

# Bericht zum Modul "Knowledge Graphs"

Im Sommersemester 2025 von König-Ries, Bachinger, Enderling

Daniel Motz

## Inhaltsverzeichnis

<b>1. Modellierung</b>	<b>2</b>
1.1. Competency Questions	2
1.2. Important and Derived Terms	2
1.3. Terms and Classes	3
1.4. Informelle Hierarchisierung	4
1.5. OntoClean: Rigidity, Unity, Identity	4
1.6. Verbesserungsmöglichkeiten	5
1.7. Ontologie	5
<b>2. Integration tabellarischer Daten in die Ontologie</b>	<b>7</b>
2.1. Struktur des Datensatzes	7
2.2. Anomalien im Datensatz	7
2.3. Die Pipeline	8
<b>3. Abfrage des RDF-basierten Knowledge Graphs</b>	<b>14</b>
3.1. SPARQL.1 – Reasoning über der Ontologie und den integrierten Daten	14
3.2. SPARQL.2 – Pizzerien, die Pizza ohne Tomaten servieren	14
3.3. SPARQL.3 – Durchschnittspreis der Pizza Margherita	14
3.4. SPARQL.4 – Pizzerien sortiert nach Stadt	15
3.5. SPARQL.5 – Pizzerien ohne Postleitzahl	15
<b>4. Alignment</b>	<b>16</b>
<b>5. Embedding</b>	<b>17</b>
5.1. Konfiguration und Begriffe	17

## §1. Modellierung

Die Aufgaben in diesem Abschnitt waren das systematische Herangehen an das Modellieren von Wissensgraphen kennenzulernen, eigene Competency Questions zu einem gegebenen Thema (hier die Domäne Pizza) zu erstellen, den Begriff der Concept Hierarchy zu erfassen sowie die wesentlichen Konzepte des OntoClean-Ansatzes zu verstehen und auf die erstellte Ontologie anzuwenden.

### §1.1. Competency Questions

1. Welche Toppings hat eine Pizza Hawaii?
2. Welche Teigsorten gibt es?
3. Welche Saucen können auf einer Pizza sein?
4. Kann ich Brokkoli auf eine Pizza Hawaii legen?
5. Ist eine Pizza Hawaii ohne Ananas immer noch eine Pizza Hawaii?
6. Was gehört zu einer Pizza Margherita?
7. Welche Zutaten hat eine Calzone?
8. Welche Sorten von Pizza gibt es?
9. Welche Pizzen sind vegetarisch?
10. Welche Pizzen sind vegan?
11. Gibt es glutenfreie Pizzen?
12. Welche Pizzen enthalten Mozzarella?
13. Welche Pizzen gibt es bei Mekan?
14. Wie wird die „Pizza FCC“ bei Mekan gemacht?
15. Wie viel kostet eine Pizza Frutti di Mare bei Giovannis Pizzeria?
16. Welche Maße hat die Pizza?
17. Welche Form hat die Pizza?
18. Welche Pizza bei Mekan ist am billigsten?
19. Welche Pizzen enthalten Schinken?
20. Welche laktosefreien Pizzen gibt es?
21. Welche Pizzen sind mit Tomaten belegt?

### §1.2. Important and Derived Terms

**Dickgedruckte** Terme sind für die Termhierarchie relevant.

- Topping → **Zutat**
- **Pizza**
- **Teigsorte**
- **Sauce**
- Brokkoli → **konkrete Zutat** die zu einer Kind von Pizza gehört
- **Pizza Hawaii** → Eine **Kind/Art** einer Pizza mit Einschränkungen im Wertebereich der möglichen Zutaten
- Ananas → **konkrete Zutat**
- **Pizza Margherita** → Eine **Kind**, wie Pizza Hawaii
- **Calzone** → Ebenfalls eine **Kind** von Pizza, allerdings ohne strikte Einschränkungen im Wertebereich der Zutaten
- **Sorte** → **Kinds** von Pizza
- **vegetarisch** → Ein direktes Prädikat für **Zutat** und ein Inferiertes für Pizza, sofern alle Zutaten der Pizza das Prädikat tragen.
- **vegan** → Ebenfalls Prädikat für **Zutat** und ein Inferiertes für Pizza
- **glutenfrei** → Analog zu vegetarisch, ...
- **Mekan** → Eine **Pizzeria**
- **Form / Maße**
- billigsten → **Preis**
- **Schinken**
- **laktosefrei**

### §1.3. Terms and Classes

Unsere Ontologie modelliert nicht konkrete, real existierende Pizzen, sondern Einträge einer Speisekarte. Anfangs modellierten wir eine am Prozess der Fastfoodkette „Dominos“ orientierte Kategorisierung. Wir stellten jedoch fest, dass dies aufwändig und zur Beantwortung der Competency Questions nicht notwendig ist. Wir stellten jedoch die Anforderung, dass die Ontologie sowohl Pizzen die in einer Pizzeria, als auch für solche, die in den „eigenen vier Wänden“, produziert werden, abbilden kann. Die Klasse **Pizza** umfasst daher die für eine Pizza wesentlichen Eigenschaften, wie etwa eine Zutatenliste.

Wir entschieden außerdem, dass weit verbreitete Rezepturen, wie etwa „Pizza Hawaii“, als Unterklasse von **Pizza** mit Einschränkungen im Wertebereich der Zutaten modelliert werden. Dafür ist es notwendig, dass wir entweder Klassen für Zutaten anlegen, wie etwa „Schinken“ oder aber wir liefern zu unserer TBox eine ABox. Wir haben uns für letzteres entschieden. Dies entstand aus der CQ „Kann man eine Pizza Hawaii ohne Ananas herstellen“.

Wir haben uns außerdem dazu entschieden, dass gewisse Zutaten eine eigene Klasse erhalten, nämlich Teig, Sauce und Käse, um erzwingen zu können, dass eine Pizza mindestens aus einem Teig und einer Sauce bestehen muss.

Zu Pizzerien haben wir folgende Gedanken: Eine Pizza kann bei einer Pizzeria hergestellt werden. Dann wird sie zu einem konkreten Preis verkauft und der Preis variiert zudem mit der Größe. In unserer Feldstudie im Raum Jena stellten wir fest, dass es Pizzerien gibt, die an ihrer Speisekartentafel eigens Klassifizierungen vornehmen. So fanden wir beispielsweise die Aufschrift „Alle Pizzen mit Tomatensauce und Käse“. Dies lieferte die Idee, dass eine Pizza gewisse Mindestanforderungen in ihren Zutaten erfüllen muss. In unserer Ontologie wurde daher ursprünglich gefordert, dass eine Pizza aus den Zutaten „mindestens eine Sauce“ und „genau einen Teig“ bestehen muss. Es gibt allerdings Ausnahmen, wie etwa die „Pizza Bianca“, die möglicherweise keine Sauce enthalten – das Bestreichen mit Crème Fraîche ist zwar möglich, aber nicht notwendig. In der Abwägung zwischen einer **konzeptionell sauberen** und nicht **übermäßig restriktiven** Definition wurde entschieden zu definieren, dass eine Pizza mindestens eine Sauce oder einen Käse enthalten muss, um als solche zu gelten. Es gibt andere Gerichte, wie etwa Focaccia, die diese Zutatenkombinationen abdecken und damit nicht in die zu modellierende Domäne „Pizza“ fallen.

