Anwendung findet die Euklidische Distanz. Für einen Knoten e_1 (aus dem kleineren Embedding) und der ähnlichste Knoten e_2 (aus dem größeren Embedding) wird die euklidische Distanz berechnet. Dabei ist d die Dimensionalität der Embeddings.

$$d(e_1, e_2) = \sqrt{\sum_{i=1}^{d}(e_{1,i} - e_{2,i})^2}$$

Um die Distanz in eine Ähnlichkeit umzuwandeln, wird die Euklidische Distanz normiert, indem die Distanz durch die Wurzel der Dimensionalität d des Embeddings geteilt wird. Dadurch liegt die normierte Distanz theoretisch im Bereich von $[0, \infty)$, wobei größere Werte eine größere Entfernung (weniger Ähnlichkeit) anzeigen.

$$\text{normierte Distanz} = \frac{d(e_1, e_2)}{\sqrt{d}}$$

Die normierte Distanz wird invertiert, um eine Ähnlichkeit zu erzeugen.

$$\text{Euklidische Ähnlichkeit}(e_1, e_2) = 1 - \text{normierte Distanz}$$

Das Knotenpaar ist mit 1.0 maximal ähnlich, wenn die Knoten identisch sind, also die Distanz $d(e_1, e_2) = 0$ beträgt. Entsprechend keine Ähnlichkeit (0.0) hat das Knotenpaar, wenn deren Distanz $d(e_1, e_2) \geq \sqrt{d}$ ist.

Für die Kosinus-Ähnlichkeit ergibt es sich folgendermaßen: Für das Knotenpaar e_1 und e_2 wird die Kosinus-Ähnlichkeit berechnet.

$$\text{Kosinus-Ähnlichkeit}(e_1, e_2) = \frac{e_1 * e_2}{\|e_1\|\|e_2\|}$$

Der Wertebereich erstreckt sich von -1.0 - 1.0, die Normierung skaliert sie auf 0.0 - 1.0.

$$\text{Normalisiert-Kosinus-Ähnlichkeit}(e_1, e_2) = \frac{\text{Kosinus-Ähnlichkeit}(e_1, e_2) + 1}{2}$$

Die Ähnlichkeit des Knotenpaar bei der Kosinus-Ähnlichkeit begründet sich auf den Winkel zwischen dem Knotenpaar. Sind die Knoten identisch, ist der Winkel $0°$:

$$cos(\theta) = 1 \Rightarrow \text{Kosinus-Ähnlichkeit}(e_1, e_2) = 1$$

Bei entsprechend keiner Ähnlichkeit, zeigen die Knoten in entgegengesetzte Richtungen, der Winkel ist $180°$:

$$cos(\theta) = -1 \Rightarrow \text{Kosinus-Ähnlichkeit}(e_1, e_2) = -1$$

Die Wahl der zwei Metriken begründet sich auf deren Eigenschaft der Einfachheit. Und als etablierte Verfahren reduziert es die Fehleranfälligkeit im Einsatz. Es gibt jeweils eine reduzierte, leicht zu interpretierende Ausgabe. Die Normierung auf einem gemeinsamen Skalar ermöglicht einen Vergleich der Ergebnisse zwischen den Metriken. Die Verständlichkeit der Metriken konkretisiert die abstrakte Aufgabe und das Embedding. Die Euklidische Distanz bietet einen absoluten Vergleich und die Kosinus-Ähnlichkeit einen abstrakten. Zusammen soll die gemeinsam geschaffene Perspektive ein all umfassenderes Verständnis der Embedding Ähnlichkeit liefern.

4 Analyse & Fazit

Das folgende Kapitel unterhält zwei Unterkapitel, Analyse und Auswertung. Die Analyse untersucht die Funktionalität der Umsetzung des im vorherigen Kapitel 3 vorgestellten Konzepts. Erstens wird untersucht, inwieweit das *Custom GraphSAGE Modell* das Embedding der Montagedaten erlernt. Zweitens bestimmt das Graph Matching Ähnlichkeiten von Sample Paaren. Es wird geschaut, was für Aussagen die Ähnlichkeiten zulassen. Die Auswertung beurteilt die in der Analyse gesammelten Erkenntnisse in Bezug auf die Fragestellung: Inwieweit das semantische Wissen über Montageprozesse die Fehlererkennung bei verwandten Produkten ermöglicht?

4.1 Analyse

Es folgt die Untersuchung und Bewertung des Knoten Embedding und dessen Eignung zur Ähnlichkeiten Erkennung. Die Entwicklungs- und Testumgebung umfasst: 12th Gen Intel(R) Core(TM) i5-12600KF (3.70 GHz), 32 GB RAM und RTX 3060 Ti (8 GB RAM). Eine TPU[3] mit $334,6$ GB stellt eine weitere Testumgebung dar. Sie ermöglichte während der Entwicklung eine unabhängige Funktionsprüfung der Implementierung. Darüber hinaus bot die Kapazität der TPU die Möglichkeit, das Training mit dem gesamten Datensatz auf frei wählbaren Hyperparametern Größen: Beliebig große Batches, Output und Hidden Layer.

Das Embedding des *Custom GraphSAGE Modell* bestimmt maßgeblich die Qualität des Graph Matching, da es das Fundament des Vergleichs darstellt. Es erlernt die Ähnlichkeiten und repräsentiert diese. Keine Label erschwert die Beurteilung erlernter Separation der Feature. Plots der einzelnen Knotentypen und des gesamten Embedding bieten eine Einschätzung der Separation. Der Vergleich über Epochen macht die erlernten Änderungen sichtbar.
Die Beurteilung der Ähnlichkeiten umfasst der zweite Teil der Analyse. Der Vergleich findet mit händisch ausgewählten Montageprozessen statt. Dabei werden aus allen Sampeln des Da-

[3]Umgebung von Google Colab

tensatzes die Sample Paare gebildet, wo die Vermutung hoher Ähnlichkeit besteht. Der Analyse Schwerpunkt veranlasst einen Wechsel der ursprünglichen *Custom GraphSAGE Modell* Training Strategie, aus Kapitel 3.6, vom Datensatz Split (train, test und val Set) zu Full Dataset Training. Im Full Dataset Training sieht ein Modell im Training alle Sample des Datensatzes. Das potenzielle nicht kennen einzelner Sample des *Custom GraphSAGE Modell*, könnte bei derer Wahl den Vergleich schwächen. Die Gegenüberstellung eines bereits aus dem Training bekannten und gänzlich unbekannten Samples führt zu einer Unausgewogenheit in der Bewertung. Um diesem Umstand zu verhindern, ist eine Anpassung der Trainingsstrategie erforderlich. In diesem Zusammenhang erweist sich das Full Dataset Training als vorteilhafter im Hinblick auf die Verbesserung des Modellverständnisses, während der Anspruch eines generalisierten Modells in diesem Anwendungsfall in den Hintergrund tritt.

Das „finale Embedding“, also das *Custom GraphSAGE Modell*, das für die Ähnlichkeiten Erkennung verwendet wird, wurde nach Full Dataset Training trainiert. Die Hyperparameter sind: *learning rate:* 0.00001, *batch size:* 16, *input dim:* $(-1, -1)$, *hidden dim:* 256, *output dim:* 64 und Anzahl der Layer: 4. Die Graphen 15 zeigen das Training Loss und den Silhouette Score.

Embedding Varianten Vergleich. Die Wahl der Hyperparameter, die zu dem „finalen Embedding“ führten, beruht auf der Trainingsperformance des Modells, gemessen an der Konvergenz des Loss. Alle Modell Embedding Varianten wurden unter denselben Bedingungen trainiert. Auf derselben Hardware, der Entwicklungsumgebung. Trainiert wurde auf der CPU nach dem Full Dataset Training. Der Datensatz war identisch. Es mündet in den fünf Embedding Modellvarianten: *Emb1*, *Emb2*, *Emb3*, *Emb4* und *finales Embedding*. Ihre Hyperparameter sind in Tabelle 1 gegenübergestellt. Ihr Loss und Silhouette Score in Abbildung 14. Bereits in den ersten fünf Epochen zeichnen sich die Unterschiede ab, eine große Learning Rate destabilisiert das Training und lässt das Loss von Beginn an stark steigen und schwanken. Ab Epoche Vier liegt der Wert von *emb4* außerhalb der dargestellten Achswerte. Eine kleinere Learning Rate in *emb1* lässt das Loss sinken und hält es stabil auf einem niedrigen Niveau. An Epoche Neun steigt es jedoch ebenfalls stark an. Auch eine größere Hidden Dim erscheint nicht hilfreich zur Stabilisierung des Trainings. *emb3* konvergiert frühzeitig mit Epoche Zwei auf ein niedriges Niveau, wie *emb1*, stagniert dann, bis es in Epoche Zwölf deutlich ansteigt. Die größere Hidden Dim bewirkte eine Verzögerung des Anstiegs des Loss, konnte es aber nicht verhindern. Keine der Hyperparameter konnte das Training langfristig stabilisieren, die Begrenzung der maximalen Distanz, wie weit zwei Knoten während des Trainings auseinandergezogen werden können, durch Einführung des Wertes $1e-12$ schon. Damit ist die maximale Distanz ein Wert nahe 1 und verhindert damit eine Überoptimierung. Das Loss des *emb2* konvergiert früh und bleibt auf einem niedrigen Niveau. Das *finale Embedding* übernimmt die Hyperparameter vom *emb2* und reduziert die Learning Rate um eine Kommastelle. Die Reduzierung der Learning Rate stabilisiert das Training, das Loss konvergiert langsamer und stagniert dann um ein niedriges Niveau.

Die Betrachtung des Silhouette Score vom *finalen Embedding* spiegelt das erwartete Verhalten wider. Beginnend in Epoche Eins mit einem Wert von 0.269, steigt er mit dem vorangehenden Fall des Loss über die Epochen, auf den durchschnittlichen Wert von 0.603 (Durchschnitt von Epoche 10 bis 25). Alle weiteren Varianten haben von Epoche Eins an einen höheren Silhouette Score und steigen in den ersten fünf bis sieben Epochen auf den durchschnittlichen Wert von 0.603 des *finalen Embedding* an, während das *finale Embedding* erst in Epoche Zehn nachzieht (Siehe durchschnittliche Silhouette Score in Tabelle 2). Die Vergleichbarkeit der durchschnittlichen Silhouette Scores ist bezüglich der variierenden Epochenanzahl pro Embedding über die sie bestimmt worden, mit Vorbehalt zu betrachten. Das Erreichen des gleichen Niveaus, trotz ihrer unterschiedlichen Lernfortschritte, besonders *emb4*, mit einem stark steigenden Loss, macht die Aussagekraft des Silhouette Score in dieser Form hinfällig. Er ist gelöst vom Training und spiegelt nicht den Lernfortschritt wider.
Die Ursache lässt drei Vermutungen technischen Ursprungs zu. Um die Laufzeit des Trainings um nicht mehr als wenige Minuten (circa fünf Minuten) zu verlängern und die verfügbaren Ressourcen dem eigentlichen Training vorzubehalten, berechnet der Silhouette Score sich über eine begrenzte Anzahl an Knoten pro Batch. Maximal 10000 Knoten pro Knotentyp werden pseudozufällig aus allen Knoten des Batch genommen. Die begrenzte Menge ist eine Annäherung, was die Aussagekraft für den gesamten Datensatz schwächt. Als gravierender wird der Einfluss des Implementierungsfehlers gewichtet, die Berechnung des Silhouette Score für eine Epoche erfolgte lediglich für den letzten Batch einer Epoche. Damit basiert die Einschätzung der Modellleistung auf einem Bruchteil des Datensatzes. Der Einfluss des dritten Punktes, die Clusteranzahl, wird als nicht tragend eingeschätzt, sondern als ein zweitrangiger Faktor, der die Repräsentativität der Metrik weiter steigern könnte. Der Silhouette Score bewertet die Zuordnung der Knoten eines Knotentyps auf drei Cluster. Die Vermutung, die Clusteranzahl ist mit drei zu gering gesetzt, die Zuordnung bzw. Aufteilung der Knoten passiert ohne großer Lernaufwand des Embedding Modells. So ergibt sich ein hoher Silhouette Score Grundwert. Darüber hinaus wird vermutet das die Verteilung der Knoten allein durch die Feature bereits reicht die Cluster Zuordnung von drei widerzuspiegeln und es bedarf eine höhere Clusteranzahl, um den erlernten Verteilungen gerecht zu werden. Die Überprüfung der Vermutungen als mögliche Ursache der fehlenden Silhouette Score Aussagekraft entzieht sich dem Ziel der Arbeit und wird an dieser Stelle nicht behandelt. Weitere Untersuchungen der Qualität des *finalen Embedding*, folgen im nächsten Abschnitt.

Tab. 1: Stellt Hyperparameter der Modell Embedding Varianten gegenüber.

Embedding	Learning Rate	Batch Size	Num Layer	Hidden Dim	Output Dim	Unterer Grenzwert	Oberer Grenzwert
Emb1	0.0001	16	4	256	64	1e-12	1
Emb2	0.0001	16	4	256	64	1e-12	1-(1e-12)
Emb3	0.0001	16	4	512	64	1e-12	1
Emb4	0.001	16	4	256	64	1e-12	1-(1e-12)
Finales Embedding	0.00001	16	4	256	64	1e-12	1-(1e-12)

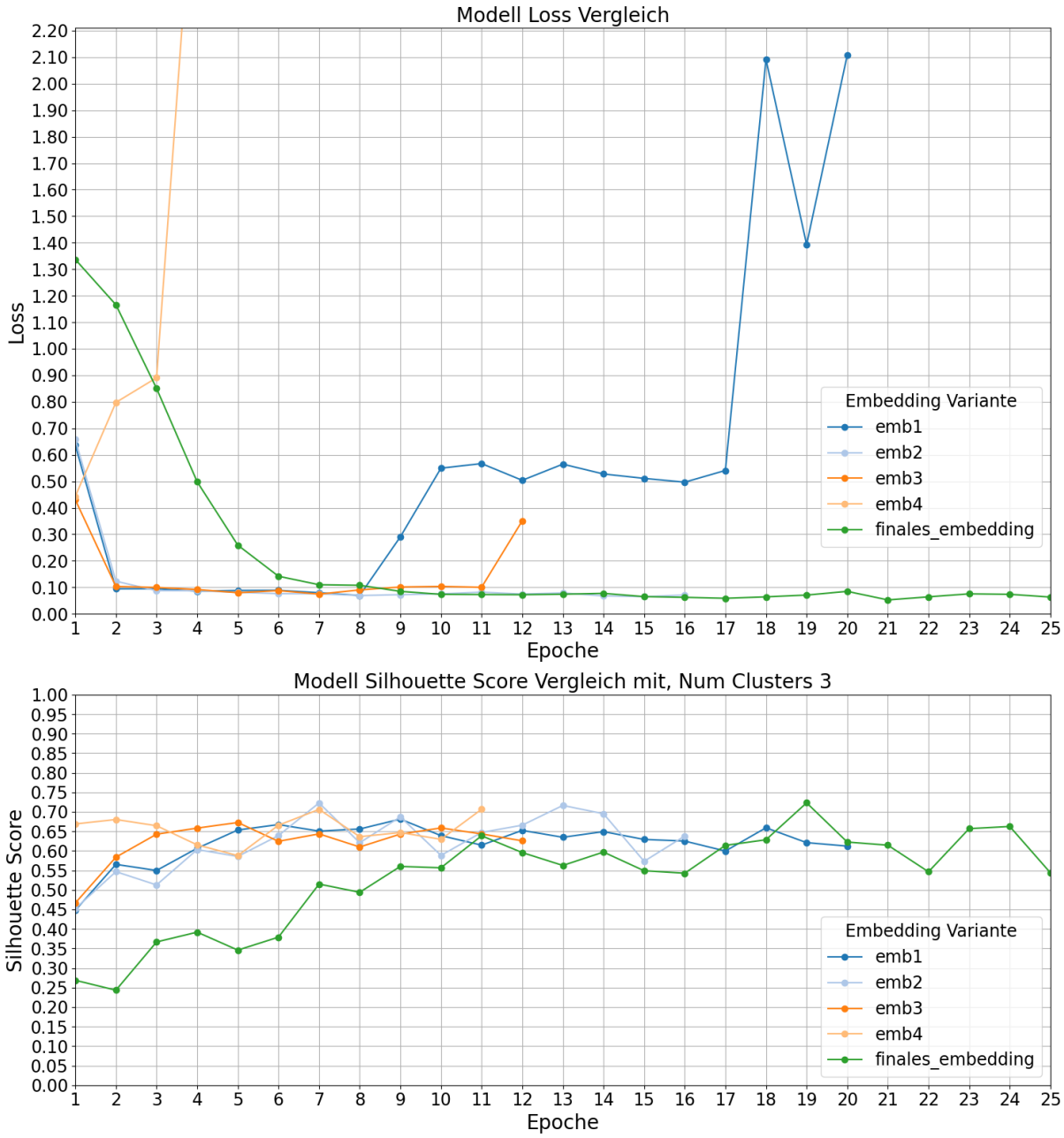


Abb. 14: Vergleich vom Loss und Silhouette Score der Modell Embedding Varianten.