Anfangs modellierten wir „Toppings“ bzw. Beläge (bspw. Schinken, Brokkoli, Blumenkohl und Zwiebeln, aber auch Saucen) in der Klasse **Zutat** und fügten data properties für wie **istGlutenfrei** ein. Dies ersetzten wir durch Typen wie etwa **Glutenfreie**, **Vegetarische** und **Vegane**.

Wir haben Teig nicht als vegan modelliert, damit wir ggf. auch Teige mit Ei erlauben können. Die Einschränkung in vegetarisch erscheint sinnvoll, jedoch wird die Zubereitung des Teigs in unserer Ontologie nicht näher spezifiziert.

Die Ontologie enthält ein Beispiel für eine unerfüllbare Klasse: **VegetarischePizza\_Hawaii**, modelliert als Schnitt zwischen **Vegetarische\_Pizza** und **Pizza\_Hawaii**. Die Bedingungen sind also, dass jedes Individuum dieses Typs ausschließlich vegetarische Zutaten enthält, sowie die für eine Pizza Hawaii notwendigen (Ananas, Tomatensauce, Teig/Boden und Schinken). Da Schinken nicht den Typ **Vegetarische** besitzt, kann es die Anforderungen der Klasse **Vegetarische\_Pizza** nicht erfüllen.

**Bemerkung 1.3.1** (Änderungen an der Ontologie basierend auf dem Feedback).

Obwohl die Aufgabenstellung zum Projekt folgendes erlaubte: „The ontology should abstract knowledge about the domain, valid for the data at hand but potentially valid for other datasets (e.g., tables)“, gab es Feedback, dass die Eigenschaft `bewertungAufGoogle` von Pizzerien nicht in den Daten begründet liegt und auch in keiner Competency Question erwähnt wird – sie wurde daher entfernt.

## §1.4. Informelle Hierarchisierung

Die Datei befindet sich im Repositorium unter [terminology-hierarchy.pdf](#).

## §1.5. OntoClean: Rigidity, Unity, Identity

Wir haben feststellen müssen, dass die Validierung von Rigidity, Identity und Unity für die Klassen unserer Ontologie weder trivial, noch (zumindest in einigen Fällen) eindeutig ist. Die folgenden Modellierungen haben wir nach bestem Wissen und Gewissen derart vorgenommen, sodass diese einerseits widerspruchsfrei sind und andererseits (aus unserer Sicht) dem Zweck der Ontologie entsprechen.

### §1.5.1. Rigidity

Von den Competency Questions ausgehend war es sinnvoll für Klassen wie **Pizza\_Hawaii** oder bspw. **Pizza\_bei\_Mekan** anti-rigidity ( $\sim R$ ) zu modellieren, denn man kann bspw. bei der Bestellung einer Pizza bestimmte Zutaten weglassen oder dazuwählen. Da jedoch eine Pizza Hawaii insbesondere durch die Zutat „Ananas“ charakterisiert wird, sollte mindestens diese Zutat wesentlich für die Klassenzugehörigkeit sein. Eine Pizza ist allerdings noch dieselbe, wenn jemand die Ananas von ihr herunterisst, jedoch dann keine Pizza Hawaii mehr.

Die Klasse **Zutat** kann ebenso kritisch bewertet werden. Brokkoli muss nicht notwendigerweise Zutat einer Pizza sein. Im Rahmen der Competency Questions ist diese Unterscheidung irrelevant. Für eine kurze Zeit überlegten wir „Zutat“ als Rolle zu definieren, sodass ein Produkt, bspw. Brokkoli, seine Eigenschaft als Zutat verlieren kann, sofern er keine Zutat bei einer Pizza ist. **Zutat** erhält daher  $+R$ , wie auch alle Unterklassen. Etwa als wir die property `enthaeltZutat` modellierten, war für die Beschreibung der range restriction notwendig die Vereinigung aller möglichen Zutatenklassen zu finden. Mit unserer Zutatenklassifizierung gab es jedoch Zutaten, die in keine Klasse fielen. Ein Beispiel hierfür ist Hinterkochschinken – er enthält Gluten, ist nicht vegetarisch, kein Teig und so weiter. Es ergab daher Sinn die Klasse Zutaten einzufügen.

Für die restlichen Klassen entschieden wir, dass sie die Property  $+R$  erhalten. Eine Pizza könnte zwar auch gegessen werden und eine Pizzeria kann auch auf indisches Essen umstellen, jedoch ist dies von unseren Competency Questions ausgehend keine potentielle Anfrage.

### §1.5.2. Identity

Wir haben uns nachträglich dazu entschieden, ausschließlich die Klasse **Pizzeria** mit  $+I$  zu modellieren und haben dafür die Eigenschaft „Adresse“ hinzugefügt. In allen anderen Klassen lässt sich aus unserer Sicht nicht eindeutig sagen, ob zwei Instanzen gleich (oder unterschiedlich) sind. Diese enthalten dementsprechend  $-I$ .

### §1.5.3. Unity

<sup>g</sup>„Milch“

Instanzen der Klasse **Pizza**, so genannte Pizzen, bilden ein zusammenhängendes Ganzes und werden daher mit  $+U$  markiert. Diese Eigenschaft wird logischerweise an alle Subklassen wie **Pizza\_Hawaii** oder **Pizza\_bei\_Mekan** vererbt. Gleiches gilt für die Klasse **Pizzeria**.

Für die meisten anderen Klassen kann aus unserer Sicht keine eindeutige Aussage über deren Unity getroffen werden. Wenn wir die Klasse **Zutat** betrachten, enthält diese sowohl Instanzen, die als ein Ganzes angesehen werden *können* (eine Tomate, ein Brokkolirösschen etc.), es gibt aber auch Zutaten, die eindeutig kein Unity-Kriterium erfüllen (Käseraspeln, Tomatensauce etc.). Dennoch haben wir uns dazu entschlossen, die Klasse **Zutat** sowie all ihre Subklassen als anti-unity ( $\sim U$ ) zu modellieren und dementsprechend *alle* in unserer Ontologie potentiell vorkommenden Zutaten nicht als ein physisches Ganzes zu betrachten. Beispielsweise betrachten wir anstatt einer Tomate eine Tomatenscheibe als Zutat, welche auch nach Teilen dieser immer noch eine Tomatenscheibe ist.

## §1.6. Verbesserungsmöglichkeiten

Die Benennung der Klasse **Pizza** ist unklar, denn sie lässt weit offen, was mit einer Pizza gemeint ist; erst der Kommentar schafft Klarheit.

## §1.7. Ontologie

Die Datei befindet sich ebenfalls im Repositorium unter [ontology.xml](#).

### §1.7.1. Anreicherung der Properties

Im Zuge der Weiterentwicklung der Ontologie habe ich mich mit der semantischen Anreicherung von Properties befasst. Ziel war es, die Korrektheit und Leistung der Ontologie zu erhöhen und fehlerhafte Assertions durch logische Inkonsistenzen erkennbar zu machen. Dabei orientierte ich mich an den in OWL verfügbaren Property-Charakteristiken wie **functional**, **asymmetric**, **irreflexive** und **inverseOf**.

Zu Beginn fiel mir auf, dass sich bestimmte Eigenschaften – beispielsweise **enthaeltZutat** – besonders gut für eine Anreicherung eignen. Diese Relation ist sowohl asymmetrisch als auch irreflexiv, da es nicht möglich ist, dass eine Pizza eine Zutat enthält, die wiederum die Pizza enthält oder mit ihr identisch ist. Zur verbesserten Navigierbarkeit (mit bspw. SPARQL) innerhalb der Ontologie ergänzte ich zudem eine inverse Eigenschaft **istZutatVon**, sodass sowohl von der Pizza auf die Zutat als auch umgekehrt geschlossen werden kann. Sie sind allerdings beide nicht **functional**.

Auch für die Relation **gehörtZuPizzeria** entschied ich mich dazu, die inverse Beziehung **hatPizza** zu modellieren. Diese Relation ist asymmetrisch, da zwar eine Pizza einer Pizzeria zugeordnet, jedoch das Gegenteil semantisch unsinnig ( $\text{Pizza} \xrightarrow{\text{hatPizza}} \text{Pizzeria}$ ). Die Funktionalität von **gehörtZuPizzeria** – im Sinne der Annahme, dass jede Pizza (ein Eintrag einer Speisekarte) genau einer Pizzeria zugeordnet ist – wurde beibehalten. Dabei ist zu bedenken, dass OWL2 die Eigenschaften nicht automatisch der T-Box hinzufügt. Grundsätzlich sollte eine Inkonsistenz festgestellt, da für eine Assertion in der inversen Relation immer eine Assertion in der ursprünglichen existiert, die die entsprechenden Prädikate trägt. Es ist daher möglich, aber überflüssig, sie in der TBox zu explizieren. Für meine Version von Protegé die HermiT 1.4.3 ausgestattet ist und auch das Paket **owlready2** (das ebenfalls eine Variante von HermiT verwendet) hat sich dies allerdings nicht bewahrheitet.

Ein Beispiel in meiner Ontologie für eine **functional data property** ist **groesse**: Eine Pizza kann genau eine Größe haben und die Aussage ist ungültig, sofern es zwei Assertions gibt die sich auf die gleiche Pizza beziehen.

Beim Prüfen der Datatype Property **waehrung** stellte sich die Frage, ob eine Standardwährung (beispielsweise „USD“) angenommen werden kann, wenn keine Angabe erfolgt. Da OWL keine direkte Möglichkeit bietet, einen Standardwert zu hinterlegen, ohne die Option zur expliziten Abweichung (etwa „EUR“) zu verlieren, entschied ich mich gegen eine globale Einschränkung des Wertebereichs mittels `allValuesFrom`. Stattdessen

Durch diese Erweiterungen ist es mir möglich, die Modellierungsschärfe der Ontologie zu erhöhen und gezielter zu überprüfen, ob konkrete Assertions valide sind. Ein Beispiel: Sollte die Eigenschaft **enthaeltZutat** irrtümlich zyklisch verwendet werden, kann ein Reasoner dies aufgrund der definierten Irreflexivität erkennen. Ebenso erleichtern inverse Beziehungen die Ableitung zusätzlicher Informationen aus wenigen ABox-Einträgen – etwa darüber, welche Zutaten in mehreren Pizzen verwendet werden.

Diese formale Präzisierung stellt einen weiteren Schritt in Richtung einer robusten und qualitativ hochwertigen Ontologie dar, die nicht nur zur Wissensmodellierung, sondern auch zur automatisierten Konsistenzprüfung eingesetzt werden kann.

## §2. Integration tabellarischer Daten in die Ontologie

Die Integration von fehlerbehafteten Daten in semi-strukturierter Form erfordern eine Vielzahl von Abwägungen und ingenieurstechnischen Methoden um die Informationen präzise und semantisch fundiert abzubilden. Im Folgenden werden die getroffenen Entscheidungen hinsichtlich *Entity Identification* und Datenbereinigung diskutiert.

### §2.1. Struktur des Datensatzes


Der gegebene Datensatz ist eine Tabelle mit 11 Spalten. Je Zeile können o.B.d.A. zwei Entitäten bzw. Individuen *identifiziert* werden: Die Spalten `name`, `address`, `city`, `country`, `postcode`, `state` und `categories` beschreiben ein **Restaurant** (`ontology:Pizzeria`), das über einen Namen verfügt, sich an einem bestimmten Ort (beschrieben durch Straße, Stadt, Land, Postleitzahl und Staat) befindet und zudem unter bestimmte Kategorien fällt. Die Spalten `menu item`, `item value`, `currency` und `item description` beschreiben einen **Eintrag in der Speisekarte** (`ontology:Pizza`) des genannten Restaurants. Die Spalten sind in der Datei wie oben beschrieben aufgeführt und suggerieren damit in ihrer Struktur einen Zusammenhang. Nimmt man diese Abbildung an, existiert eine „1:n“-Zuordnung zwischen *Restaurant* und *Speisekarteneintrag*. Dies erscheint schlüssig.

#### Bemerkung 2.1.1 (Categories).

Bei der ursprünglichen Aufstellung der Competency Questions wurden keine aufgeführt, die nach der Kategorie einer Pizzeria fragen. Daher findet sich in der Ontologie diese Object Property auch nicht wieder und wird in diesem Schritt nicht berücksichtigt.

Sofern die Pizzeria gleiche Ausprägungen in den Merkmalen Name, Adresse und Stadt besitzt, wird angenommen, dass es sich um dieselbe Pizzeria handelt. Dies ist das Ergebnis der Abwägung zwischen Exaktheit der Reidentifikation und Übergengauigkeit – sollte bspw. die Postleitzahl in einer Zeile fehlen, jedoch sind Name, Stadt und Adresse gleich, so würde der Vergleich fehlschlagen und eine neue Pizzeria instanziiert werden. Eine nachträgliche Verbesserung war das Aufnehmen von Bundesstaat und Land in die

```
1 pizz_key = (row['name'], row['address'], row['city'], row['state'], row['country'])
```



Die Wahl des Merkmals Stadt ist erstmal arbiträr. Ebenso gut hätte die Postleitzahl anstelle der Stadt zur Unterscheidung der groben Region dienen können. Die Daten haben allerdings einige Einträge, bei denen die Postleitzahl nicht gesetzt ist, der Name der Stadt allerdings sehr wohl. Daher fiel die Wahl (datensatzspezifisch) auf das Merkmal Stadt, statt Postleitzahl.

### §2.2. Anomalien im Datensatz

Es gibt einige Beispiele für Namen und Kategorien die suggerieren, dass die aufgeführten Einträge keine Pizzerien sind (und bspw. die Erfassung der Daten fehlerhaft war).

Name	Categories
24 Hour Express Locksmith Inc	Locksmiths
7 Day 24 Hours Emergency Locks	Locks & Locksmiths

Tabelle 1: Namen, die suggerieren, dass es sich nicht um eine Pizzeria handelt

Jedoch werden in einem Fall 7 und im anderen 17 Speisekarteneinträge aufgeführt.