Tab. 2: Der durchschnittliche Silhouette Score über alle Trainingsepochen für jede Embedding Variante. Die Vergleichbarkeit der durchschnittlichen Silhouette Scores ist mit Vorbehalt zu betrachten, da die Anzahl der Epochen pro Embedding variiert.

Embedding	Durchschnittlicher Silhouette Score
Emb1	0.621
Emb2	0.618
Emb3	0.623
Emb4	0.655
Finales Embedding	0.529

Finales Embedding. Das Training Loss (Abbildung 15) zeigt bis zur zehnten Epoche einen kontinuierlichen Lernfortschritt, ab da pendelt das Loss um den durchschnittlichen Wert von 0.069 (0.06899779863888397). Der durchschnittliche Wert ergibt sich aus Epoche zehn bis fünfundzwanzig.
Es deuten ein lernendes System an, das ab Epoche Zehn ein lokales Minimum erreicht. Weitere Evidenz dafür sollte die Betrachtung der Modellausgabe eines Sample über die Epochen zeigen. Die Vermutung, wenn das Modell lernt, sollte im Umkehrschluss das Embedding weiter zusammenrücken, da Ähnlichkeiten im Embeddingraum zusammengezogen werden. t-SNE stellt die Ausgabe eines Samples für fortlaufende Epochen dar. Der Vergleich der Plots sollte das Zusammenrücken des Embedding über die Epochen sichtbar machen und damit bestätigen, dass das Modell über die Epochen lernt.

Die Wahl des repräsentativen Sample für die Untersuchung erfolgte händisch. Sample *buPAOv1_MORE_EFFICIENT_v2_260* bietet eine nachvollziehbare Anzahl an *Processsteps* von 6257. Die Montage hat die *MAID* 2020599, sie erfolgt an fünf *Workstations*. Mit einer Größe von 17.808 KB liegt das Sample an der Spitze im letzten Drittel des Datensatzes.
Für jede ausgewählte Epoche wird ein t-SNE Plot des Samples erzeugt. Alle Knoten des Sample werden berücksichtigt. Die Reduktion der Dimensionen erfolgt auf zwei Dimensionen, die Berechnung läuft für 1000 Iterationen und es werden die 30 nächsten Nachbarn für die Berechnung verwendet. Für die Vergleichbarkeit der Plots über die Epochen wird der Seed bzw. Random State für eine einheitliche Initialisierung, auf den Wert 42 gesetzt. Verglichen werden die Embedding der Epochen 1, 5, 10, 15, 17, 19 und 25. Wenige Epochen bestätigen bereits den anfänglich starken Lernfortschritt. Ab Epoche 15 werden die Untersuchungsabstände kleiner, um potenzielle Lernfortschritte aufzuzeigen, die das Loss nicht widerspiegelt.

Die zwei Plots in Abbildung 16 zeigt die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der ersten und siebzehnten Trainingsepoche. Hervorgehoben ist das Embedding der *measure* Knoten. Knoten desselben Typs haben dieselbe Färbung. *mesaure* illustriert anschaulich das Lernen des Modells. Anfänglich liegen die Knoten grob in kleinere Cluster aufgeteilt, die nah bei einander liegen. Über die Epochen zieht das

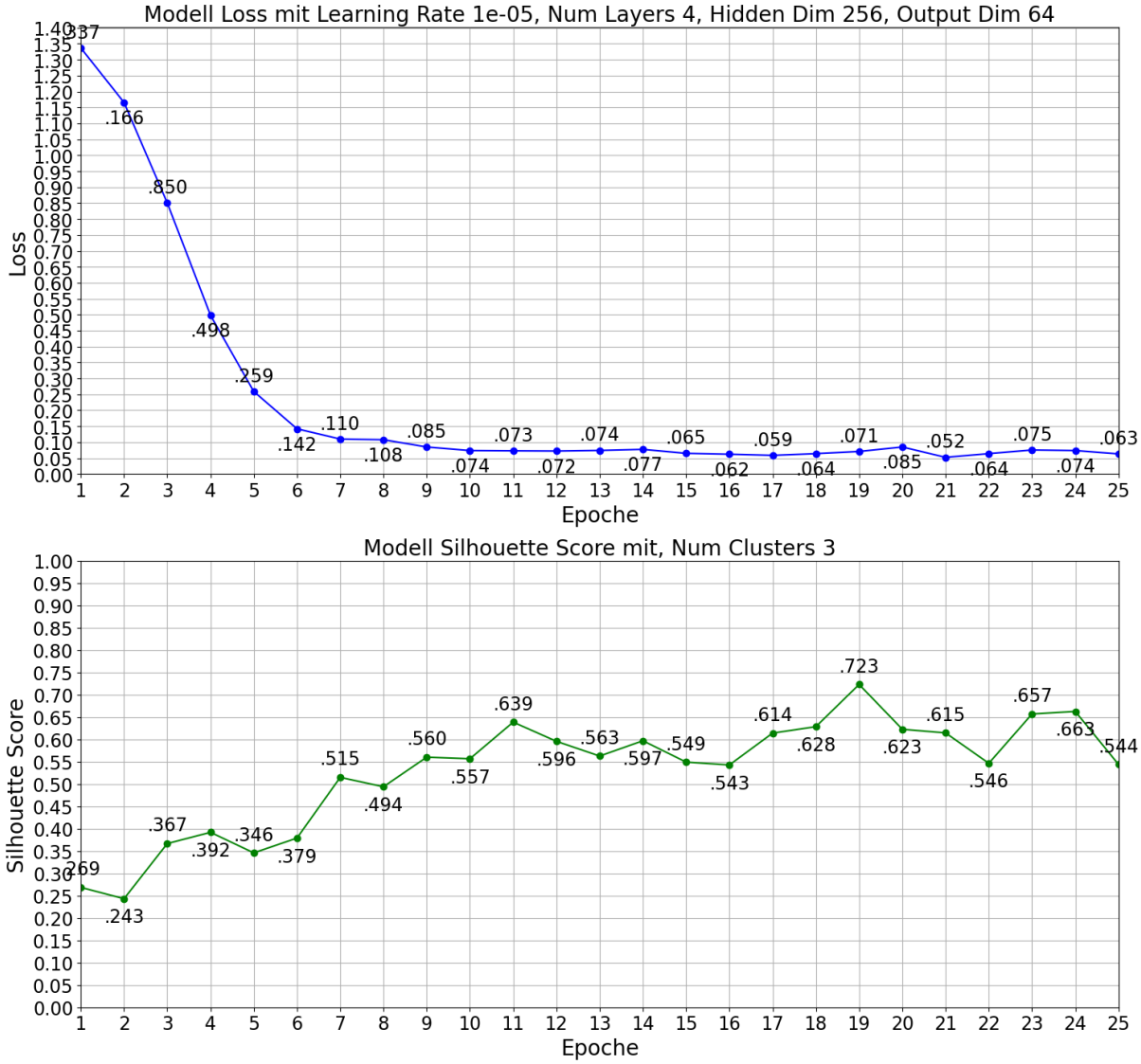


Abb. 15: Das Training Loss und der Silhouette Score pro Epoche des finalen Embedding Modells. Die Vorkommastellen der Loss und Silhouette Score Werte pro Epoche.

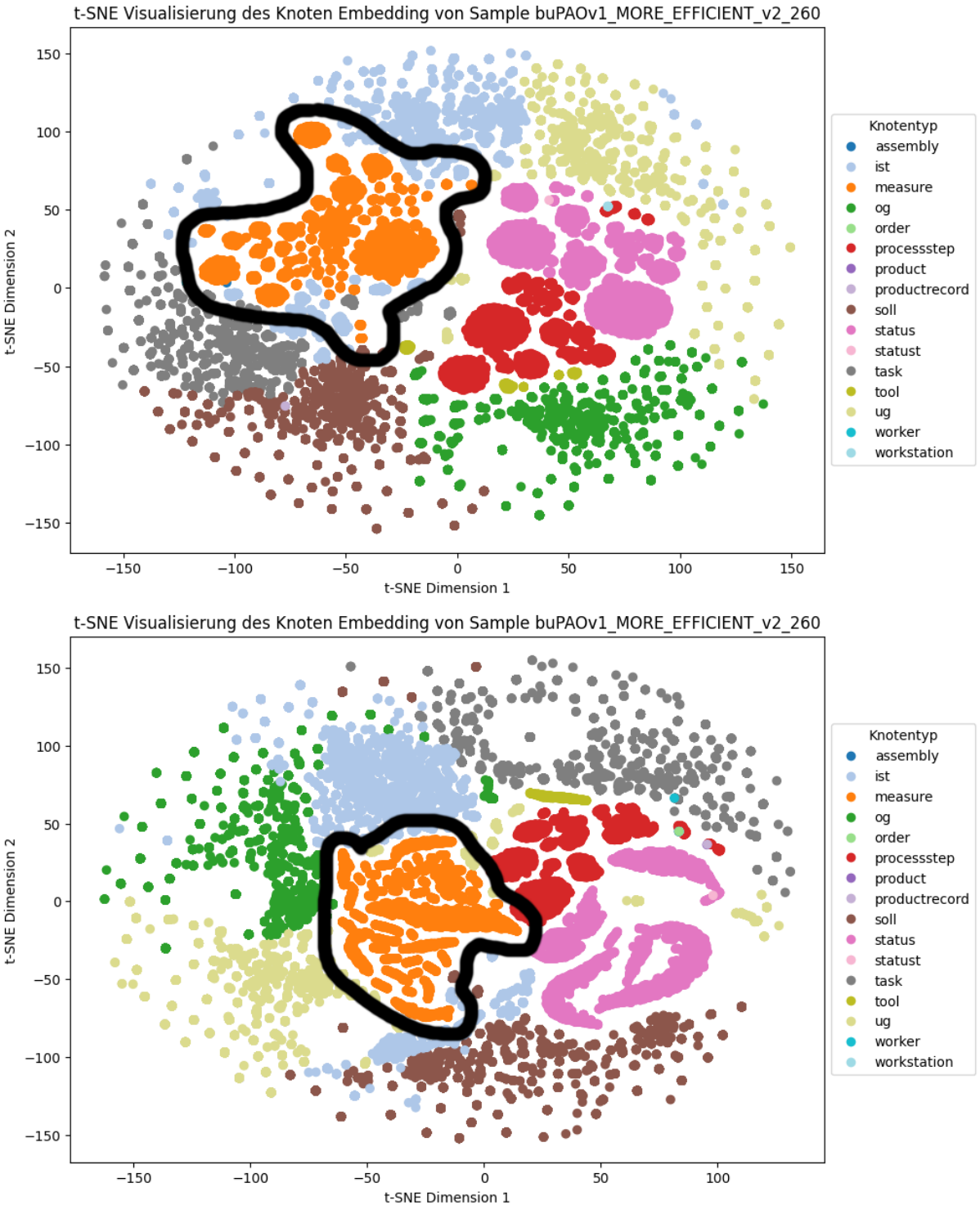


Abb. 16: Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der ersten und siebzehnten Trainingsepoche. Hervorgehoben ist das Embedding der *measure* Knoten.

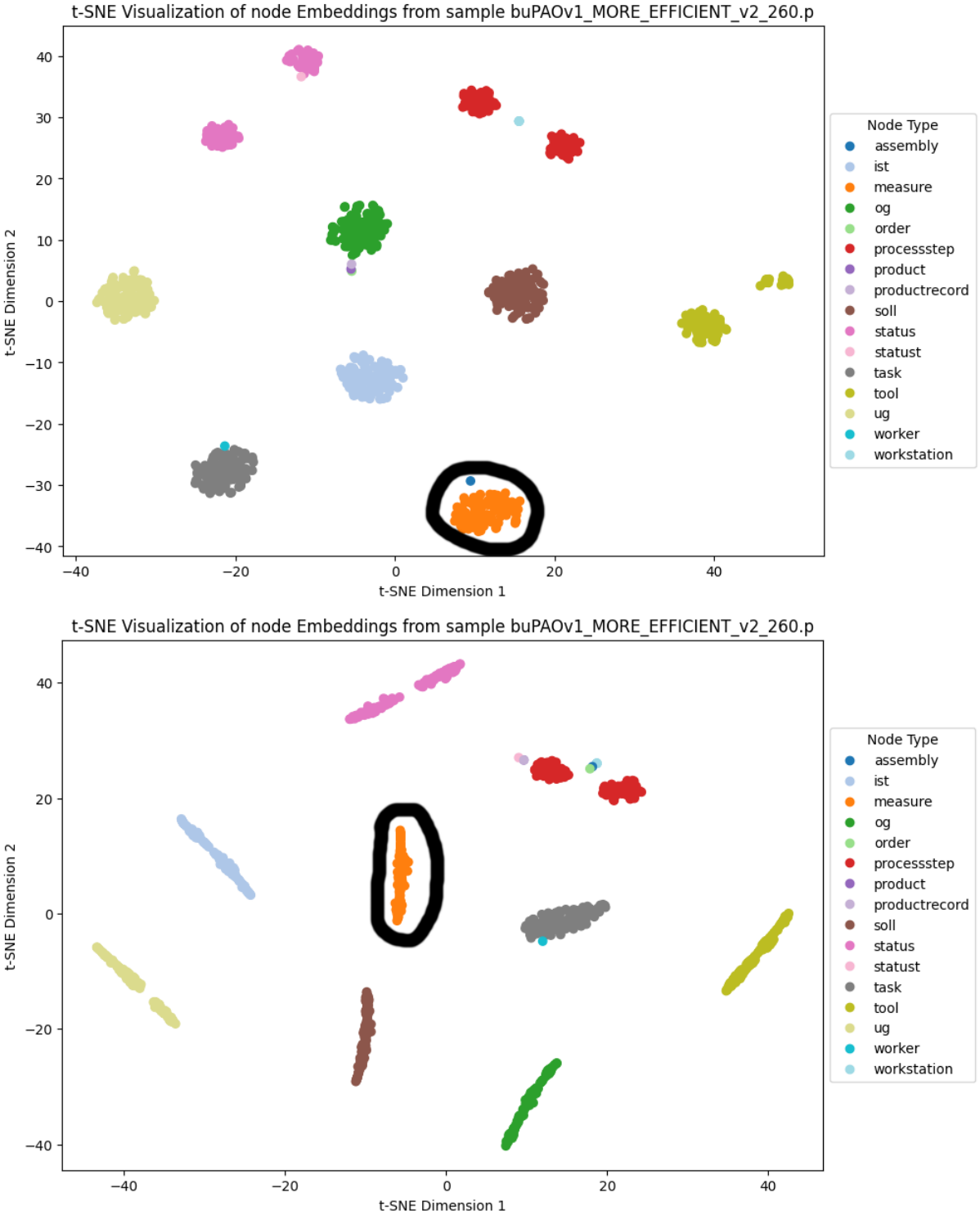


Abb. 17: Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der ersten und siebzehnten Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt. Hervorgehoben ist das Embedding der *measure* Knoten.

Modell die Knoten weiter räumlich zusammen. Ähnliche Knoten überlagern sich und bilden kleine längliche Cluster. Die Vermutung, das engere Zusammenrücken zeigt das die Knoten sehr ähnlich sind, bis auf wenige Eigenschaften. Der Unterschied der Eigenschaften drückt sich durch ihre lineare Anordnung aus. Die Visualisierung der Transformation der Cluster gibt weitere Bestätigung für das Lernen des Modells.
Noch deutlicher zeigt Abbildung 17 die erlernte Repräsentation der Knoten. Es zeigt den Output desselben Samples, in der ersten und siebzehnten Trainingsepoche. Die gezeigten Knoten wurden drastisch auf die ersten 100 pro Knotentyp begrenzt. Die *measure* Knoten liegen als ein Cluster vor und nach siebzehn Trainingsepochen embedded das Modell die Knoten kompakt als eine Art lineare Funktion. Die Embedding Plots der ausgelassenen Epochen befinden sich im Anhang A.7 und A.8.

Ähnlichkeiten Erkennung. Zur Einschätzung der Funktionalität des Siamese Modell wurden händisch Sample ausgewählt und verglichen. Der Vergleich der Sample mit sich selbst ergeben trivialerweise eine Ähnlichkeit von 1.0. Zwei unterschiedliche Sample zeigten die Grenzen der Metrik Implementierung auf. Die Variation der Anzahl an Knoten zwischen Samplen ergab die Notwendigkeit der Anpassung des Vergleichs. Ein einfacher Vergleich der Mengen der Knoten pro Knotentyp ist somit nicht möglich. Eine Mittlung über alle Knoten eines Knotentyps reduziert sie auf einen Wert und ermöglicht Vergleichbarkeit. Dagegen spricht, die Mittlung reduziert die semantische Aussagekraft. Alternativ, das zufällige Bilden von Paaren, deren Distanzen verglichen werden. Die Zufälligkeit verhindert die Reproduzierbarkeit für wiederholte Vergleiche. Stattdessen zeigt die Wahl und Paarung der ähnlichsten Knoten zwischen den Sampeln keine der vorher genannten Einschränkungen. Der Vergleich von Teilmengen macht es zu einer Art „Semantischem Subgraph Matching“. Die Ergebnisse fünf ausgewählter Sample Paare und der Vergleich desselben Sample *buPAOv1_MORE_EFFICIENT_v2_22* sind in Tabelle 3 aufgeführt.
Alle Kosinus-Ähnlichkeiten und euklidische Distanzen zeigen hohe Ähnlichkeitswerte. Dies ist auf die Domäne bedingte Ähnlichkeit der Montagedaten in der Struktur und Worten zurückzuführen. Zusätzlich fand der Vergleich gezielt auf den Paaren mit der größten Ähnlichkeit statt. Unterschiede sind feingranular. Mit einer übergreifenden Ähnlichkeit von $0,99999$ (abgerundet) sticht die Kosinus-Ähnlichkeit gegenüber der Euklidischen Distanz heraus. Die Vermutung liegt nah, dass die Kosinus-Ähnlichkeit seine Stärke auf den Sampeln nicht ausspielen kann, da die Grenzwerte für eine Unterscheidung zu groß und die der Knoten zu klein sind. Diese Erkenntnis macht die Kosinus-Ähnlichkeit für diesen Anwendungsfall unbrauchbar und wird in Folge für die weitere Analyse nicht mehr aufgeführt.