Einige Einträge erwecken den Eindruck eines Formatierungsfehlers, etwa: gibt es einen Eintrag in dem das erste Feld (name) folgenden Wert enthält: 'l Bistro,100 S 42nd St,Grand Forks,US,58201,ND,"Italian Restaurant,Seafood Restaurant,Pizza Place,Italian Restaurant, Seafood Restaurant, and Pizza Place",Roasted Vegetable and Goat Cheese Pizza,,,Oven roasted vegetables and kalamata olives with marinara sauce topped with goat cheese and mozzarella. Diese Einträge wurden *manuell* durch das entfernen des führenden einfachen Anführungszeichen korrigiert.

Auch das Feld für `item description` wurde verschiedentlich verwendet. Am häufigsten führt es Zutaten an, wie etwa „tomatoes, garlic, mozzarella, basil“. Daher wurde von mir die Annahme getroffen, dass ein typisches `item description`-Feld eine Zutatenliste darstellt. Jedoch finden sich auch Beschreibungen wie

- 1 each (324.00 g)
- Create your own pizza

Die erste Ausprägung suggeriert, dass die Pizza mit dieser Beschreibung 324g wiegt. Dies ist in der Ontologie nicht modelliert und im Sinne der Competency Questions nicht relevant; weiters ist die Information im Datensatz nur selten verfügbar. Eine Erweiterung der Ontologie um diese Data Property wurde daher nicht vorgenommen. Außerdem finden sich mitunter sehr spezifische Angaben für Zutaten:

- Seven layer bean dip
- Sun dried tomatoes
- sliced chicken breast

Relevant im Sinne der Competency Questions ist lediglich die grobe Kategorie der jeweiligen Zutat: „beans“, „tomato“ und „chicken“. Eine später separat besprochene Erweiterung ist die Einführung einer tieferen Zutatenhierarchie, die dies ermöglicht. Ebenfalls werden Singular- und Pluralformen einer Zutat verwendet, bspw. „tomato“ und „tomatoes“. In meiner persönlichen Erfahrung ist dies in der realen Welt keine Angabe der Menge an Tomaten auf einer Pizza, sondern eine Präferenz des Erstellers der Speisekarte. Daher wurde hier festgelegt, dass das Ziel der Extraktion der Zutat immer der Singular sein soll, bspw. „beans“ → „bean“. (Dies ist später für die Integration von bestehenden Knowledge Graphs relevant.)

### §2.3. Die Pipeline

Ich habe diesen Abschnitt zur Vertiefung gewählt, weil viele Möglichkeiten zur Verbesserung in jedem Teilschritt existieren. Die Pipeline ist in etwa wie folgt aufgebaut:

$$\text{data.csv} \xrightarrow{\text{Bereinigen \& Extrahieren}} \text{Pizza- und Zutatenliste} \xrightarrow{\text{Matching}} \text{TBox oder ABox anpassen}$$

Aus den tabellarischen Daten werden natürlich auch andere Informationen (bspw. über Restaurants) extrahiert. Diese Schritte erforderten jedoch nur wenig Bereinigung und sind daher nur in der technischen Dokumentation erwähnt.

Alle Schritte sind fehleranfällig; Bei der Bereinigung der Daten können wertvolle Informationen verworfen und unsinnige als wichtig angesehen werden. Die Auflösung mit den bestehenden Zutaten in der Ontologie kann verwechselt werden, aber auch eine falsche Entsprechung gefunden werden. Selbst ein manuelles Mapping ist nicht fehlerfrei, aufgrund der teilweise fehlenden



Informationen über die Pizzatypen, bspw. „Choose Your Own Pizza“. Auch das Matching mit der Ontologie kann semantisch falsch sein und letztlich muss der Interpretant unter einem Bezeichner der Ontologie nicht dasselbe wie der Entwickler verstehen.

### §2.3.1. Vorüberlegungen

Dieser Abschnitt soll grob zusammenfassen, wie die Pipeline funktioniert und warum ich welche Entscheidungen getroffen habe. Die technischen Details sind im Repositorium dokumentiert.

Im Datensatz finden sich einige typische Pizzasorten/-typen/-klassen, wie etwa *Pizza Margherita*. Hier ist recht klar welche Zutaten zu ihr gehören und gehören sollten. Die Wahrscheinlichkeit, dass populäre Sorten in der Ontologie modelliert sind ist hoch und die Variationen halten sich in geringem Maße. Jedoch gibt es ebenfalls Sorten am anderen Ende des Spektrums oder solche, die je nach Region (und Land) sehr unterschiedlich zubereitet werden. Grundsätzlich sind alle Macharten valide und sollten daher vom Extraktionsschritt berücksichtigt und dem Nutzer in der letztendlichen Abfrage transparent gemacht werden. Dies muss allerdings aus Gründen der Praktikabilität und Überschaubarkeit nur bis zu einem bestimmten Detailgrad möglich sein; diese Schwelle gilt es zur Umsetzung dieses Pipeline-Schritts zu setzen.

Folgende zwei grundlegenden Ansätze sind denkbar: 1) Man geht vom Namen des Menüeintrags aus oder 2) man geht von der Beschreibung aus. Es ist natürlich sinnvoll beide Merkmale zu nutzen, um den Informationsgehalt maximal auszuschöpfen. Eine Möglichkeit, zur Nutzung beider Merkmale, wäre, basierend auf dem Bezeichner des Menüeintrags (bspw. `menu item = Pizza Margherita`) die Zutatenliste durch die Ontologie bei einer Übereinstimmung zu übernehmen (bspw. Tomate, Mozzarella, Basilikum) und in der ABox festzuschreiben. Falls keine Entsprechung existiert, wird kein solcher Pizzatyp bzw. -klasse und die Zutatenliste festgelegt, sondern ein Pizzaeintrag mit der Zutatenliste wird angelegt. Dieser Ansatz hat kein Problem, wenn die Zutatenliste leer ist, solange es einen Pizzatypen zum Bezeichnes aus dem Datensatz gibt, jedoch ignoriert er gänzlich die individuelle Rezeptur einer Pizzeria, die ggf. dem in der Ontologie definierten Typen und seiner Einschränkungen widersprechen. Wenn man beide Informationen einbeziehen möchte, muss man sich über Schritte zur Konfliktresolution einigen. Für meinen Ansatz habe ich eine übersichtliche Menge an Sorten definiert, hier sind die „Pizza Bianca“ und „Pizza Frutti Di Mare“ zu nennen. Für dieses Projekt habe ich mich wegen der Zeitbegrenzung für *Ansatz 1*) entschieden: Die extrahierten Zutaten spielen nur eine Rolle, wenn keine Übereinstimmung über das Feld `menu item` gefunden werden kann.

### §2.3.2. Ingredient Extraction Step

Unabhängig von der Wahl des „Merging“-Verfahrens ist es sinnvoll die Zutaten zu extrahieren und kategorisieren. Dies wäre für einen Datensatz dieser Größe ein hoher Aufwand, insbesondere, wenn man allein agiert. Das Feld `item description` wird verschiedentlich genutzt und ist nicht in jedem Fall eine Zutatenliste (sondern gelegentlich leer oder mit irreführenden Angaben gefüllt). Die überwältigende Mehrheit macht jedoch indikative Einträge über die Gestalt der Pizza. Dies mit klassischen Werkzeugen des Natural Language Processing zu verarbeiten wäre in der Konzeption aufwändig, daher habe ich mich eines Large Language Models bedient.

1. Ein Large Language Model bereinigt die Rohdaten (Größenangaben und sonstige unpassende Beschreibungen werden entfernt). Je Zeile in der ursprünglichen CSV-Datei gibt es nun einen strukturierten Datensatz.
2. Diese Liste wird einem Clusteringverfahren übergeben und führt ein

1. Clustering der `menu item`-Namen in Pizzasorten, und ein
2. Zuordnen der Zutaten in Expertenkategorien durch.
3. Das Clustering wird für die Integration der Daten in die Ontologie abgelegt.

#### §2.3.2.1. LLM Prompt

Der Prompt verlangt, basierend auf den Merkmalen `menu item` und `item description`, zu klassifizieren ob es sich um eine Pizza handelt, welche Zutaten sie enthält und den Namen des Menüeintrags zu kanonisieren. Explizit wird das LLM aufgefordert die Zutaten zu „vereinfachen“ (Beispiel *green pepper* soll zu *pepper* werden). *Dies ist selbstverständlich fehleranfällig*, jedoch gab es keine bedeutenden Auffälligkeiten in einer stichprobenartigen Kontrolle. Der Schritt kann durch eine weitere Anpassung des Prompts und Einstellen der Temperatur (Zufälligkeit), sowie das Auslassen der Aufforderung des Konfabulierens vermutlich verbessert werden. Erstaunlich gut funktioniert die Klassifizierung, ob etwas eine Pizza ist. Der Prompt fordert bspw. „Pizza Bagel“ nicht als Pizza anzuerkennen<sup>1</sup>.

#### §2.3.2.2. Clustering

Das Ziel dieses Clustering-Schritts ist die Verbesserung der Zusammenfassung von gleich oder ähnlich bedeutenden Zutaten. Beispielsweise ist es für die meisten Abfragen irrelevant, ob man *sun dried tomatoes* oder *spiced tomatoes* auf der Pizza vorfinden kann. Der LLM-Step hat jedoch einiges auf der Strecke gelassen (unter anderem das gerade genannte Beispiel). Die Hoffnung ist, dass nach einem Embedding der Zutaten ein Clustering-Verfahren ähnliche Zutaten zusammenfassen kann. Hier soll die Besonderheit von der Domäne Pizza ausgenutzt werden: Bestimmte Pizzasorten enthalten oft die gleichen (oder ähnliche Zutaten) die mit kleinen Variationen oder auch Euphemismen ausgestattet sind. Beispielsweise wird oft *Pecorino* und *Parmesan* auf Pizzen in ähnlicher Art verwendet (weil es beides würzige italienische Hartkäse sind). Im Datensatz finden sich auch solche Sorten gelegentlich, jedoch sind sie etwas unspezifischer, beispielsweise „Italian Pizza“ oder „Junior Pizza“.

Das Clustering setzt ein Sentence Embedding ein. Die Zutat wird nicht einfach nur als Wort mit einem vortrainierten Embedding vektorisiert, sondern in einem Satz der die Zutat benennt und aufzählt, auf welchen `menu items` sie vorkommt. „<Zutat> in Margherita, Caprese, Quattro Formaggi“. Das Clustering wird mit einem *agglomerativem Clustering*-Verfahren berechnet. Dies würde es prinzipiell ermöglichen einen beliebigen Schnitt im Dendrogramm zu setzen, jedoch habe ich mich dafür entschieden das Ergebnis dieses Schritts in eine semiautomatisch angelegte Sammlung von Buckets (das oben erwähnte „Expertenwissen“) mit Stichworten einordnen zu lassen:

- Jeder Blatt-Cluster enthält eine Liste Zutaten.
- Die Zutaten werden nach den Stichworten der Buckets durchsucht.
- Enthält ein Blatt-Cluster ein Stichwort so wird es diesem Bucket untergeordnet.

Dies ist vorteilhaft, weil das automatische Clustering (mit Labeling mithilfe des Zentroiden) einer Kategorie zugordnet werden kann. Die Stichwortsuche ohne vorheriges Clustering war weniger erfolgreich. Der große Vorteil von der Einordnung in manuelle Kategorien ist die Übersichtlichkeit für eine spätere Abfrage mit SPARQL. In einer echten (Produktions-)Anwendung könnte diese Kategorieliste ausziseliert werden um potente Abfragen zu ermöglichen, da das automatische Clustering teilweise zu wünschen übrig lässt.

---

<sup>1</sup>Grundsätzlich wäre ein Pizza Bagel im Sinne meiner Ontologie eine Pizza. Jedoch zeigt dieses Beispiel, dass man unerwünschte Einträge mit einem LLM leicht filtern kann.

### §2.3.2.3. Qualität des Clustering

Für Pizzasorten hat das Clustering schlecht funktioniert. Etwa gehört zur Kategorie „Bianca“ die „Tuna Pizza“, die „Whole Breakfast Pizza“, allerdings auch die „White Pizza“ und ihre Varianten. Aufgrund dieses Ergebnisses habe ich mich dazu entschieden es nicht weiterzuverfolgen. Das Clustering der Zutaten weist ebenfalls Mängel auf.

#### Bemerkung 2.3.2.3.1.

Das Clustering findet sich in der Datei [week2/cluster\\_labels.json](#) und enthält zwei Abschnitte: Pizza-Sorten Clustering (oben) und darunter das Zutatenclustering.

```
1  ... "Other": {
2    "pizza crust": [
3      "",
4      "bbq",
5      "beef",
6      "buffalo",
7      "cheddar",
8      "ground beef",
9      "hot",
10     "lettuce",
11     "pizza crust",
12     "pork",
13     "pulled pork",
14     "ranch",
15     "steak"
16   ], ... }
```

Listing 1: Beispiel für ein semantisch wenig wertvolles Clustering.

Der Bucket „Other“ enthält den Cluster „pizza crust“, der sich zu einem „Catch-All“ entwickelt hat.

Leider konnte auch mit keinem (angemessenen) Schwellwert die Kategorie *Basilikumpesto* gefunden werden (selbst für 0.5 im Agglomerative Clustering wurden sie nicht zusammengefasst). Hier reicht vermutlich die Menge der Trainingsdaten nicht aus um mit diesem Verfahren zu einem angemessenen Ergebnis zu kommen.

```

1    ... "basil pesto sauce": [
2      "basil aioli",
3      "basil pesto sauce",
4      "chipotle pesto",
5      "garlic pesto",
6      "hempseed pesto",
7      "lemon aioli"
8    ],
9    "basil pesto": [
10     "basil pesto",
11     "marinara sauce",
12     "pesto",
13     "pesto sauce",
14     "ranch dressing"
15   ], ...

```

Listing 2: Basilikumpesto wird aufgrund des Worts „Sauce“ nicht in eine Kategorie zusammengefasst.