Tab. 3: Ähnlichkeiten fünf händisch gewählter Sample Paare. Und der Vergleich des buPAOv1_MORE_EFFICIENT_v2_22 mit sich selbst. Aus Übersichtsgründen wurden die Sample-Namen auf dessen Nummer gekürzt. Ergänze *buPAOv1_MORE_EFFICIENT_v2_* als Präfix.

Sample Eins	Sample Zwei	Kosinus-Ähnlichkeit	Euklidische Distanz
113	12	0.9999957084655762	0.9972022771835327
830	512	0.9999969005584717	0.994002640247345
999	12	0.9999948740005493	0.9990066289901733
3	1111	0.999998152256012	0.9994323253631592
1	5	0.9999952912330627	0.9994302988052368
22	22	1.0	1.0

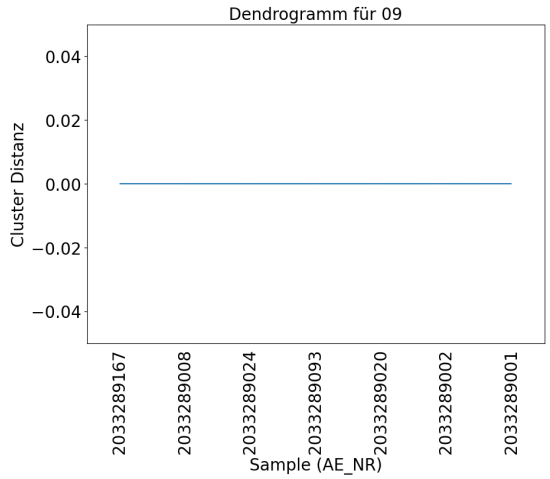
Nach der vorherigen technischen Demonstration folgt die gezielte Untersuchung hinsichtlich der These. Dafür werden Montagevarianten desselben Produkts auf ihre Ähnlichkeit verglichen. Die Ermittlung geeigneter Montageprozesse für die Analyse stützt sich auf eine interne, nicht veröffentlichte praktische Analyse der Montagedaten vom BIBA (Bremer Institut für Produktion und Logistik GmbH), durchgeführt von Dirk Schweers. Sie ermittelt vorhan-

dene Montageprozesse und ordnet sie nach der *MA_NR*. Anschließend werden alle Montageschritte eines Montageprozesses konkateniert und gehashed. Das Hashen dient neben der vereinfachten Handhabung des Formates dazu, gleiche Montageanleitungen des Datensatzes in der Analyse nur einmal zu verwenden. Die Dimension der gehashten Montageprozesse von den Varianten eines Produktes werden mit Principal Component Analysis kurz PCA reduziert und in Vektoren überführt. Es bewirkt eine Reduzierung der Varianz der Montageprozesse gleicher Klassen und die Maximierung der Unterschied der Montageprozesse verschiedener Klassen. Anschließend bestimmt eine Analyse mithilfe von Silhouette Score und Agglomeratives Clustering die optimale Clusteranzahl für die Varianten. Die Varianten stellen Versionen der Montageprozesse da. Eine Klasse fasst eine Version der Montageprozesse zusammen. Als mögliche Clusteranzahlen wurden Werte im Bereich von Eins bis 220 berücksichtigt.
Die Analyse ermittelte elf Produkte, die gefertigte Varianten unterhielten und sie von einer bis 204 unterschiedliche Cluster unterteilt. Die Varianten wurden entsprechend mit Agglomerative Clustering in die vorab bestimmte optimale Anzahl an Cluster aufgeteilt. In Tabelle 4 sind die Produkte unter ihrer jeweiligen *VARIANTE_NUMMER* aufgeführt, zusammen mit der Anzahl der zugeordneten Cluster.
Ausgehend von der Analyse wird für jedes Produkt ein Paar an Varianten bestimmt, die auf ihre Ähnlichkeit verglichen werden. Die Paar-Bestimmung erfolgt über die Distanzberechnung zwischen den Clustern. Stellvertretend dafür werden die Distanzen zwischen den zuvor gewählten Repräsentanten der Cluster berechnet. Die erste zugeordnete Montageprozess eines Clusters in der Clustering-Ausgabe ist der Repräsentant eines Clusters. Die Repräsentanten Paare mit der maximalen Distanz werden auf Ähnlichkeit verglichen.
Die Montageprozesse der Produkte mit der *VARIANTE_NUMMER* 09 und 10 teilen ein Cluster und fallen damit aus der vorher beschrieben Paarbildung, die mindestens zwei Cluster benötigt. Neben der Clusterung, half die Erzeugung von Dendrogrammen für jede *VARIANTE_NUMMER* bei dem Verständnis der Beziehung der Varianten. Das Dendrogramm zeigt die Distanz der Produktvarianten. Es beruft sich nicht auf das vorherige Clustering, sondern erzeugt ein eigenes hierarchisches Agglomerative Clustering. Jedes montierte Produkt wird anfänglich als ein eigenständiges Cluster angesehen. Sie werden nicht auf die optimale Clusteranzahl, sondern auf ein gemeinsames Cluster reduziert. Für 09 sind die Montageprozesse identisch, sie fallen für die Paarbildung und Analyse raus (Abbildung 18a). 10 besteht aus zwei Montageprozessen. Sie zeigen eine Distanz (Abbildung 18b). Der Fall, dass die Analyse für die Montageprozesse die optimale Clusterung von einem Cluster bestimmt hat. Und das Dendrogramm zeigt, dass es sich um genau zwei Montageprozesse handelt, weist als Untersuchungsgegenstand zu wenig Signifikanz auf.

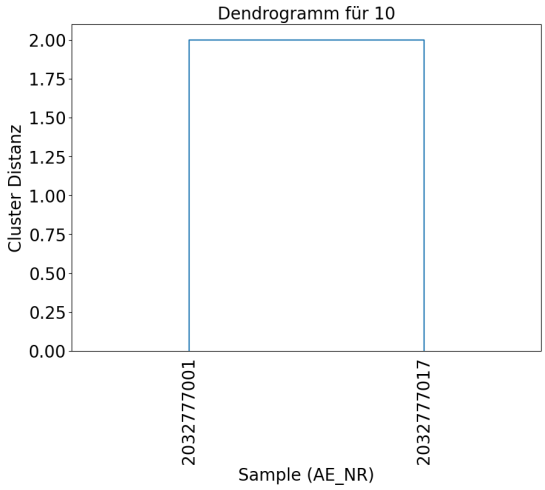
Nachdem die Analyse der Montageprozesse die maximal entfernten Varianten Paare eines Produktes bestimmt hat, bewertet das Siamese Modell dessen Sample auf Ähnlichkeiten. Erwartet wird eine hohe Ähnlichkeit, begründet durch Ihre Beschaffenheit, demselben Produkt zu entspringen. Die Bewertung der Paarung unterhält Tabelle 5.

Tab. 4: Die mit der Analyse aus den Montagedaten bestimmten Produkte und die optimale Anzahl an Clustern für die Varianten. Die *MA_NR* der Repräsentanten und Ihre Sample-Nummer. Aus Übersichtsgründen wurden die Sample-Namen auf dessen Nummer gekürzt. Ergänze *buPAOv1_MORE_EFFICIENT_v2_* als Präfix.

VARIANTE_NUMMER	Optimale Cluster Anzahl	Repräsentant Eins	Repräsentant Zwei	Sample Eins	Sample Zwei
01	2	2028454	2031289	618	783
02	204	2032313	2030122	936	678
03	2	2032753	2020969	1034	462
04	2	2018646	2030074	238	677
05	22	2020439	2024896	253	428
06	2	2027142	2032554	602	1106
07	4	2018270	2029038	221	648
08	56	2027619	2030319	595	750
09	1	-	-	-	-
10	1	-	-	-	-
11	6	2031716	2033012	845	1052



(a) Die Varianten zeigen keine signifikanten Unterschiede.



(b) Die montierten Varianten zeigen einen Unterschied.

Abb. 18: Die Dendrogramm Plots der Produkte 09 und 10. Sie zeigen die Distanz der gefertigten Produkt Varianten.

Die Ergebnisse bestätigen die Vermutung. Alle bis auf das letzte Paar haben eine 99.999 prozentige Übereinstimmung. Das stellvertretende Plotten der *task* Knoten des Sample Paar *buPAOv1_MORE_EFFICIENT_v2_618* (Obere Abbildung) und *buPAOv1_MORE_EFFICIENT_v2_783* (Untere Abbildung), illustriert die Ergebnisse. Die Strukturen sind ähnlich, es lassen sich übergreifend Muster erkennen. Vier hervorstehende, wurden in den Plots markiert (Abbildung 19).

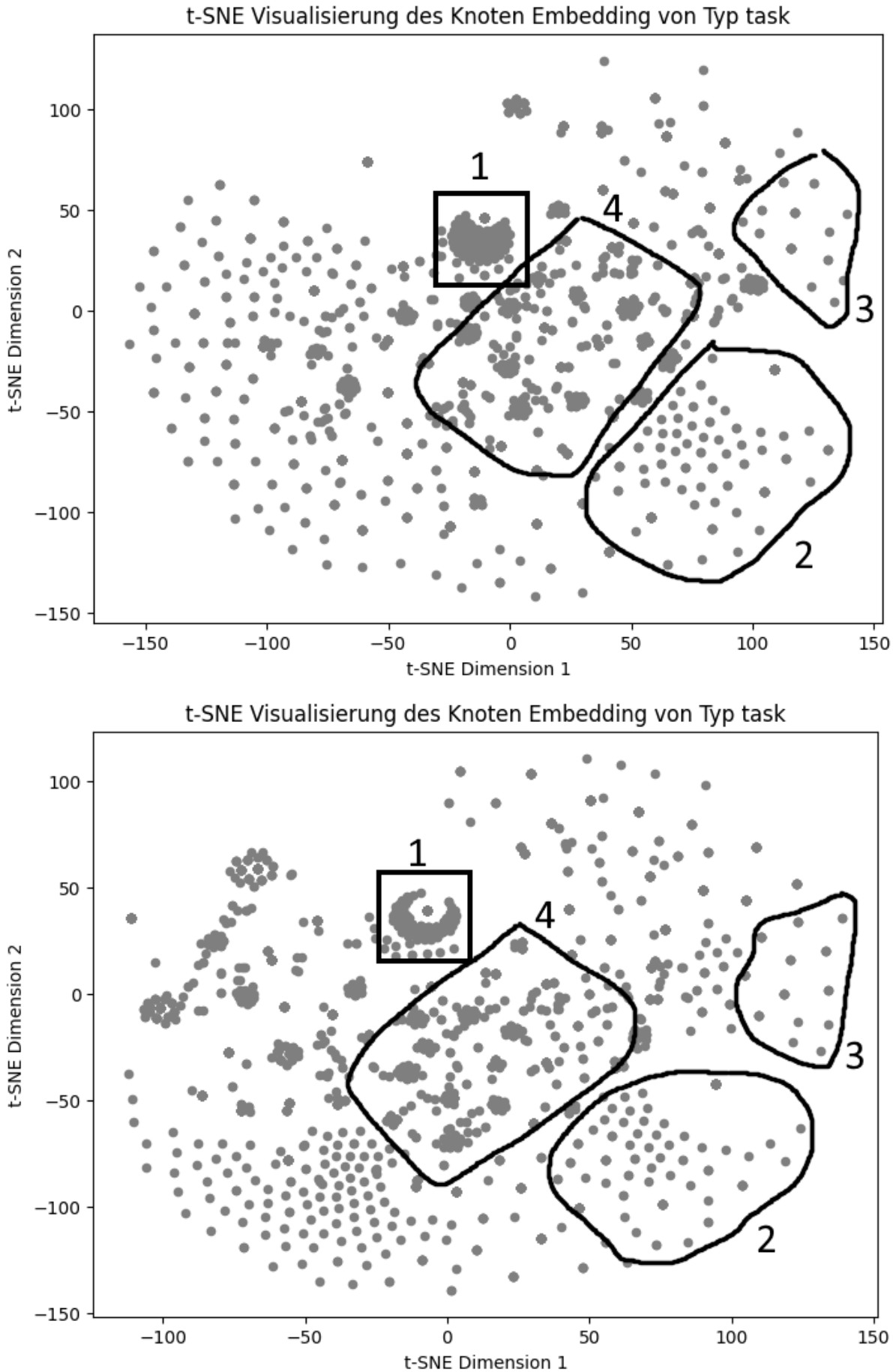


Abb. 19: Zeigt das Embedding der *task* Knoten, der Sample *buPAOv1_MORE_EFFICIENT_v2_618* (Obere Abbildung) und *buPAOv1_MORE_EFFICIENT_v2_783* (Untere Abbildung). Herausstechend ähnliche Teile wurden übergreifend nummeriert.

Tab. 5: Ähnlichkeiten der durch die Analyse bestimmten Sample-Paare. Aus Übersichtsgründen wurden die Sample-Namen auf dessen Nummer gekürzt. Ergänze *buPAOv1_MORE_EFFICIENT_v2_* als Präfix.

Sample Eins	Sample Zwei	Kosinus-Ähnlichkeit	Euklidische Distanz
618	783	-	0.9996063709259033
936	678	-	0.9995277523994446
1034	462	-	0.9992935061454773
238	677	-	0.9995816349983215
253	428	-	0.9995154142379761
602	1106	-	0.9992775321006775
221	648	-	0.9993839859962463
595	750	-	0.9996150135993958
845	1052	-	0.9988863468170166

Doch wie gestaltet sich die Ähnlichkeit zwischen verschiedenen Produkten, wenn sie keine Varianten voneinander sind? Um diese Frage zu beantworten, wird für jedes, durch die Analyse ermittelte Produkt, eine Variante als Repräsentant gewählt. Wie bereits demonstriert unterscheiden sich die Varianten eines Produktes mit 0.999 (gerundet) sich erst ab der vierten Nachkommastelle. Sie sind damit so identisch, dass jeder von ihnen als Repräsentant in Frage kommt. Entsprechend wird die erste Variante jedes Produktes als Repräsentant genommen.
Sie werden hierarchisch geclustered, das Dendrogramm 20 zeigt die Diskrepanz der Montageprozesse der Repräsentanten. Am interessantesten erscheinen die unähnlichsten, also die den größten Abstand im Dendrogramm aufweisen. Dies sind Montageprozesse der Produkte aus den orangen Cluster zu Montageprozessen der grünen und roten Cluster. Auch die Distanz zwischen den grünen und roten Clustern erscheint signifikant. Aus allen wird ein Repräsentant gewählt. Ihre Sample werden untereinander auf Ähnlichkeit geprüft. Die Ergebnisse unterhält Tabelle 6. Die Paarungen haben alle eine ermittelte Ähnlichkeit von 99.8 Prozent (gerundet).

Tab. 6: Ähnlichkeiten gewählter Repräsentanten unterschiedlicher Produkte. Aus Übersichtsgründen wurden die Sample-Namen auf dessen Nummer gekürzt. Ergänze *buPAOv1_MORE_EFFICIENT_v2_* als Präfix.