Das Clustering ist jedoch in Summe sehr gut und fasst Zutaten ähnlicher Bedeutung gut zusammen.

```

1    "Hard Cheeses": {
2      "white parmesan sauce": [
3        "brazil nut parmesan",
4        "white parmesan sauce"
5      ],
6      "pecorino": [
7        "grana padano",
8        "parmesan",
9        "parmesan cheese",
10       "pecorino",
11       "pecorino cheese",
12       "pecorino romano",
13       "romano",
14       "romano cheese"
15     ]
16   },

```

Listing 3: Beispiel für ein gelungenes Clustering: *Hartkäse*

### §2.3.3. Integration der Daten in die Ontologie

Es ist nicht offensichtlich, wie die Ergebnisse des Clusterings in die Ontologie integriert werden sollen. Denkbar sind etwa folgende Ansätze: 1) Man fügt die Fragmente (Expertenkategorie und dem Cluster als Subklasse) der TBox hinzu, erstellt in der ABox die Zutaten als benannte Individuen und gibt ihr den Typ der Clusteringkategorie, 2) man verwendet Vokabulare wie das SKOS um einen Thesaurus bzw. kleines WordNet anzulegen oder 3) man verwendet die generierten Labels der automatischen Kategorisierung und betrachtet sie als Zutat.

Für das Beispiel *Tomate* würde Ansatz 3) fast funktionieren: Die Expertenkategorie *Tomate* enthält einen Cluster mit dem Label *sundried tomato*, welche zumindest einige Tomaten zusammenfasst, jedoch enthält sie auch *tomato sauce*. Würde man diese Ebene zusammenstauchen würde es die Qualität wesentlich verschlechtern. Es gibt jedoch auch noch schlechtere Beispiele, etwa *Mushrooms* oder die bereits genannte *Pizza Crust*-Cluster. Ansatz 2) mit SKOS ist insofern eine Herausforderung, als dass SKOS selbst ein ausgereiftes RDFS-Vokabular ist und eine formale Sprache sein soll. Das Projekt hat jedoch den Fokus OWL als Modallogik; es wäre daher etwas überflüssig (auch wenn SKOS genau für Thesauri und einfache Taxonomien gedacht ist) quasi eine zweite Logik mit eigenen Regeln anzulegen. Ansatz 1) ist daher für den Scope dieses Projekt meiner Auffassung am Besten geeignet.

Dabei ist besonders darauf zu achten, dass ein potenziell schlechtes Clustering keine Qualitätsminderung der Ontologie und des Knowledge Graphen im Vergleich zum Ausgangszustand bewirkt (wie bspw. bei Ansatz 3). Daher ist hier ein Vergleich zum vorherigen Zustand sinnvoll: Die Daten wurden nach dem LLM-Step zu bestehenden Zutaten in der Ontologie und sonst zu Items in Wikidata gemappt und anschließend als Zutat in die ABox übernommen. Durch diese Erweiterung der TBox bleibt diese Information erhalten und der Nutzer kann in der SPARQL Abfrage wählen, ob er die Expertenkategorie „Tomatoes“, den labelled Cluster „Sundried Tomato“ abfragen möchte oder gar direkt die Zutat „tomato“. Die Cluster werden in der TBox mit einem Vermerk „automatisch generiert“ abgelegt, damit Nutzer informiert ihren Umgang wählen können.

### §3. Abfrage des RDF-basierten Knowledge Graphs

Zur Abfrage von RDF-basierten Knowledge Graphs wird in aller Regel der ebenfalls von der W3C spezifizierte SPARQL-Formalismus verwendet. Einige Systeme, wie etwa Wikibase/Wikidata, bilden lediglich in das Resource Description Framework ab und verwenden andere interne Repräsentation.

#### §3.1. SPARQL.1 – Reasoning über der Ontologie und den integrierten Daten

Wie in Abschnitt 1.7.1 angeschnitten können Reasoner verwendet werden um bestimmte Schlussfolgerungen in der TBox zu machen. Es ist grundsätzlich nicht sinnvoll die deduktive Hülle zu explizieren, u.a. weil sich dadurch Speicherplatz sparen und Übersichtlichkeit schaffen lässt. Wird in Protégé (mit Reasoner HermiT 1.4.3) ein Ding als Typ verwendet, so wird gefolgert und expliziert, dass es als Klasse aufzufassen ist. Zum Reasoning verwende ich *OWL RL* (statt *OWL DL* und *Lite*), weil es ausreichend und Stand der Technik ist. Leider hat die Python-Implementation von *OWL RL* den entscheidenden (eigentlich Fehler), dass Tripel inferiert und asserted werden, die nicht dem RDF-Standard entsprechen. Daher enthält das Script für das Reasoning `reasoning.py` eine Funktion `remove_invalid_owl_triples` die Tripel entfernt, bei denen das Subjekt ein Literal ist. Die Python-Implementation nimmt bspw. Statements wie `"0.25"^^xsd:decimal owl:SameAs "0.25"^^xsd:decimal` auf (was GraphDB strikt ablehnt). Die Ursache konnte ich bisher nicht abschließend klären.

#### §3.2. SPARQL.2 – Pizzerien, die Pizza ohne Tomaten servieren

Die Abfrage sucht lediglich nach Dingen vom Typ *Pizzeria* und filtert anschließend mit `FILTER NOT EXISTS` Dinge aus der Ergebnismenge, die eine Pizza servieren die Tomatensauce oder Tomaten enthält. Wichtig ist hier, dass möglicherweise unvollständige Informationen, die allerdings die Ergebnismenge nicht einschränken sollen, als `OPTIONAL` statiert werden, etwa wenn die Adressangabe unvollständig ist: es soll alles verfügbare in das Ergebnis projiziert, aber nicht die Ergebnismenge eingeschränkt werden.

##### Bemerkung 3.2.1 (Nach der Verbesserung).

Durch die Verbesserung hat sich lediglich im `FILTER NOT EXISTS`-Block die Bezeichnung zu `:tomato_ToppingCategory` geändert.

#### §3.3. SPARQL.3 – Durchschnittspreis der Pizza Margherita

Die Abfrage ist insofern unintuitiv, dass es Dinge vom Typ *Pizza* gehen soll, die genau die Zutaten „Tomate“ und „Mozzarella“ enthalten. In SPARQL kann dies formuliert werden als „mindestens *Tomate* und *Mozzarella*“ und „höchstens *Tomate* und *Mozzarella*“ in der Menge der Zutaten. Die Aussage „mindestens“ ist durch die Zeile 6 gegeben, jedoch sind es ohne den `FILTER`-Block in der Zeile 11 ff. potentiell Pizzen, die auch mehr enthalten. Für diese Abfrage war es auch entscheidend Preise auszuschließen die ungültige Fließkommazahlen (`NaN`) sind. Dafür sorgt das `FILTER`-statement in der vorletzten Zeile der Abfrage.