Repräsentant Eins	Repräsentant Zwei	Sample Eins	Sample Zwei	Kosinus Ähnlichkeit	Euklidische Distanz
1026732055	2027142001	513	602	-	0.9977644085884094
1026732055	2033289002	513	1053	-	0.9978015422821045
2027142001	2033289002	602	1053	-	0.997979462146759

Die Untersuchung zeigt, Varianten eines Produktes haben 99.9 prozentige Ähnlichkeit und unterscheiden sich in der zweiten Nachkommastelle. Hingegen Produkte, die keinem gemeinsamen entspringen, sind zu 99 Prozent ähnlich und unterscheiden sich in der ersten Nachkommastelle. Somit lässt das Graph Matching für den Vergleich zweier Montageprozesse die

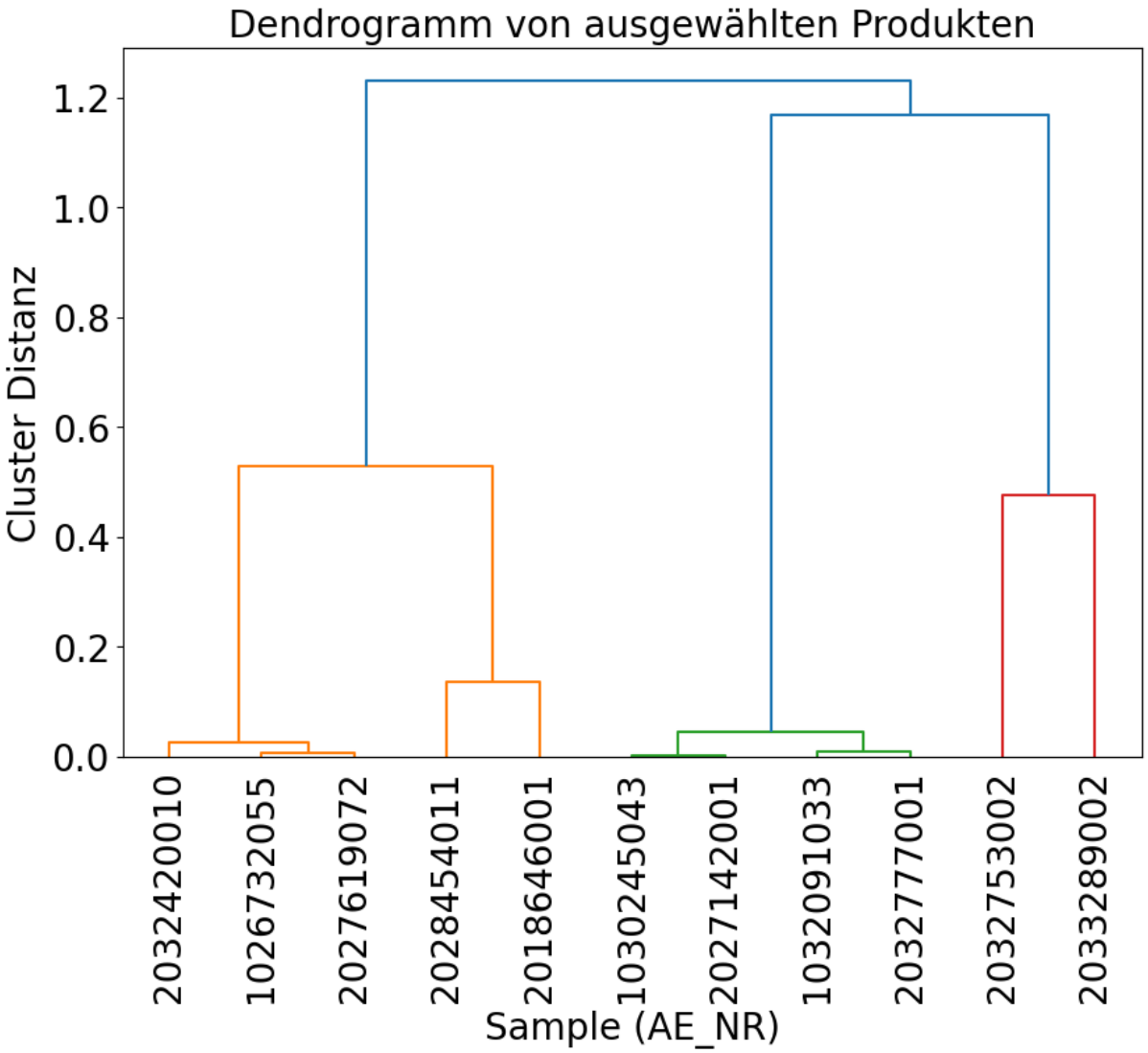


Abb. 20: Hierarchisches Clustering ausgewählter Montageprozesse, dargestellt als Dendrogramm.

Aussage zu, ob sie Produktvarianten oder fremd sind. Ohne dessen *VARIANTEN_NUMMER* zu kennen. Inwieweit die gesammelten Erkenntnisse bei der Beantwortung der Fragestellung helfen, folgt im nächsten Kapitel 4.2.

4.2 Fazit

Die Arbeit untersucht: Montageprozesse semantisch in Beziehung zu setzen, hilft bei der Erkennung von Montagefehlern. Die Problemstellung der übergreifenden Fehlererkennung wurde praktisch auf die Erkennung von Ähnlichkeiten heruntergebrochen. Die Vermutung, Fehlererkennung bei verwandten Produkten ist nichts weiter als die Erkennung von Ähnlichkeiten. Da der erkannte Fehler als ein Muster in den Montagedaten gesehen werden kann. Wenn ein ähnliches Muster bei verwandten Montagedaten gefunden wird, ist die Wahrscheinlichkeit der Anfälligkeit des Fehlers auch bei der Montage dieses Produkts gegeben.
Die praktische Umsetzung macht sich dabei die Beziehung der Montagedaten zunutze. Bisher wurden die Montagedaten einzeln betrachtet, doch die Zusammenführung bzw. Vergleich gibt gewünschte Aussagen der Ähnlichkeit. Das Vorgehen mündet heruntergebrochen in drei Schritten. Erstens Montagedaten einer Montage semantisch in Beziehung setzen. Zweitens ein gemeinsames Embedding der Montagedaten erlernen. Und abschließend die Montagedaten von Produkten untereinander auf Ähnlichkeiten vergleichen.
Die Montagedaten wurden wie in Kapitel 3.1 beschrieben, nach der entworfenen Ontologie *buPAOv1* semantisch in Beziehung gesetzt. Das *Custom GraphSAGE Modell* erlernte das Embedding der Montagedaten. Plots über die Epochen bestätigten den Lernprozess des Modells. Das Siamese Modell vergleicht Montagedaten auf Ähnlichkeiten. Die Vergleiche ermöglichen eine Aussage, ob es sich um Varianten eines gemeinsamen Produktes handelt oder die Montage unterschiedlicher Produkte. Varianten unterscheiden sich in der vierten Nachkommastelle und unterschiedliche Produkte in der dritten Nachkommastelle. Mit einem übergreifenden Wert von 0.99 (abgerundet), haben sie eine hohe Ähnlichkeit. Die Ergebnisse bestätigen die grundlegende Eigenschaft der hohen vermuteten Ähnlichkeit der Daten der Montagedomäne. Die Evidenz bestätigt die Funktionalität des Systems und zeigt, dass das *custom GraphSAGE Modell* erfolgreich Ähnlichkeiten zusammenführt. Zusammen mit dem Siamese Modell wurde eine Anwendung geschaffen, die den Vergleich von Montageprozessen als Wissensgraphen ermöglicht. Dennoch macht die praktische Umsetzung die Schwächen und Limitierungen des Konzepts deutlich. Insbesondere besteht eine Diskrepanz zwischen den Ergebnissen zur Ähnlichkeit und der Übertragbarkeit von erkannten Montagefehlern eines Produkts auf die Montage eines anderen Produkts, was die Aussagekraft des Systems in diesem Kontext einschränkt. Doch wie sehen Lösungen für die erkannte Schwäche aus?

Anstatt die Ähnlichkeiten auf einen Wert zu reduzieren, wird eine Aussage für jeden Knotentyp ausgegeben. Die Heterogenität der Daten ermöglicht dies. Diese Teilung schlüsselt die Zusammensetzung des gesamten Ähnlichkeitswerts für das Graphen-Paar auf. Weiteren Einblick gibt die Rekonstruktion der Graphen. Die ähnlichsten gewählten Knotenpaare werden einheitlich hervorgehoben. Es lässt das Graphen paar auf Muster untersuchen. Wo gemeinsame Knoten jeweils in den Graphen stehen und wie sie im Verhältnis zueinander stehen. Doch eins wurde deutlich. Die Ergänzungen steigern vermeintlich die Aussagekraft und das Verständnis der Ähnlichkeit, doch ohne zusätzliche Informationen gelingt keine Einordnung.

Eine Form von Supervision wird benötigt. Label, die erkanntem Bedeutung geben. Dafür ist als die Vorarbeit die Beschaffung der Label vonnöten. Bei der Montage werden erkannte Fehler mit dem Montageassistenzsystem aufgenommen. Entsprechend wird dem Montageprozess eine Fehlerklasse zugewiesen oder dem expliziten Montageschritt, indem der Fehler auftritt. Das Unsupervised Konzept kann mit den zusätzlichen Informationen nicht umgehen und muss entsprechend ersetzt werden. Ein End-to-End Deep Graph Modell erlernt die Klassifikation für die Montagegraphen. Entweder wird dem gesamten Montagegraph eine Fehlerklasse zugewiesen oder ein oder mehrere Montageschritte werden markiert, wo das Modell einen Fehler vermutet. Die Klassifikation kann weiterhin auf dem Wissensgraph-Datensatz stattfinden und den Vorteil der semantischen Beziehungen der Daten nutzen, sowie die Integration weiterer Daten. Im Fall der Klassifikation einzelner Montageschritte bedarf eine Erweiterung der Label. Anstelle der Fehlerklassenzuordnung des gesamten Graphen müssen explizit die relevanten Montageschritte erkennbar werden. Die Verwendung des Wissensgraph-Datensatz fordert zusätzlich eine Übersetzung der erkannten Montageschritte auf die Knoten der Wissensgraphen.

Das schließt das Fazit. Nach der Zusammenfassung der Arbeit folgt der Ausblick, im nächsten Kapitel 5. Dieser führt die Diskussion fort und zeigt mögliche Richtungen zur Fortsetzung der Forschung auf.

5 Zusammenfassung & Ausblick

Die Zusammenfassung fasst die wesentlichen praktischen Aspekte der Arbeit zusammen. Es beschreibt die erzeugte Ontologie *buPAOv1*, entstanden aus den Montagedaten. Die Anwendung des Convolution Graph Embedding Modell, GraphSAGE. Abschließend folgen die Erkenntnisse des Ähnlichkeiten-Vergleichs mit dem Siamese Modell.
Der Ausblick führt die Diskussion über die weiteren möglichen Forschungsausrichtungen fort. Es beinhaltet die Steigerung der Aussagekraft der Metriken und den Vergleich der Umsetzung mit Varianten und alternativen Modellarchitekturen.

5.1 Zusammenfassung

Die Montage stellt einen zentralen Punkt der Produktion von Produkten dar. Die sich ändernden Kundenbedürfnisse bewirken eine steigende Produktvielfalt bei kürzeren Lebenszyklen der Produkte und ein damit einhergehend steigender Kostendruck. Veranlasst die Produzenten, nach Wegen der Montagekostensenkung zu suchen. Das Nutzen von Daten, besonders im frühzeitigen Erkennen von Montagefehlern, gewinnt an Bedeutung. Die Untersuchung der Montageprozessstruktur und dessen Vergleich, gibt Aufschlüsse über die Montage. Die Aussagekraft ist jedoch begrenzt und mündet in einem NP Problem. Weitere nützliche Daten der Produktion und Beziehungen der Daten untereinander werden außen vor gelassen. Doch mit Deep Graph Learning tut sich eine neue Möglichkeit auf.

Das Ziel ist die Demonstration eines praktischen Deep Learning Systems, das Erkenntnisse zur Fehlererkennung auf echten Montagedaten zulässt. Indem die Montagedaten semantisch in Beziehung gesetzt und anschließend auf Ähnlichkeiten verglichen werden.
Die Problemstellung der übergreifenden Fehlererkennung wurde praktisch auf die Erkennung von Ähnlichkeiten heruntergebrochen. Die Vermutung, Fehlererkennung bei verwandten Produkten ist nichts weiter als die Erkennung von Ähnlichkeiten. Da der erkannte Fehler als nichts weiter als ein Muster, in den Montagedaten gesehen werden kann. Wenn ein ähnliches Muster bei verwandten Montagedaten gefunden wird, ist die Wahrscheinlichkeit der Anfäl-

ligkeit des Fehlers auch bei der Montage dieses Produkts gegeben.

Die Umsetzung gliedert sich in drei Schritte. Allem vorweg, ein Vorverarbeitungsschritt. Die Montagedaten unterhalten Montageprozesse von Hydraulikaggregaten. Sie bestehen aus Montageschritten, die sequenziell ausgeführt werden, erkennbar an dem Zeitstempel, der Zeitpunkt, an dem das Assistenzsystem die Durchführung des Schrittes gemessen hat. Gebündelt wird die Montage eines Produktes unter dessen Auftragsnummer.
Auf Basis der Montagedaten entsteht die Ontologie *buPAOv1* (BIBA-Uni-Prueger-Assembly-Ontology- Version-1), die die Daten in Beziehung setzt. Genutzt wird der *RDF* Standard (Resource Description Framework) des World Wide Web Consortium (*W3C*) (W3C RDF Working Group, 2014). Jeder Auftrag in den Montagedaten wird mit dem eigens entwickelten Tool namens *Data Handler* nach der Ontologie *buPAOv1* in Beziehung gesetzt. Es entstehen Wissensgraphen in der Form der Ontologie. Der entstandene Datensatz trainiert das an heterogene Graphen angepasste GraphSAGE (Hamilton et al., 2018) Modell, genannt *Custom GraphSAGE Modell*. Es lernt Unsupervised das Knoten Embedding der Wissensgraphen. Damit dies möglich wurde, ergab sich die Notwendigkeit, die Wissensgraphen in ein für das Modell verständliches Format zu bringen. Die textuellen Feature der Knoten wurden in Vektoren überführt. Ein mit den Begriffen der Montagedaten trainiertes Word2Vector Modell übersetzt die Feature. Die Struktur der Graphen bildet das *HeteroData* Format ab, ein Dict ähnliche Struktur. Die Kanten sind Index-Paare die auf die Feature der verbundenen Knoten zeigen.
Der Vergleich auf Ähnlichkeit zweier Sample, führt das Siamese Modell aus. Zwei ausgewählte Sample werden mit dem *Custom GraphSAGE Modell* embedded und ihr Embedding wird auf Ähnlichkeiten verglichen. Pro Knotentyp werden zwischen den Graphen die ähnlichsten Knotenpaare bestimmt. Die Ähnlichkeit wird über dessen Distanz im Embeddingraum bestimmt, je näher, desto ähnlicher sind sich die Knoten. Die zwei Metriken Kosinus-Ähnlichkeit und Euklidische Distanz berechnen separat die Distanzen der Knotenpaare. Die Ausgabe ist ein Ähnlichkeitswert der zwei Sample zwischen 0.0 und 1.0. 0.0 bedeutet, sie haben keine Ähnlichkeit und 1.0 sie sind identisch.

Das System demonstriert erfolgreich, wie die semantische Bedeutung der Montagedaten explizit eingebunden werden kann, indem sie in Beziehung gesetzt werden. Und die Ähnlichkeitsbestimmung auf echten Montagedaten. Die Bewertung des Systems erfolgte in zwei Schritten. Die Embedding Qualität des Modells wurde untersucht. Es wurde geschaut, was und wie das Modell die Features repräsentiert. Dafür wurden *t-SNE* Plots der Embedding über die Trainingsepochen auf ihren Lernfortschritt verglichen. Zweitens wurden die Ergebnisse der erzeugten Ähnlichkeiten des Siamese Modells untersucht und hinsichtlich der Aussagekraft der Fehlererkennung bewertet. Nach der Demonstration der Funktionalität durch den Vergleich beliebiger Sample, wurden gezielt ausgewählte Sample verglichen. Die Auswahl der Sample erfolgte über ein vorheriges Clustering, die Montageschritte wurden gehashed und auf der Ebene verglichen und geclustered. Erkannt wurden elf montierte Produkte und ihre

jeweiligen Varianten. Das Siamese Modell verglich die Wissensgraphen der Varianten untereinander und der nach der Analyse bestimmten unterschiedlichen Produkte.
Die Analyse ergab, Varianten desselben Produktes sind ähnlicher, als Montageprozesse unterschiedlicher Produkte. Sie lässt entsprechend die Aussage zu, ob zwei verglichene Montageprozesse Varianten desselben oder unterschiedlicher Produkte sind. Es zeigt aber auch deutlich die Limitierung auf, ein Ähnlichkeitswert hat eine Diskrepanz zur Fehlererkennung, es lässt keine übertragende Aussage zu. Die Zuordnung fehlt. Es benötigt eine Form von Supervision, Fehlerklassen, zur Klassifizierung.
Vorschläge zur Fortführung der Forschung folgen im nächsten Kapitel 5.2.

5.2 Ausblick

Die weitere Forschung kann sich auf zwei Schwerpunkte konzentrieren: die Erweiterung der Datenbasis und die Verbesserung der Fehlererkennung. Eine Erweiterung der *buPAOv1*-Ontologie würde das Verständnis des Modells vertiefen. Insbesondere könnte der *Processstep* präziser beschrieben werden, indem die beteiligten Bauteile für jeden *Processstep* explizit aufgeführt werden. Die Daten stammen aus CAD (Computer-Aided Design) Daten, die die geometrische Beziehung der Bauteile an dem beteiligten *Processstep* darstellen (Chen et al., 2020). Es eröffnet die Möglichkeit der Schaffung von Fehlerklassen. Anhand der neu gewonnen Informationen können vorab für die *Processstep*, physikalische Ober- und Untergrenzen bestimmen, ab dem die Belastung zu hoch oder Verbindung zu lose ist und einen Fehler verursacht. Entsprechend lernt das Modell den Zusammenhang der physikalischen Grenzen und der vom Assistenzsystem gemessenen Ausführung der *Processstep*, festgehalten in *measure*, *og*, *ug*, *ist* und *soll*.
Auf der Seite der Fehlererkennung und Embedding Modell, bietet sich die Korrektur der Bewertungsmetrik Silhouette Score an. Der in der Analyse erkannte Implementierungsfehler korrigieren. Anstatt die Berechnung des Silhouette Score für jede Epoche durch den alleinigen letzten Batch der Epoche, wird jeder Batch einbezogen und über die Scores gemittelt.
Die weitere Steigerung der Aussagekraft wird erreicht, indem die Cluster Anzahl dynamisch an die Feature Vielfalt des jeweiligen Knotentyps angepasst werden. Knoten wie *Worker* haben im Datensatz genau einen Knoten und können entsprechend nicht in der selben Cluster Anzahl zugeordnet werden wie die Knoten vom *Processstep*, mit einer Vielzahl an Feature Variation. Ausgehend von der Anzahl an Feature des jeweiligen Knotentyps könnten Cluster Abstufungen bestimmt, beispielsweise 5, 10 und 15 Prozent und dann die Modellperformanz gegenüberstellen.
Neben der Steigerung der Aussagekraft der Metriken, bietet sich das Finden einer mehr performanten Embedding Modell Abwandlung an. Das GraphSAGE Modell (Hamilton et al., 2018) bietet neben *mean*, weitere Aggregatoren an, *pooling* und *LSTM*, die Nachbarfeatures anders einbeziehen. Die Überführung des heterogenen Datensatzes in homogene Daten. Erlernt es von dem heterogenen Modell nicht erreichbare Muster? Die Erweiterung des negativen Sampling um eine Gewichtung. Die Entfernung bestimmt den Einfluss des negativ bestimmten Ziel-

knotens. Ein denkbares Resultat wäre eine feingranularer Embedding Unterteilung der Knoten.
Die Arbeit demonstriert das Konzept praktisch, die weitere Forschung kann darüber hinaus gehen. Der Vergleich mit anderen Deep Learning Architekturen. Der Stand der Technik vergleicht in Kapitel 2.4 die Architekturen auf ihren Nutzen für diesen Anwendungsfall. Neben dem in der Arbeit umgesetzten Convolutional, zwei weitere, Auto-Encoder und Translational. Als geeignet werden die Modelle *HGATE* (Wang et al., 2023b) und *CapsE* (Nguyen et al., 2019) benannt.

Literaturverzeichnis

I. Amazon Web Services. *Amazon Neptune Documentation*, 2023. URL https://docs.aws.amazon.com/neptune/.

Y. Bai, D. Xu, Y. Sun, and W. Wang. Glsearch: Maximum common subgraph detection via learning to search, 2021. URL https://arxiv.org/abs/2002.03129.

Z. Chen, B. Jinsong, Z. Xiaohu, and T. Liu. An assembly information model based on knowledge graph. *Journal of Shanghai Jiaotong University (Science)*, 25, 04 2020. DOI 10.1007/s12204-020-2179-y.

ELAM. Montageanleitung kraftstoffpumpe, 2020. Nicht öffentlich zugänglich, interne Dokumentation.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, Feb. 2022. ISSN 2162-2388. DOI 10.1109/tnnls.2021.3070843. URL http://dx.doi.org/10.1109/TNNLS.2021.3070843.

N. Jiang, B. Ning, and J. Dong. A survey of gnn-based graph similarity learning. In *2023 8th International Conference on Image, Vision and Computing (ICIVC)*, pages 650–654, 2023. DOI 10.1109/ICIVC58118.2023.10269885.

D. Keiser, C. Petzoldt, V. Walura, S. Leimbrink, and M. Freitag. Concept and integration of knowledge management in assembly assistance systems. *Procedia CIRP*, 118:940–945, 2023. ISSN 2212-8271. DOI https://doi.org/10.1016/j.procir.2023.06.162. 16th CIRP Conference on Intelligent Computation in Manufacturing Engineering.

K.-Y. Kim, D. G. Manley, and H. Yang. Ontology-based assembly design and information sharing for collaborative product development. *Computer-Aided Design*, 38(12):1233–1250, 2006. ISSN 0010-4485. DOI https://doi.org/10.1016/j.cad.2006.08.004. URL https://www.sciencedirect.com/science/article/pii/S0010448506001680.

S. Lemaignan, A. Siadat, J.-Y. Dantan, and A. Semenenko. Mason: A proposal for an ontology of manufacturing domain. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, pages 195–200, 2006. DOI 10.1109/DIS.2006.48.

H. Li, X. Wang, Z. Zhang, and W. Zhu. Out-of-distribution generalization on graphs: A survey, 2022. URL https://arxiv.org/abs/2202.07987.

M. Liu, B. Zhou, J. Li, X. Li, and J. Bao. A knowledge graph-based approach for assembly sequence recommendations for wind turbines. *Machines*, 11(10), 2023a. ISSN 2075-1702. DOI 10.3390/machines11100930. URL https://www.mdpi.com/2075-1702/11/10/930.

X. Liu, L. Mian, Y. Dong, F. Zhang, J. Zhang, J. Tang, P. Zhang, J. Gong, and K. Wang. Oag_{know} know : Self-supervised learning for linking knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):1895–1908, 2023b. DOI 10.1109/TKDE.2021.3090830.

N. Lohse, G. Valtchanov, S. Ratchev, M. Onori, and J. Barata. Towards a unified assembly system design ontology using protégé. 07 2005.

I. Neo4j. *Neo4j Documentation*, 2023. URL https://neo4j.com/docs/.

D. Q. Nguyen, T. Vu, T. D. Nguyen, D. Q. Nguyen, and D. Phung. A capsule network-based embedding model for knowledge graph completion and search personalization, 2019. URL https://arxiv.org/abs/1808.04122.

Ontotext. *GraphDB Documentation*, 2023. URL https://www.ontotext.com/products/graphdb/.

OpenCypher. *Cypher Query Language Documentation*, 2023. URL https://opencypher.org/.

PyTorch-Geometric-Team. *PyTorch Geometric Tutorial: Heterogeneous Graphs*, 2023. URL https://pytorch-geometric.readthedocs.io/en/latest/tutorial/heterogeneous.html. Zugriff am: 15. November 2024.

Rex, Ying, Z. Lou, J. You, C. Wen, A. Canedo, and J. Leskovec. Neural subgraph matching, 2020. URL https://arxiv.org/abs/2007.03092.

P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

Y. Sun, G. Li, J. Du, B. Ning, and H. Chen. A subgraph matching algorithm based on subgraph index for knowledge graph. *Front. Comput. Sci.*, 16(3), June 2022. ISSN 2095-2228. DOI 10.1007/s11704-020-0360-y. URL https://doi.org/10.1007/s11704-020-0360-y.

N. T. Tam, H. T. Trung, H. Yin, T. Van Vinh, D. Sakong, B. Zheng, and N. Q. V. Hung. Entity alignment for knowledge graphs with multi-order convolutional networks (extended abstract). In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2323–2324, 2021. DOI 10.1109/ICDE51399.2021.00247.

S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks, 2020. URL https://arxiv.org/abs/1911.03082.

W3C RDF Working Group. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts/, 2014. URL https://www.w3.org/TR/rdf11-concepts/. World Wide Web Consortium (W3C) Recommendation.

W3C RDF Working Group. RDF Primer. https://w3c.github.io/rdf-primer/spec/, 2023. URL https://w3c.github.io/rdf-primer/spec/. World Wide Web Consortium (W3C) Working Draft.

W3C RDF Working Group. RDF 1.2 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf12-concepts/, 2024. URL https://www.w3.org/TR/rdf12-concepts/. World Wide Web Consortium (W3C) Recommendation.

W3C SPARQL Working Group. SPARQL 1.1 Query Language, 2013. URL https://www.w3.org/TR/sparql11-query/. World Wide Web Consortium (W3C) Recommendation.

J. Wang, X. Qu, J. Bai, Z. Li, J. Zhang, and J. Gao. Sages: Scalable attributed graph embedding with sampling for unsupervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):5216–5229, 2023a. DOI 10.1109/TKDE.2022.3148272.

W. Wang, X. Suo, X. Wei, B. Wang, H. Wang, H.-N. Dai, and X. Zhang. Hgate: Heterogeneous graph attention auto-encoders. *IEEE Transactions on Knowledge and Data Engineering*, 35 (4):3938–3951, 2023b. DOI 10.1109/TKDE.2021.3138788.

X. Wang, Q. Zhang, D. Guo, and X. Zhao. A survey of continuous subgraph matching for dynamic graphs. *Knowl. Inf. Syst.*, 65(3):945–989, Oct. 2022. ISSN 0219-1377. DOI 10.1007/s10115-022-01753-x. URL https://doi.org/10.1007/s10115-022-01753-x.

World Wide Web Consortium (W3C). OWL 2 Web Ontology Language: Conformance, 2012. URL https://www.w3.org/TR/owl2-conformance/. W3C Recommendation, Zugriff am: 15. Dezember 2024.

M. Yahya, J. G. Breslin, and M. I. Ali. Semantic web and knowledge graphs for industry 4.0. *Applied Sciences*, 11(11), 2021. ISSN 2076-3417. DOI 10.3390/app11115110. URL https://www.mdpi.com/2076-3417/11/11/5110.

Q. Yan, J. Fan, M. Li, G. Qu, and Y. Xiao. A survey on knowledge graph embedding. In *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*, pages 576–583, 2022. DOI 10.1109/DSC55868.2022.00086.

Abbildungsverzeichnis

Tabellenverzeichnis

Universität Bremen

Urheberrechtliche Erklärung

für schriftliche Arbeiten

Bei Abschlussarbeiten ist diese Erklärung verpflichtend in jedes Exemplar der Arbeit einzubinden.

Nachname: Vorname/n:

Matrikelnummer:

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Alle Stellen, die ich wörtlich oder sinngemäß aus anderen Werken entnommen habe, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

Datum: Unterschrift:

Einverständniserklärung

zur elektronischen Überprüfung der eingereichten Arbeit auf Plagiate

Eingereichte Arbeiten können mit der Software Plagscan auf einen hauseigenen Server auf Übereinstimmung mit externen Quellen und der institutionseigenen Datenbank untersucht werden.

Zum Zweck des Abgleichs mit zukünftig zu überprüfenden Studien- und Prüfungsarbeiten kann die Arbeit dauerhaft in der institutionseigenen Datenbank der Universität Bremen gespeichert werden.

Mit meiner Unterschrift erkläre ich mich einverstanden,

dass die von mir vorgelegte und verfasste Arbeit zum Zweck der Überprüfung auf Plagiate auf den Plagscan-Server der Universität Bremen hochgeladen wird.

dass die von mir vorgelegte und verfasste Arbeit zum o.g. Zweck dauerhaft auf dem Plagscan-Server der Universität Bremen gespeichert wird.

Datum: Unterschrift:

Hinweis

Im Jahr 2019 wird die Software zunächst in einigen Fachbereichen eingesetzt. Dieser Abschnitt ist daher nur für Studierende der Fachbereiche 04, 07, 08, 10 und 11 zu berücksichtigen.

Das Einverständnis mit der Überprüfung durch die Plagiatsoftware und der dauerhaften Speicherung des Textes ist freiwillig. Die Einwilligung kann jederzeit durch Erklärung gegenüber der Universität Bremen mit Wirkung für die Zukunft widerrufen werden.

Weitere Informationen zur Überprüfung von schriftlichen Arbeiten durch die Plagiatsoftware sind im Nutzungs- und Datenschutzkonzept enthalten.

.....

Erklärung zur Veröffentlichung von Abschlussarbeiten

Abschlussarbeiten werden zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten.

Archiviert werden:
1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

Bitte kreuzen Sie Zutreffendes an:

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach frühestens 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum: Unterschrift:

A Anhang

A.1 Ontologie Beispiel Code

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
         xmlns:ex="http://www.demo.ontologie.com/beziehung-von-menschen/v1#
            ">

    <!-- Definition der Klassen -->
    <owl:Class rdf:about="ex:Person"/>
    <owl:Class rdf:about="ex:Erwachsener">
        <rdfs:subClassOf rdf:resource="ex:Person"/>
    </owl:Class>
    <owl:Class rdf:about="ex:Kind">
        <rdfs:subClassOf rdf:resource="ex:Person"/>
    </owl:Class>

    <!-- Definition der Eigenschaften -->
    <owl:DatatypeProperty rdf:about="ex:istJahreAlt">
        <rdfs:domain rdf:resource="ex:Person"/>
        <rdfs:range rdf:resource="xsd:integer"/>
    </owl:DatatypeProperty>

    <owl:ObjectProperty rdf:about="ex:verheiratetMit">
        <rdfs:domain rdf:resource="ex:Erwachsener"/>
        <rdfs:range rdf:resource="ex:Erwachsener"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="ex:geschiedenVon">
        <rdfs:domain rdf:resource="ex:Erwachsener"/>
        <rdfs:range rdf:resource="ex:Erwachsener"/>
```

```
31      </owl:ObjectProperty>
32
33      <owl:ObjectProperty rdf:about="ex:elternteilVon">
34          <rdfs:domain rdf:resource="ex:Erwachsener"/>
35          <rdfs:range rdf:resource="ex:Kind"/>
36      </owl:ObjectProperty>
37  </rdf:RDF>
```

Programm A.1: Beziehung von Menschen Ontologie im RDF Format. Die Beschreibung nutzt zusätzlich den von W3C entworfenen Standard OWL. OWL ist eine Ergänzung des RDF Standards (World Wide Web Consortium (W3C) (2012)).

A.2 Wissensgraph Beispiel Code

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
         xmlns:ex="http://www.demo.ontologie.com/beziehung-von-menschen/v1#
            ">

    <!-- Kinder -->
    <ex:Kind rdf:about="ex:Bob">
        <ex:istJahreAlt rdf:datatype="xsd:integer">9</ex:istJahreAlt>
    </ex:Kind>

    <ex:Kind rdf:about="ex:Eliza">
        <ex:istJahreAlt rdf:datatype="xsd:integer">5</ex:istJahreAlt>
    </ex:Kind>

    <ex:Kind rdf:about="ex:Eva">
        <ex:istJahreAlt rdf:datatype="xsd:integer">11</ex:istJahreAlt>
    </ex:Kind>

    <!-- Erwachsene -->
    <ex:Erwachsener rdf:about="ex:Tim">
        <ex:istJahreAlt rdf:datatype="xsd:integer">45</ex:istJahreAlt>
        <ex:geschiedenVon rdf:resource="ex:Pam"/>
    </ex:Erwachsener>

    <ex:Erwachsener rdf:about="ex:Jim">
        <ex:istJahreAlt rdf:datatype="xsd:integer">40</ex:istJahreAlt>
        <ex:verheiratetMit rdf:resource="ex:Pam"/>
    </ex:Erwachsener>

    <ex:Erwachsener rdf:about="ex:Pam">
        <ex:istJahreAlt rdf:datatype="xsd:integer">42</ex:istJahreAlt>
        <ex:verheiratetMit rdf:resource="ex:Jim"/>
        <ex:geschiedenVon rdf:resource="ex:Tim"/>
    </ex:Erwachsener>

    <!-- Eltern-Kind-Beziehungen -->
    <ex:Erwachsener rdf:about="ex:Jim">
        <ex:elternteilVon rdf:resource="ex:Bob"/>
        <ex:elternteilVon rdf:resource="ex:Eliza"/>
    </ex:Erwachsener>

    <ex:Erwachsener rdf:about="ex:Pam">
        <ex:elternteilVon rdf:resource="ex:Bob"/>
```

```
46          <ex:elternteilVon rdf:resource="ex:Eliza"/>
47          <ex:elternteilVon rdf:resource="ex:Eva"/>
48      </ex:Erwachsener>
49
50      <ex:Erwachsener rdf:about="ex:Tim">
51          <ex:elternteilVon rdf:resource="ex:Eva"/>
52      </ex:Erwachsener>
53
54  </rdf:RDF>
```

Programm A.2: Wissensgraph Instanz der Beziehung von Menschen Ontologie im RDF Format

A.3 Überblick der Montagedaten

Tab. 7: Einträge-Beschreibung der Montagedaten. Default führt Repräsentanten jeder Spalte auf.