**Bemerkung 3.3.1** (Nach der Verbesserung).

Durch die Verbesserung wird nun nach der Tomaten-Expertenkategorie und dem Cluster für Mozzarella gefragt. Dies hat den Durchschnittspreis von \$15 auf \$13 Dollar gesenkt.

```
1  ?tomato rdf:type :tomato_ToppingCategory.  
2  ?mozzarella rdf:type :mozzarella_Toppings.  
3  ?pizza :enthaeltZutat ?tomato, ?mozzarella.
```

### §3.4. SPARQL.4 – Pizzerien sortiert nach Stadt

Die Abfrage ist insofern korrekt, als sie die Anzahl der Pizzerien pro Stadt ermittelt und nach Bundesland sowie Restaurantanzahl sortiert. Durch `rdf:type :Pizzeria` wird die betrachtete Klasse eingegrenzt. Die Adressdaten werden über verschachtelte **OPTIONAL**-Blöcke eingebunden, sodass auch unvollständige Adressen nicht ausgeschlossen werden. Die Gruppierung über **GROUP BY ?region ?locality** sorgt dafür, dass die Zählung eindeutig auf Stadt- und Bundeslandebene erfolgt. `COUNT(?restaurant)` liefert die benötigte Aggregation, während `ORDER BY ?region DESC(?restaurantCount)` die geforderte Sortierung sicherstellt.

### §3.5. SPARQL.5 – Pizzerien ohne Postleitzahl

Die Aufgabe fordert, dass alle Restaurants, zu denen kein Eintrag zur Postleitzahl existiert, zurückgeliefert werden sollen. Dies ist dadurch erfüllt, dass das **FILTER**-statement alle Einträge aus der Ergebnismenge entfernt, zu denen keine Postleitzahl vermerkt wurde.

## §4. Alignment

Im Wesentlichen sieht die Aufgabe vor, dass man mit drei verschiedenen Ansätzen ein Alignment von der eigenen zu einer gegebenen Ontologie durchführt und die Ergebnisse des Alignments für die eigene Ontologie evaluiert. Im zweiten Schritt, sollen die Verfahren mit einer Ground Truth getestet werden. Ich verwende im Rahmen dieses Projekts vier Verfahren:

1. Eigenes, lexikographisches Alignment mit Übersetzung (`own_alignment.py`)
2. AgreementMakerLight
3. BERTMap
4. ChatGPT o4-mini-high (beide Ontologien werden einfach eingegeben, bei 4o war der Kontext zu klein)

Ergebnisse

1	Loaded reference mappings: 33						
2	aml	TP: 30	FP: 3	FN: 3	P:0.909	R:0.909	F1:0.909
3	own	TP: 5	FP:1576	FN: 28	P:0.003	R:0.152	F1:0.006
4	bertmap	TP: 28	FP: 24	FN: 5	P:0.538	R:0.848	F1:0.659
5	chatllm	TP: 6	FP: 1	FN: 27	P:0.857	R:0.182	F1:0.300



## §5. Embedding

### §5.1. Konfiguration und Begriffe

Für die Ähnlichkeit und Unähnlichkeit wurden jeweils drei Paare gewählt. Es gibt in dieser Betrachtung zwei Konfigurationen:

- Configuration 1: `embed_size=200`, `walk_depth=2`, `reasoner="elk"`, `outfile=cfg1`
- Configuration 2: `embed_size=400`, `walk_depth=4`, `reasoner="elk"`, `outfile=cfg2`

Die Konfigurationen entscheiden sich in zweierlei Hinsicht: Der Größe des Merkmalsvektor für das Embedding und der Random-Walk Depth. Die Tiefe der Random Walks in OWL2Vec\* beeinflusst die Qualität der erzeugten Embeddings deutlich. Größere Tiefen erfassen komplexere semantische Zusammenhänge, können jedoch auch irrelevante oder störende Informationen einbeziehen. Daher muss eine geeignete Tiefe gewählt werden, um ein gutes Gleichgewicht zwischen Ausdrucksstärke und Genauigkeit zu erzielen.

Die Paare wurden gewählt um einem ausgewogenen unterschiedlicher Taxonomieebenen und Typen zu repräsentieren. Margherita und Pizza sind sich vermutlich recht ähnlich, da Margherita eine Einschränkung über der

Term1	Term2
<code>http://ontology.daniel-motz.de/ontology#</code> Pizza_Margherita	<code>http://ontology.daniel-motz.de/ontology#</code> Pizza
mozzarella	kaese
zutat	tomatensauce
margherita	scampi
dessert	waehrung
breakfast	schinken

Tabelle 2: Die zur Beobachtung gewählten Paare

Das Embedding wurde zunächst jedoch nur mit 100 Samples aus der ABox angefertigt. Das Ergebnis war allerdings eintönig: Die Begriffe waren durch das Embedding nicht wesentlich unterscheidbar.

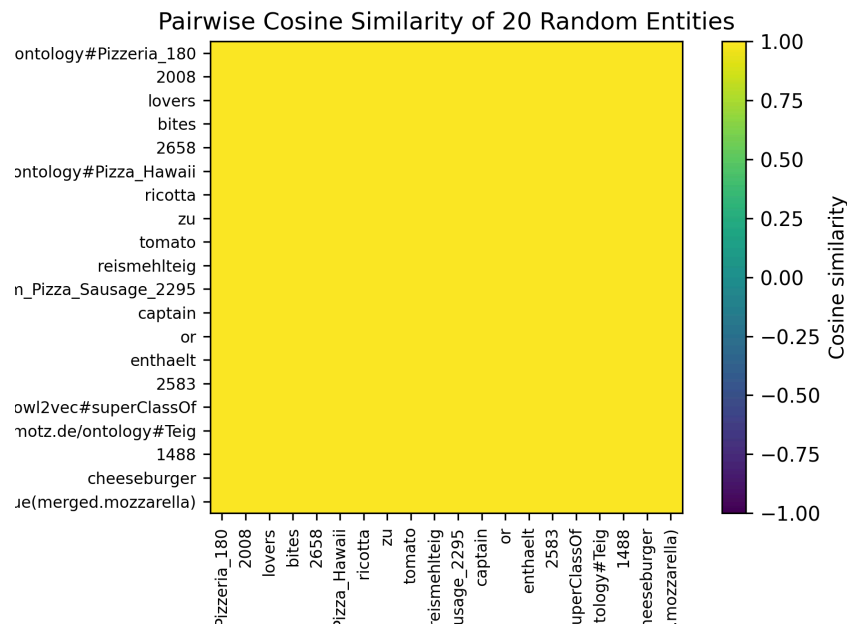


Abbildung 1: Eine Auswahl 20 zufälliger Embeddings,  
 Configuration 1, Sample Size 100

Offensichtlich war die Größe der Stichprobe nicht ausreichend um Unterschiede in den Begrifflichkeiten festzustellen, daher wurde die Sample Size auf zunächst 2000 Beispiele erhöht. Dies führte einerseits zu einer Annäherung an die erwarteten Ergebnisse, als auch eine allgemein höhere Streuung. Dies ist allein durch das Erhöhen der Stichprobe begründet – Word2Vec erhält die Gelegenheit die Worte in mehr und unterschiedlichen Kontexten zu beobachten und kann dadurch die Bedeutung besser approximieren. Die nachfolgenden zwei Tabellen zeigen die Ergebnisse

Term1	Term2	Cosine Similarity	Euclidean Distance
margherita	pizza	0.8380	1.5132
mozzarella	kaese	0.9327	1.0649
zutat	tomatensauce	0.8764	1.4870
margherita	scampi	0.8932	1.2671
dessert	waehrung	0.9882	0.3191
breakfast	schinken	0.9062	1.0817

Tabelle 3: Configuration 1, Sample Size 2,000  
Erwartet ähnliche Embeddings sind grün und erwartet  
unähnliche rot markiert.