Eintrag	Typ	Default	Beschreibung
ID	Integer	0	Aufzählung der Reihen.
PLA_ID	Integer	0	Produktlebenszyklus-ID. Beschreibt in welcher Phase sich das Produkt befindet.
VARIANTE	String	„1111 / AA.01“	Bestehend aus VARIANTE_NUMMER / VARIANTE_NAME Beschreibt es eindeutig eine Variante eines Produktes
VARIANTE_NUMMER	Integer	1111	Eindeutige ID zur Identifizierung einer Produktvariante
VARIANTE_NAME	String	„AA.01“	Varianten Bezeichnung
MA_NR	Number String	2016786	Eindeutige Auftrags-ID
AE_NR	Number String	2016786076	Eindeutige Produkt-ID
BAND	String	„Linie_01“	Bezeichnung der Fertigungsstraße auf dem der Arbeitsschritt ausgeführt wird
SEQUENZNR	Integer	61120	Fortlaufende Nummer gefertigter Produkte
BEREICH	String	„Station_01“	Arbeitsstation
MESSSTELLE	String	„Presse_01“	Werkzeug Bezeichnung
MAUFNEHMER	String	„Digi_01“	Werkzeug Anschluss oder Einstellung
MESSZEIT	Timestamp	„2020_09_14 _09_15_29_00“	Zeitstempel an dem die im Arbeitsschritt beteiligte Maschine eine Messung absolviert
MESSART	String	„Abschluss“	Der Typ des absolvierten Messwertes
PROGRAMM	String	„Minimum X“	Prozesschritt Konfiguration
BEDEUTUNG	String	„Messwert“	Zeitpunkt der Erhebung der Messung im Prozessschritt
TAFNAME	String	„Bohrungen oelen“	Beschreibung der Aufgabe des Prozessschrittes
STATUS	Integer	1	Bewertung des Prozessschrittes, Repräsentiert als Nummer
STATUSTEXT	String	IO	Beschreibung der Bewertung des Prozessschrittes
UG	Float/String	33,3	Der geringste Wert der die Aufgabe annehmen darf
IST	Float/String	1877.36	Der Wert, der durch die Aufgabe erzeugt wurde
OG	Float/String	4,5	Der höchste Wert der die Aufgabe annehmen darf
SOLL	Float/String	34,5	Optimaler Wert. Der Wert, der die Aufgabe annehmen soll
EINHEIT	String	„mm“	Die betrachtete Einheit des gemessenen Wertes
WERKER	String	„“	Die am Prozessschritt beteiligte Person
GELOESCHT	Interger	0	—

A.4 Montageanleitung der Kraftstoffpumpe

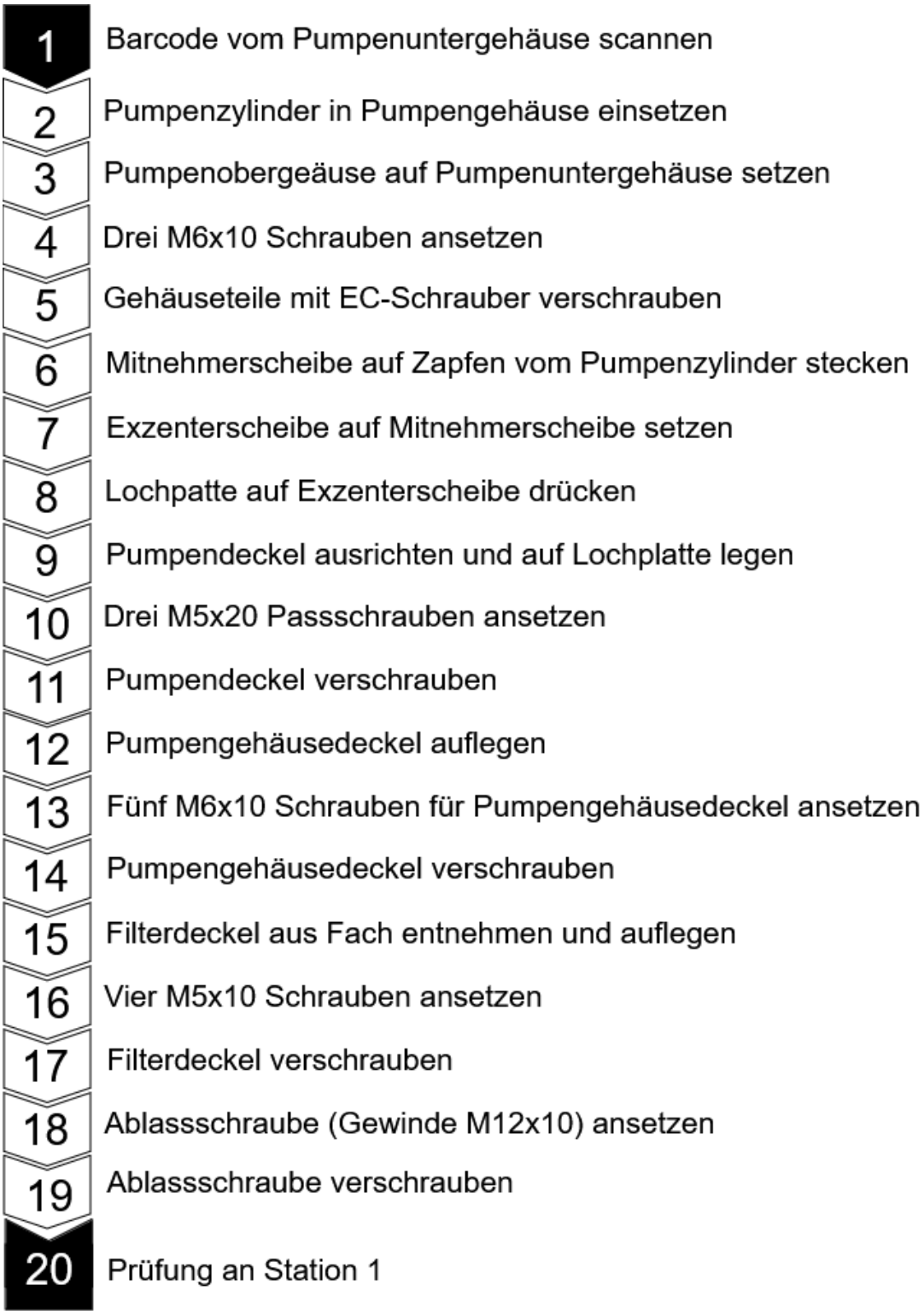
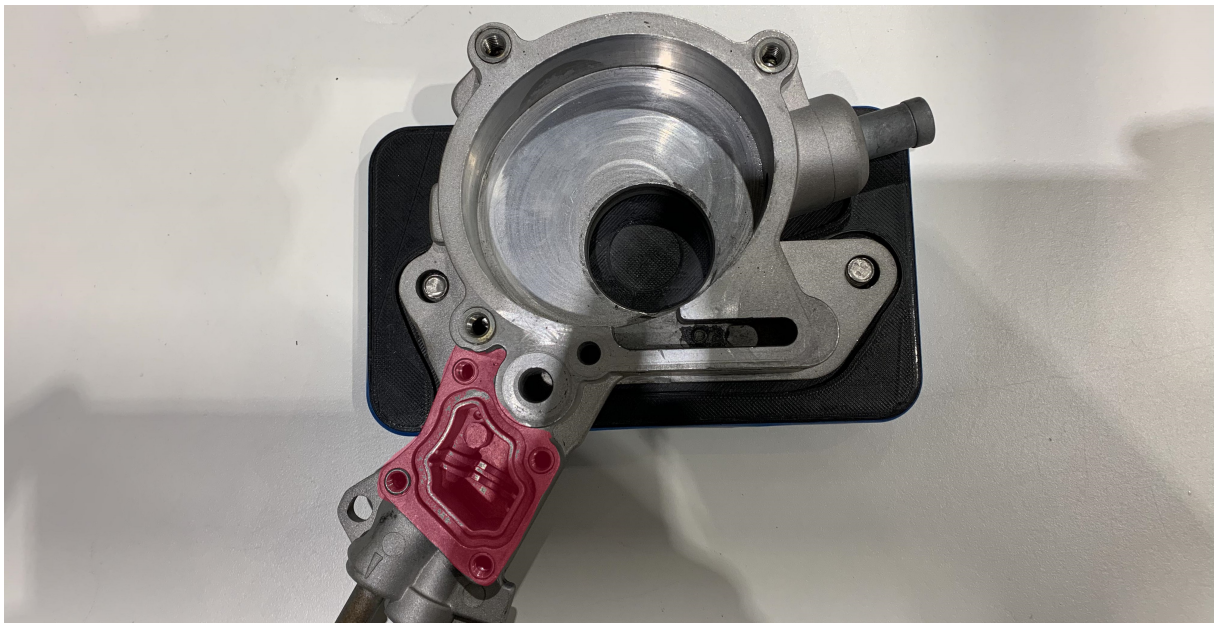
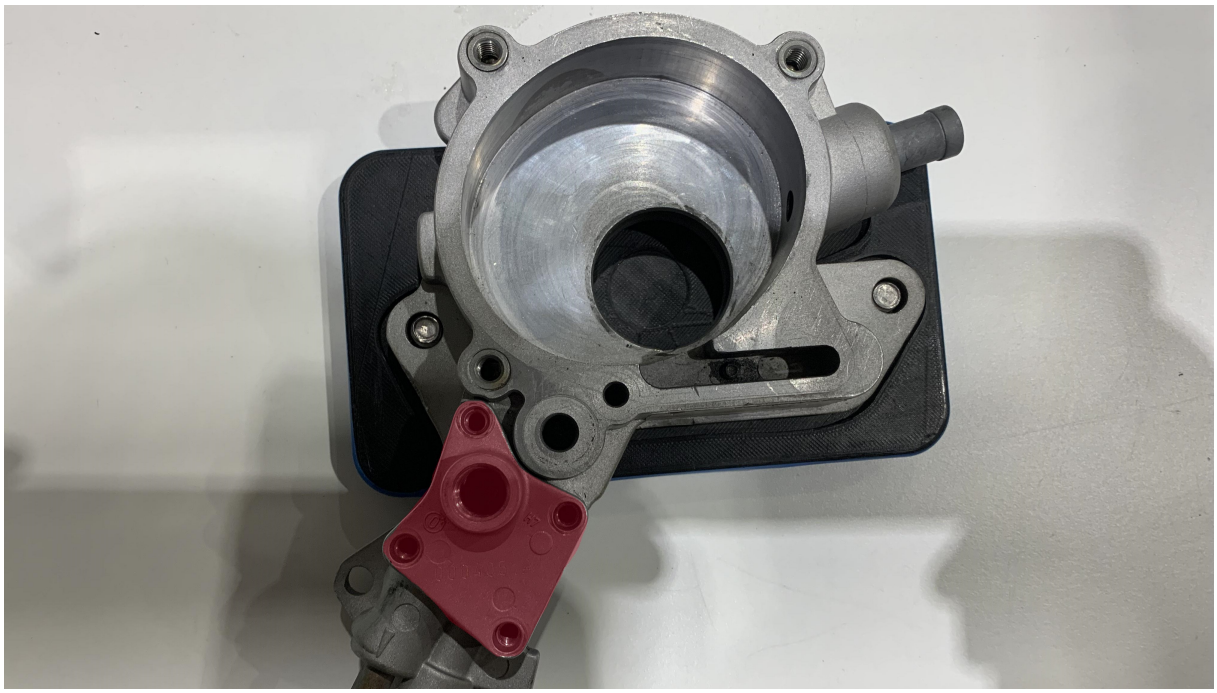


Abb. 21: Beschreibt den Ablauf der 20 wesentlichen Montageschritte der Kraftstoffpumpe. Von der Aufnahme der Montage bis zur Abnahme. Entnommen aus der internen Dokumentation ELAM (2020).

A.5 Montage des Filter

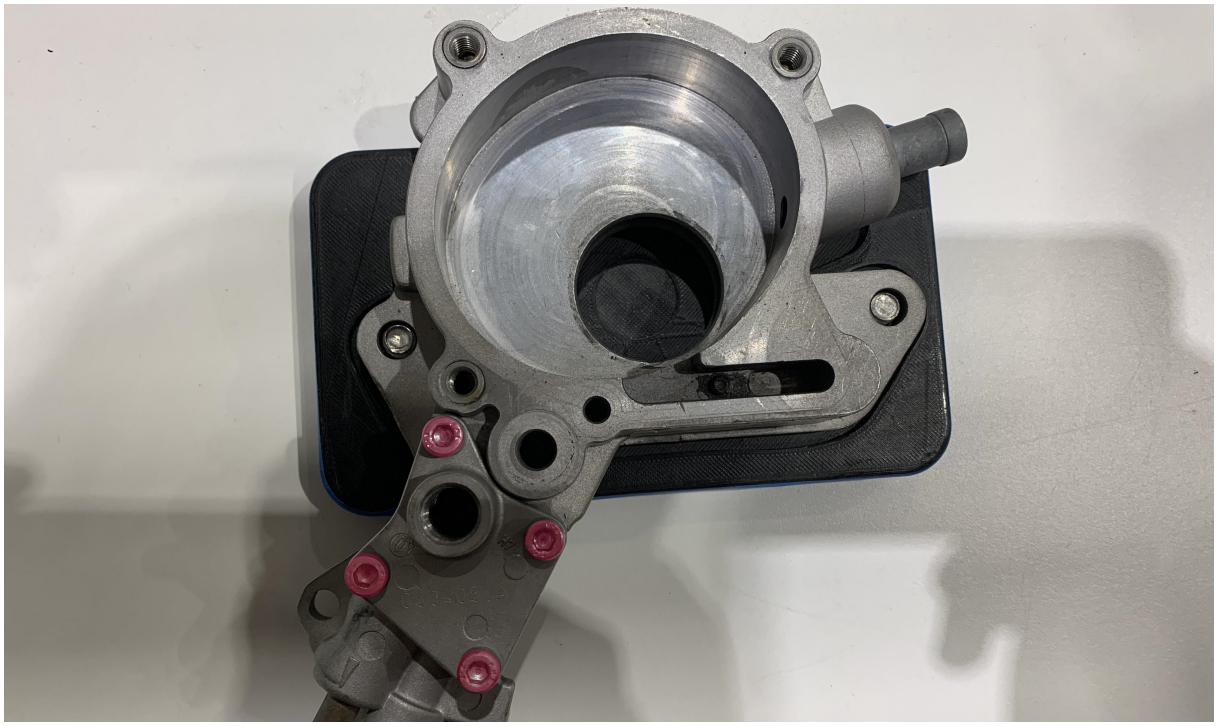


(a) Zeigt das Pumpenuntergehäuse auf der Montageplatte. In rot hervorgehoben ist der Filter.

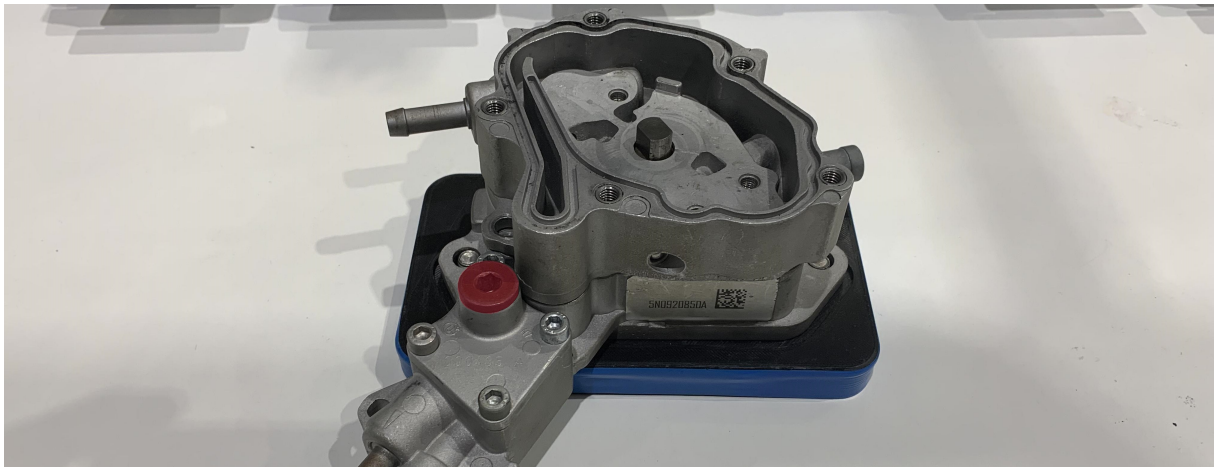


(b) Zeigt Schritt 15 der Montageanleitung. Der Filterdeckel wurde aus dem Fach entnommen und aufgelegt. Der Filterdeckel ist rot hervorgehoben.

Abb. 22: Ausschnitt aus der Montage der Ölpumpe. Zeigt die Montage des Filter, Schritte 15 bis 19 in der Montageanleitung.



(c) Zeigt Schritt 16 und 17 der Montageanleitung. Die vier M5x10 Schrauben wurden angesetzt (16) und anschließend der Filterdeckel verschraubt (17). Die vier M5x10 Schrauben sind rot hervorgehoben.



(d) Zeigt Schritt 18 und 19 der Montageanleitung. Die Ablassschraube wird angesetzt (18) und verschraubt (19). Die Ablassschraube ist rot hervorgehoben.

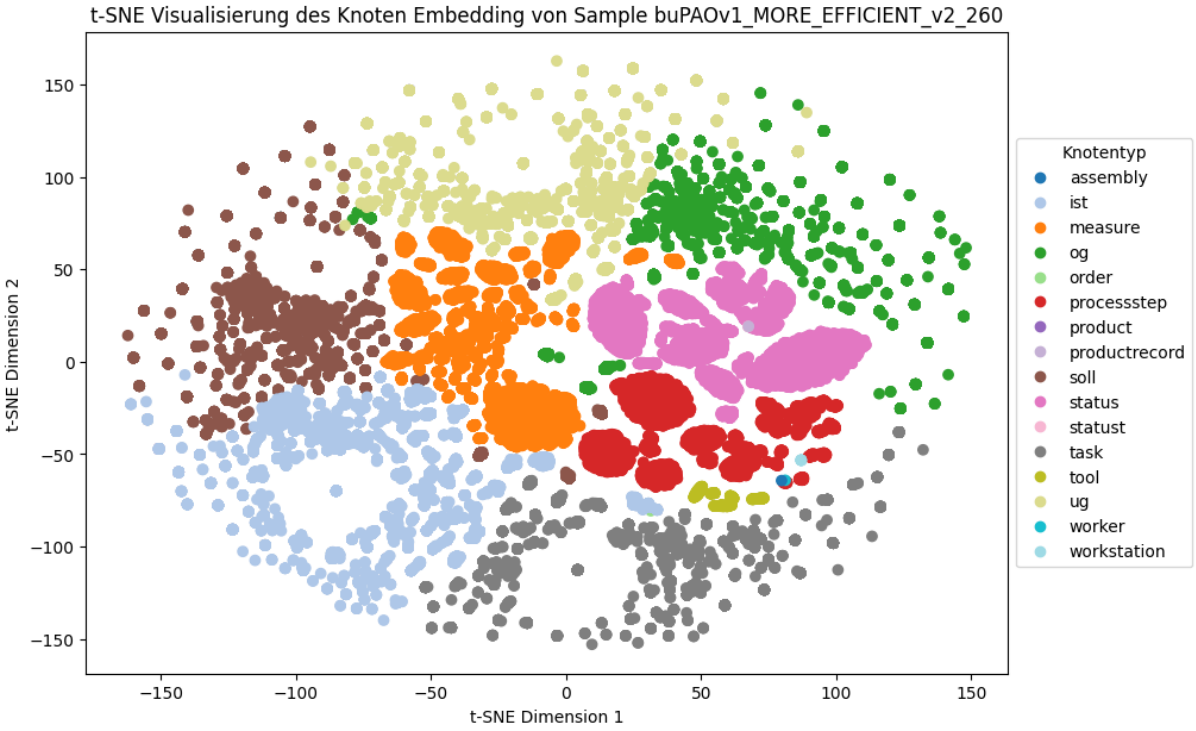
Abb. 22: Ausschnitt aus der Montage der Ölpumpe. Zeigt die Montage des Filter, Schritte 15 bis 19 in der Montageanleitung.

A.6 Zuordnung der Montagedaten zur Ontologie

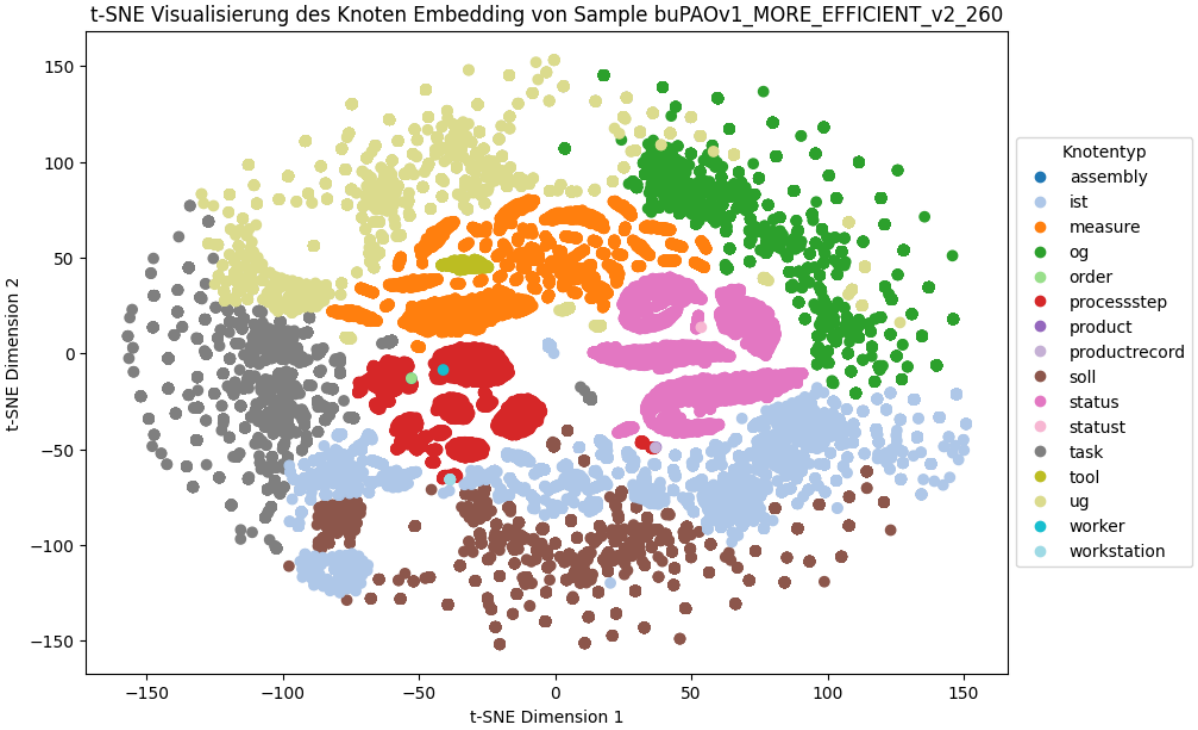
Tab. 8: Zuordnung der Montagedaten zu den Literalen der Ontologie. Ein Literal hat entweder einen Platzhalter Wert oder die der angegeben Spalte der Montagedaten.

Literal	Werte
MAID	MA_NR
hasVisualisationID	Platzhalter: noVisualisationID
hasPartList	Platzhalter: noPartList
hasDeletedID	GELOESCHT
hasSequenceID	SEQUENZNR
hasProductID	AE_ID
hasVariantID	VARIANT_NUMMER
hasVariantName	VARIANT_NAME
hasProductName	Platzhalter: noName
hasLabel	Platzhalter: noLabel
hasPRID	PLA_ID
hasPRDescription	Platzhalter: noPRDescription
hasAssemblyLine	BAND
hasTimestamp	noTimestamp
hasWorkstationName	BEREICH
hasProcessID	“Natürliche Zahl“
hasProcessstepName	Präfix von TAFNAME
hasWorkerName	WERKER
hasWorkshift	Platzhalter: noWorkshift
hasStatusID	STATUS
hasStatusDescription	STATUSTEXT
hasToolID	MESSSTELLE+MAUFNEHMER
hasToolName	MESSSTELLE
hasToolDescription	Platzhalter: noDescription
nextMaintenance	Platzhalter: noMaintenance
hasPort	MAUFNEHMER
hasTaskDescription	TAFNAME
hasValue	UG
hasValue	IST
hasValue	OG
hasValue	SOLL
hasUnit	EINHEIT
hasTimestamp	MESSZEIT
hasType	MESSART
hasMeasureValue	PROGRAMM
hasMeaning	BEDEUTUNG

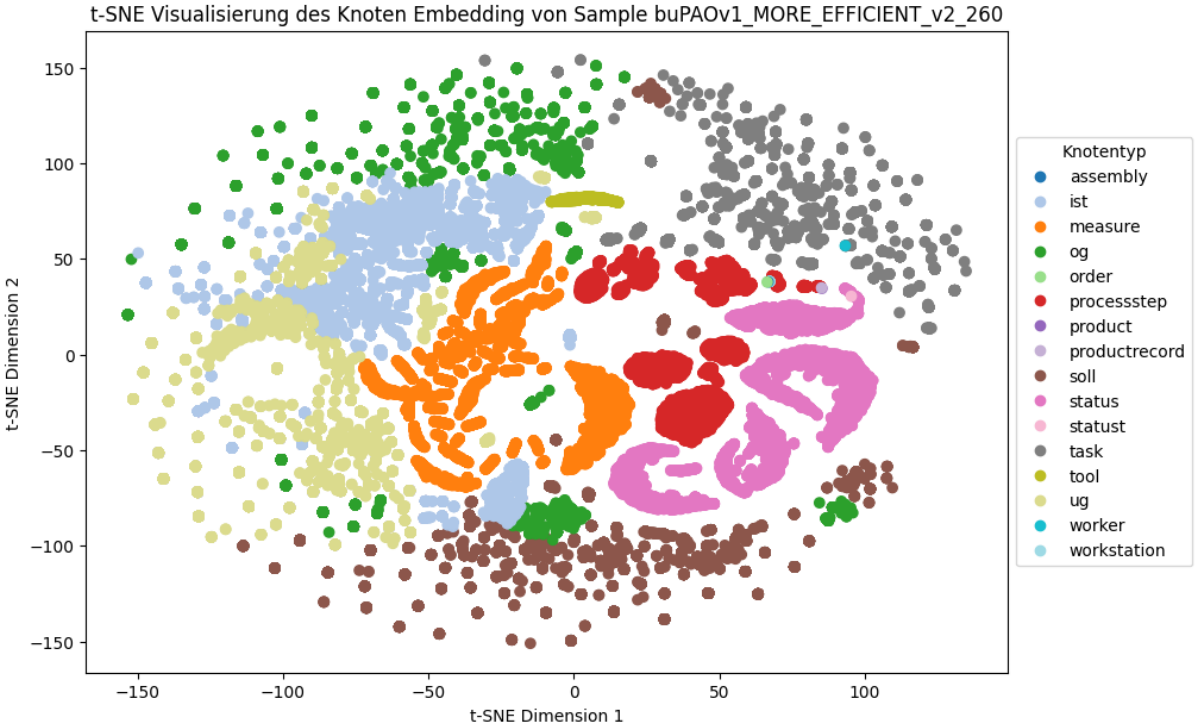
A.7 Modell Embedding Ausgabe



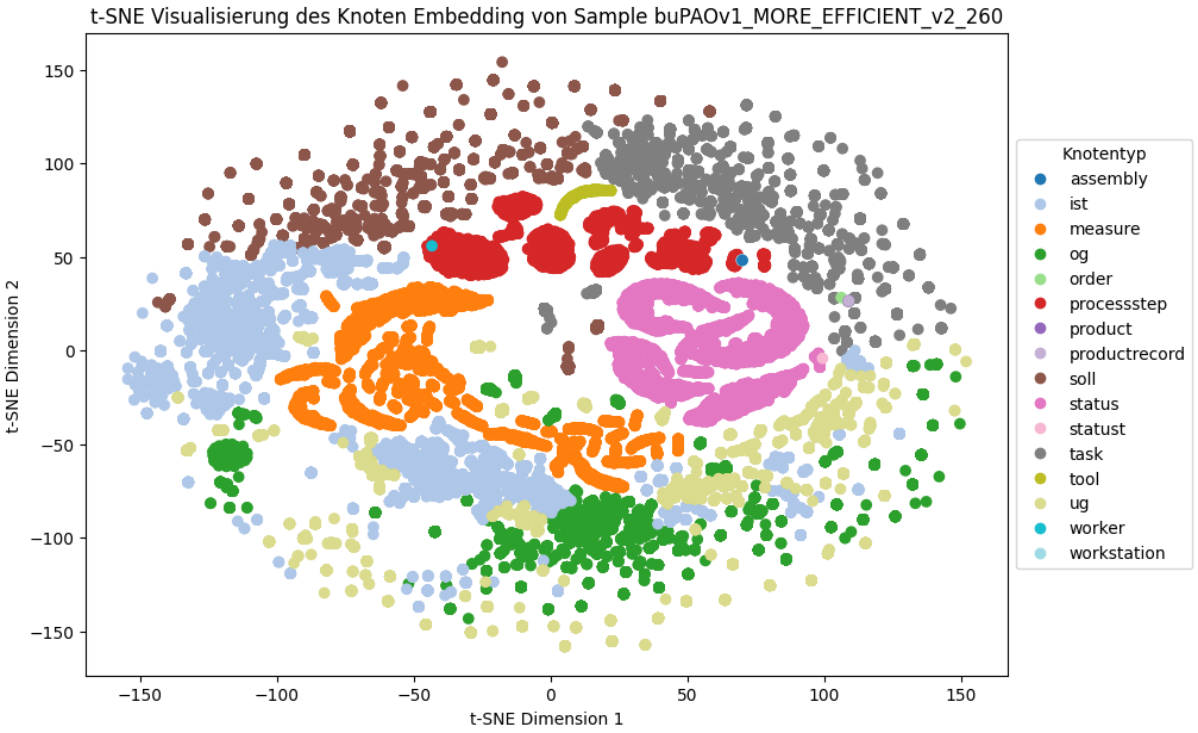
(a) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünften Trainingsepoche.



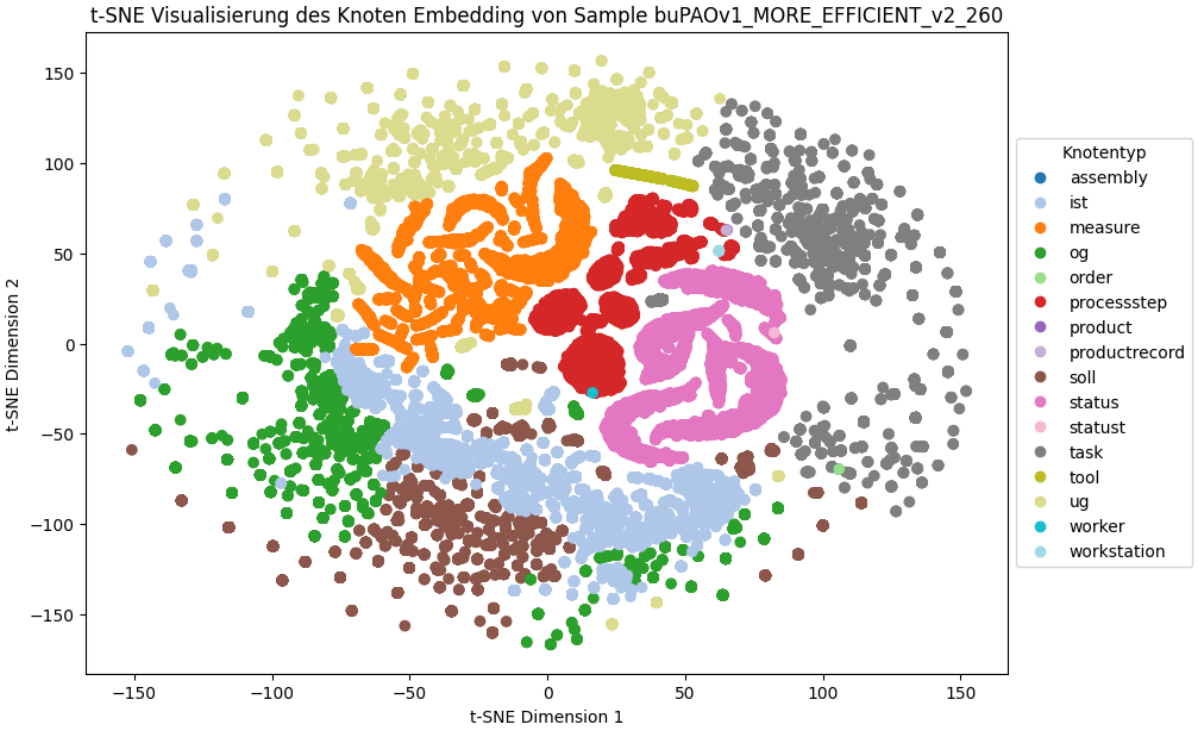
(b) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der zehnten Trainingsepoche.



(c) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünfzehnten Trainingsepoche.

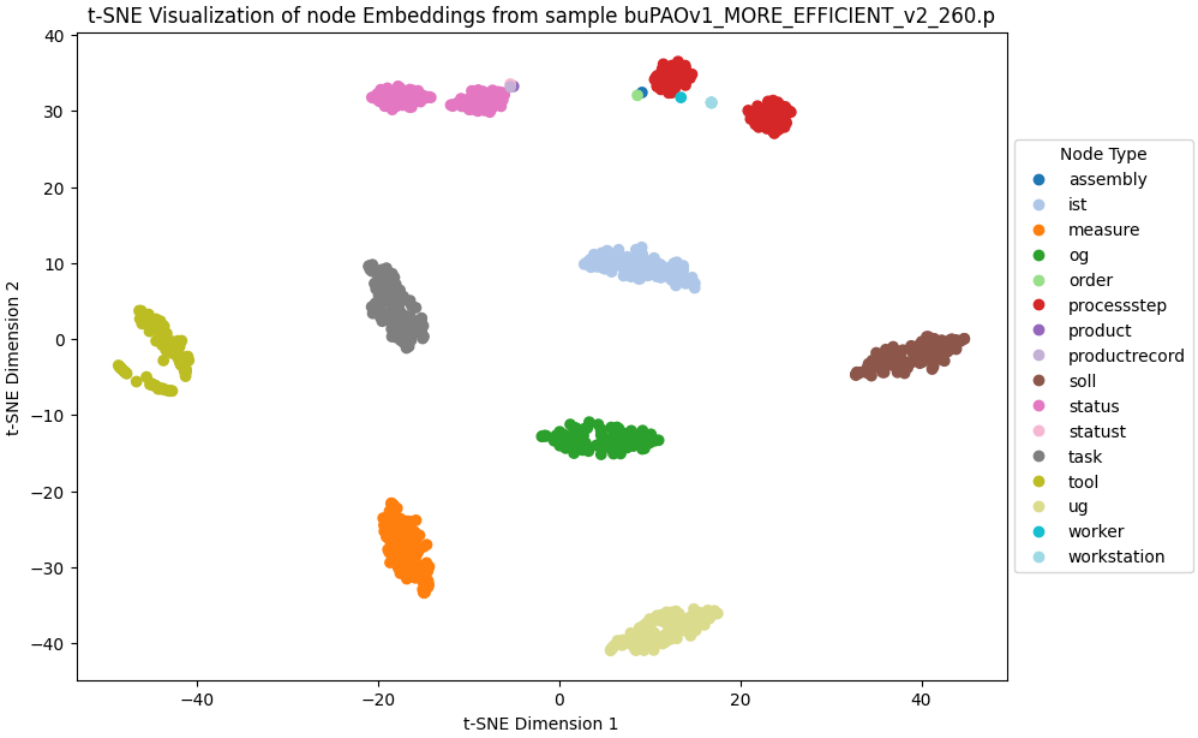


(d) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der neunzehnten Trainingsepoche.