Term1	Term2	Cosine Similarity	Euclidean Distance
margherita	pizza	0.9014	2.1179
mozzarella	kaese	0.8048	2.2473
zutat	tomatensauce	0.7766	3.6689
margherita	scampi	0.7736	2.3606
dessert	waehrung	0.9773	0.5176
breakfast	schinken	0.8766	1.2961

Tabelle 4: Configuration 2, Sample Size 2,000  
Erwartet ähnliche Embeddings sind grün und erwartet  
unähnliche rot markiert.

Term1	Term2	Cosine Similarity	Euclidean Distance
margherita	pizza	0.3188	6.3461
mozzarella	kaese	0.6492	2.9628
zutat	tomatensauce	0.6761	3.2407
margherita	scampi	0.3230	6.2231
dessert	waehrung	0.8470	1.1231
breakfast	schinken	0.4825	3.7763

Tabelle 5: Configuration 1, Sample Size 10,000  
Erwartet ähnliche Embeddings sind grün und erwartet  
unähnliche rot markiert.

Term1	Term2	Cosine Similarity	Euclidean Distance
margherita	pizza	0.7744	4.0436
mozzarella	kaese	0.6192	4.4495
zutat	tomatensauce	0.7864	3.7651
margherita	scampi	0.6472	4.9613
dessert	waehrung	0.8779	1.6322
breakfast	schinken	0.9005	2.0396

Tabelle 6: Configuration 2, Sample Size 10,000  
Erwartet ähnliche Embeddings sind grün und erwartet  
unähnliche rot markiert.

Wie schon erwähnt, beeinflussen die Parameter Random-Walk-Tiefe und die Merkmalsvektorgroße wesentlich, wie gut semantische Zusammenhänge durch Embeddings erfasst werden können. Größere Dimensionen und tiefere Random-Walks bieten in der Regel reichhaltigere semantische Informationen, bergen jedoch auch das Risiko, irrelevante oder störende Daten einzubeziehen.

#### §5.1.1. Pairs Expected to Be Similar:

##### 1. **margherita and pizza:**

Zwischen „margherita“ und „pizza“ wurde eine hohe Ähnlichkeit erwartet, da Margherita eine spezifische Pizza-Variante ist. In Konfiguration 1 zeigte sich bei steigender Stichprobengröße von 2000 auf 10.000 Samples zunächst eine hohe Cosine Similarity (0.8380), die jedoch überraschend auf 0.3188 absank, was auf Instabilitäten hindeuten könnte. Configuration 2 hingegen bestätigte die Erwartung konsistenter, mit einem stabilen Rückgang der Cosine Similarity von 0.9014 auf 0.7744.

##### 2. **mozzarella and kaese:**

Auch bei den Begriffen „mozzarella“ und „käse“ wurde eine hohe Ähnlichkeit erwartet, da Mozzarella eine Käsesorte ist. In Konfiguration 1 wurde bei 2000 Samples eine sehr hohe Cosinus-Ähnlichkeit von 0,9327 gemessen, welche jedoch bei 10.000 Samples deutlich auf 0,6492 absank, möglicherweise ein Hinweis auf eine semantische Verwässerung bei größeren Stichproben. Konfiguration 2 zeigte zunächst eine moderate Ähnlichkeit (0,8048 bei 2000 Samples), die sich bei 10.000 Samples leicht auf 0,6192 verringerte und somit unerwartet schwächer wurde.

##### 3. **zutat and tomatensauce:**

Zwischen den Begriffen „zutat“ und „tomatensauce“ wurde eine starke semantische Verbindung angenommen, da Tomatensauce eindeutig eine Zutat ist. In Konfiguration 1 sank die Cosinus-Ähnlichkeit leicht von 0,8764 bei 2000 Samples auf 0,6761 bei 10.000 Samples, was möglicherweise auf eine Kontextverwässerung hindeutet. Konfiguration 2 hingegen zeigte zunächst eine moderate Ähnlichkeit (0,7766 bei 2000 Samples), die bei 10.000 Samples leicht auf 0,7864 anstieg und somit eine größere Stabilität veranschaulichte.

#### §5.1.2. Pairs Expected to Be Dissimilar:

##### 1. **margherita and scampi:**

Bei Begriffspaaren, die semantisch unähnlich sein sollten, wie „margherita“ und „scampi“, wurde erwartet, dass sie klar voneinander entfernt sind, da Margherita vegetarisch ist, Scampi hingegen Meeresfrüchte beschreibt. Konfiguration 1 zeigte bei 2000 Samples zunächst eine überraschend hohe Ähnlichkeit (0,8932), die aber bei 10.000 Samples deutlich auf 0,3230 sank, was besser zu den ursprünglichen Erwartungen passte. In Konfiguration 2 wurde zunächst eine moderate Ähnlichkeit von 0,7736 gemessen, welche bei 10.000 Samples leicht auf 0,6472 zurückging und somit die erwartete semantische Distanzierung widerspiegelte.

##### 2. **dessert and waehrung:**

Unerwartete Ergebnisse zeigten sich bei den klar semantisch unverbundenen Begriffen „dessert“ und „währung“, für die eine niedrige Ähnlichkeit erwartet wurde. In beiden Konfigurationen wurde überraschenderweise eine konstant hohe Ähnlichkeit festgestellt (Konfiguration 1: 0,9882 bei 2000 Samples, 0,8470 bei 10.000 Samples; Konfiguration 2: 0,9773 bei 2000 Samples und 0,8779 bei 10.000 Samples). Dies könnte auf Anomalien in den Embeddings oder auf häufige Co-Occurrence hinweisen, die nicht unbedingt eine semantische Verbindung widerspiegeln.

### 3. breakfast and schinken:

Das Begriffspaar „breakfast“ und „schinken“ habe ich zunächst als unähnlich betrachtet, jedoch ist ein semantischer Zusammenhang aufgrund der Häufigkeit von Schinken zum Frühstück durchaus plausibel. In Konfiguration 1 wurde bei 2000 Samples eine hohe Ähnlichkeit von 0,9062 gemessen, die bei 10.000 Samples deutlich auf moderate 0,4825 sank, was auf eine Kontextsensitivität hinweist. Konfiguration 2 hingegen zeigte eine konsistent hohe Ähnlichkeit (0,8766 bei 2000 Samples, 0,9005 bei 10.000 Samples), was darauf schließen lässt, dass die Embeddings relevante kontextuelle Zusammenhänge effektiv erfasst haben.

#### §5.1.3. Conclusion:

Konfiguration 2 hat grundsätzlich bessere semantische Kontingenzen gezeigt. Die unerwarteten Embeddings sind mit hoher Wahrscheinlichkeit auf Overfitting bei Konfiguration und Mangel an Daten für diesen Ansatz zurückzuführen. „Mangel an Daten“ ist eine allgemeine anwendbare Begründung, jedoch bei der Größe dieses Datensatzes durchaus anwendbar.