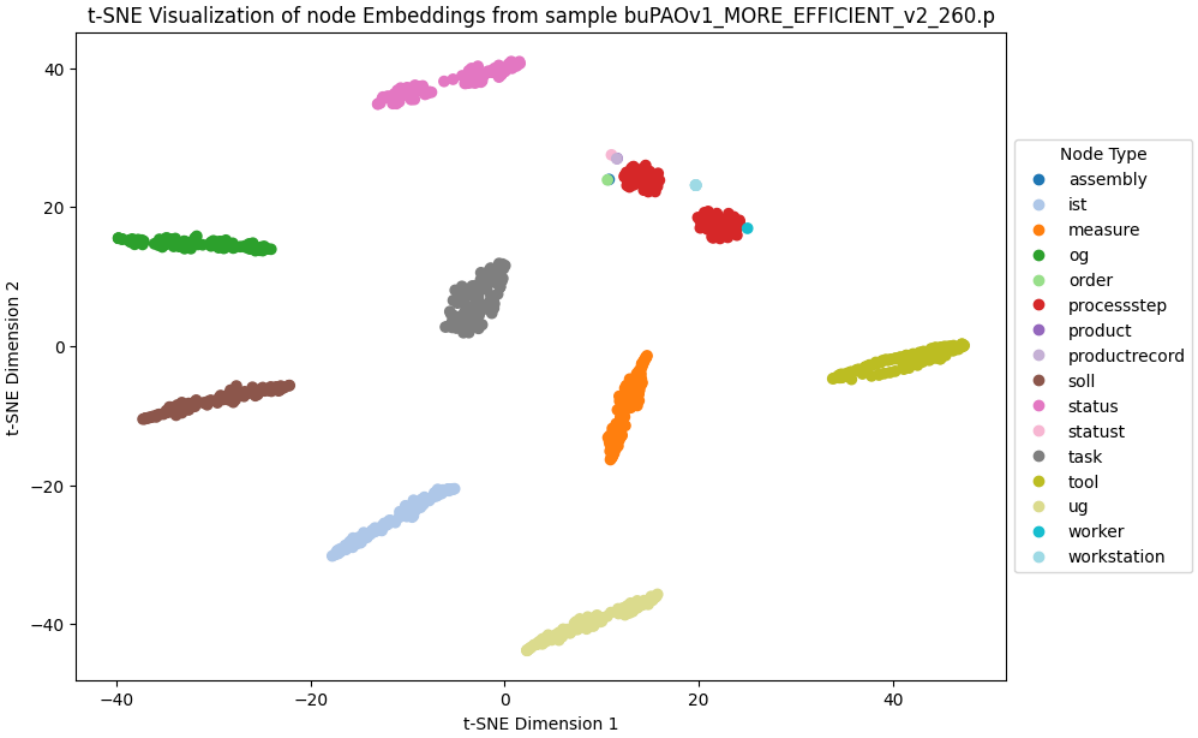


(e) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünfundzwanzigsten Trainingsepoche.

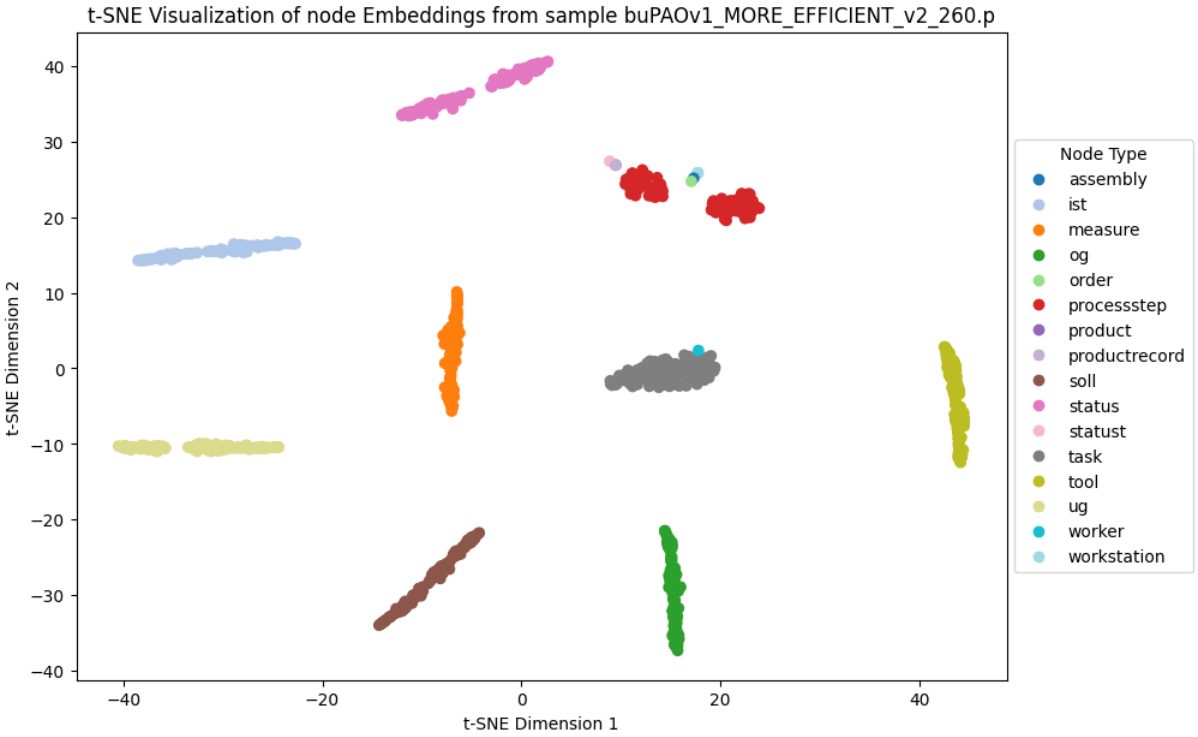
A.8 Reduzierte Modell Embedding Ausgabe



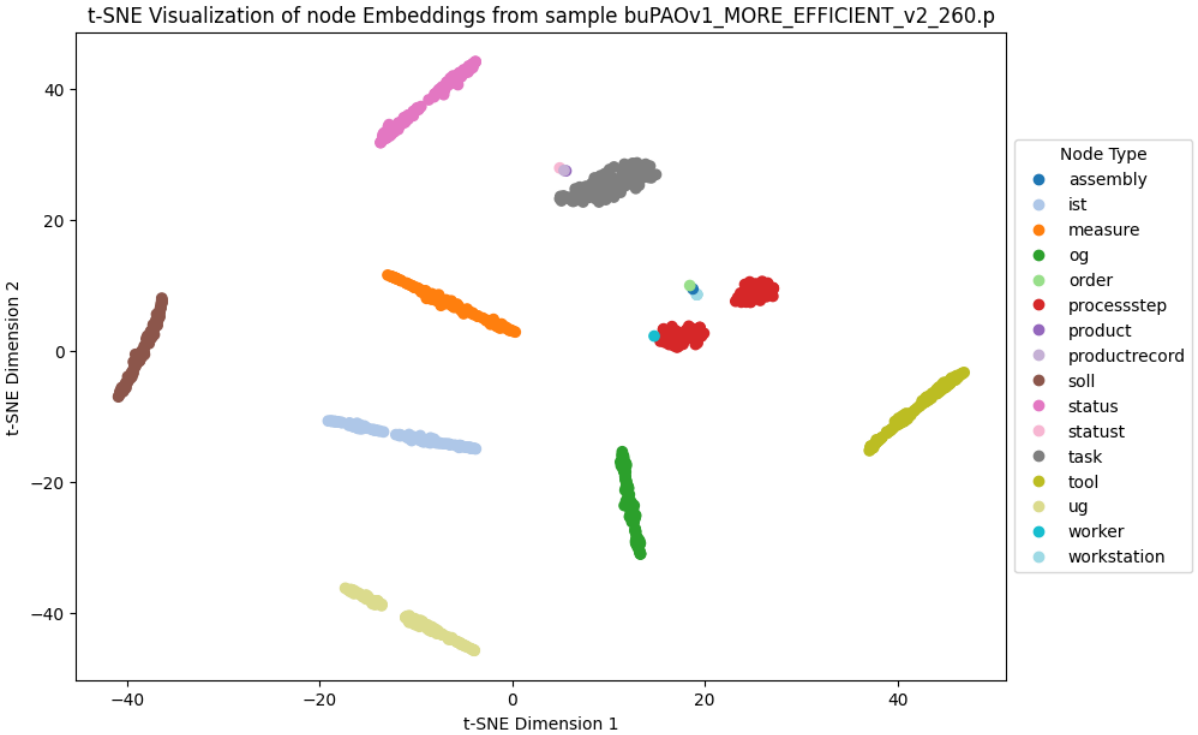
(a) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünften Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt.



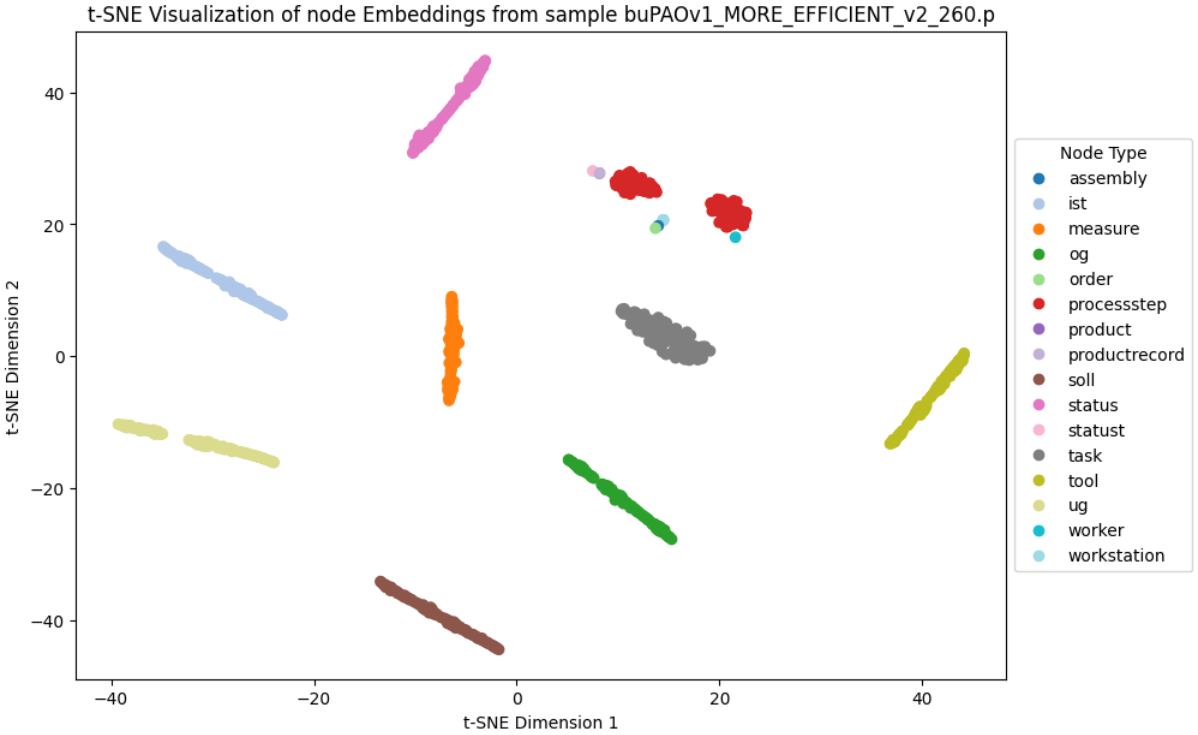
(b) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der zehnten Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt.



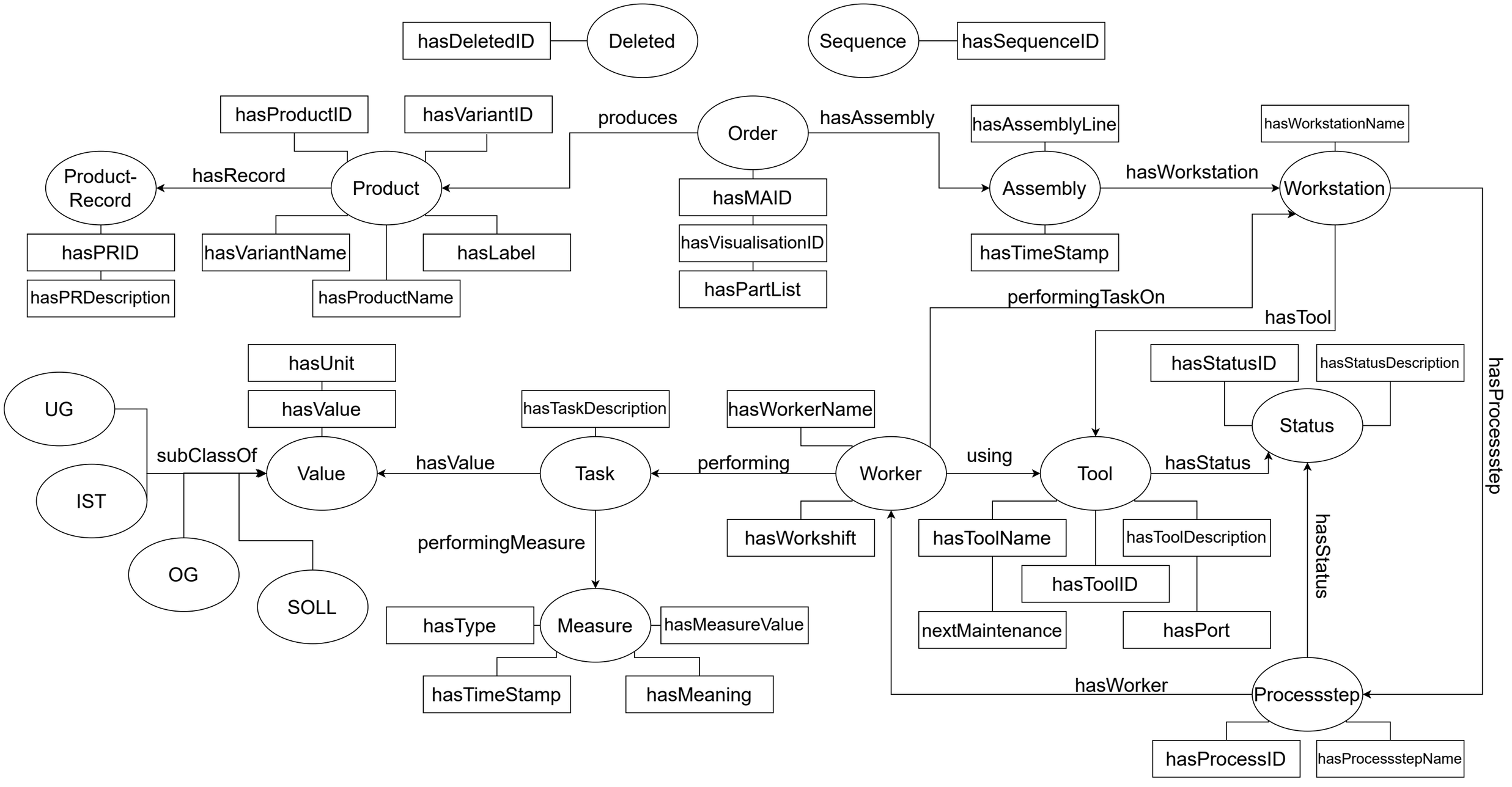
(c) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünfzehnten Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt.



(d) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der neunzehnten Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt.



(e) Die Embedding Ausgabe des Sample *buPAOv1_MORE_EFFICIENT_v2_260* in der fünfundzwanzigsten Trainingsepoche. Die Menge der Knoten pro Knotentyp wurde auf die ersten 100 begrenzt.


hasDeletedID
Deleted
Sequence
hasSequenceID
hasProductID
hasVariantID
produces
Order
hasAssembly
hasAssemblyLine
hasWorkstationName
Product-
Record
hasRecord
Product
hasMAID
Assembly
hasWorkstation
Workstation
hasPRID
hasVariantName
hasLabel
hasVisualisationID
hasTimeStamp
hasPRDescription
hasProductName
hasPartList
performingTaskOn
hasTool
hasStatusID
hasStatusDescription
hasProcessID
hasUnit
UG
hasValue
hasTaskDescription
hasWorkerName
Status
subClassOf
Value
hasValue
Task
performing
Worker
using
Tool
hasStatus
IST
hasWorkshift
hasToolName
hasToolDescription
hasStatus
OG
performingMeasure
hasToolID
SOLL
hasType
Measure
hasMeasureValue
nextMaintenance
hasPort
hasTimeStamp
hasMeaning
hasWorker
Processstep
hasProcessID
hasProcessstepName