



BIRZEIT UNIVERSITY

MASTER THESIS

---

فهم وربط الكينونات بالنصوص العربية في مجال هندسة البرمجيات

Arabic Named Entity Disambiguation and Linking in  
Software Engineering

---

*Author:*

Alaa' OMAR

*Supervisor:*

Prof. Mustafa JARRAR

*This thesis was submitted in partial fulfillment of the requirements for  
the Master's degree in Software Engineering from the Faculty of Graduate  
Studies at Birzeit University, Palestine*

August 28, 2022



**BIRZEIT UNIVERSITY**

MASTER THESIS

Arabic Named Entity Disambiguation and Linking in Software Engineering


By Alaa' Omar

Approved by the thesis committee:

Prof. Mustafa Jarrar, Birzeit University (Chairman)

.....

Prof. Adel Taweel, Birzeit University (Member)

 .....

Prof. Mohammed Khalilia, Birzeit University (Member)

 .....

Date Approved:

## Declaration of Authorship

I, Alaa' OMAR , declare that this thesis titled, “Arabic Named Entity Disambiguation and Linking in Software Engineering” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---



## *Acknowledgements*

During this thesis, I have learned several essential research skills that, I believe, will shape my research career. Therefore, I would like to express my sincere gratitude to Prof. Mustafa Jarrar for the continuous support during all phases of this research; especially for giving me ideas, revising my thesis, and giving me all datasets and tools from his team that I needed to conduct this research.

My special thanks go to Prof. Adel Taweel and Prof. Radi Jarrar for all the discussions and their insights that contributed greatly to this thesis. Special thanks to Prof. Mohammad Khalilia who was continuously supporting me with fine-tuning BERT and reviewing my experiments. I would like also to thank Tymaa Hammouda for developing and deploying the needed APIs and integrating them with Birzeit University's WSD framework; and to Sana Ghanem for reviewing and revising my Wojood-NED annotations.

Nobody has been more important to me in the pursuit of this degree than my family. I would like to thank my parents whose love and support are always with me in whatever I pursue. My mother has always been my source of encouragement, her love and care never ceased to help me during the hardest moments. To my children for their patience and tolerance over the last three years. I must express my very profound gratitude to my husband Aysar for providing me with unfailing support and continuous encouragement throughout my years of study. I would also thank him for his contribution to this thesis by auditing the WikiGlossContext corpus, and preparing illustrations. This accomplishment would not have been possible without him.

Finally, I would like to extend my sense of gratitude to everyone who expressed his support and/or made Dua for me....



# Contents

<b>Declaration of Authorship</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	7
1.2.1 Tasks Definition . . . . .	9
1.3 Research Questions . . . . .	9
1.4 Research Contributions . . . . .	10
1.5 Structure of Thesis . . . . .	11
<b>2 Background</b>	<b>13</b>
2.1 Introduction . . . . .	13
2.2 Knowledge Graphs . . . . .	13
2.2.1 Knowledge Graphs Applications . . . . .	15
2.2.2 Wikidata . . . . .	16
The Wikidata Query Service . . . . .	19
2.3 Neural Arabic Language Models . . . . .	20
2.3.1 BERT . . . . .	20
BERT Architecture . . . . .	21

	BERT Input/Output . . . . .	21
	BERT Pre-training . . . . .	22
	BERT Fine-tuning . . . . .	23
2.3.2	BERT Models for Arabic . . . . .	23
	BERT multilingual . . . . .	24
	AraBERT . . . . .	24
	QARiB . . . . .	25
	CAMeLBERT . . . . .	25
	ARBERT & MARBERT . . . . .	26
2.4	Summary . . . . .	26
<b>3</b>	<b>Literature Review</b>	<b>27</b>
3.1	Introduction . . . . .	27
3.2	Planning and Conducting The Review . . . . .	28
3.3	Named Entity Recognition in Software-related Text . . . . .	33
3.4	Named Entity Disambiguation and Linking . . . . .	39
	3.4.1 Rule-based Approaches . . . . .	39
	3.4.2 Word Embedding Approaches . . . . .	40
	3.4.3 Neural-based Approaches . . . . .	41
	3.4.4 Hybrid Approaches . . . . .	48
	3.4.5 Statistical Approaches . . . . .	51
3.5	Ontology Concept Matching . . . . .	52
3.6	Word Sense Disambiguation . . . . .	54
3.7	Highlight the Research Gap . . . . .	55
3.8	Summary . . . . .	57
<b>4</b>	<b>Research Methodology</b>	<b>59</b>
4.1	The Entity Disambiguation and Linking Process . . . . .	59
4.2	The Need For Annotated Corpora . . . . .	63
4.3	The Wojood-NED Corpus . . . . .	63



4.3.1	Software-specific NEL Tagging . . . . .	64
4.3.2	Linking Wojood-NED to Wikidata . . . . .	68
4.4	WikiGlossContext Pairs Corpus . . . . .	69
4.4.1	Gloss-Context Pairs Extraction . . . . .	69
4.5	Splitting into Training, Validation and Test Datasets . . . . .	76
4.5.1	Splitting WikiGlossContext Induced by The Cross-related Method	77
4.5.2	Splitting WikiGlossContext Induced by The False-Local Method	78
4.5.3	Combined Datasets . . . . .	79
4.6	Summary . . . . .	82
<b>5</b>	<b>Candidate Lookup</b>	<b>83</b>
5.1	Introduction . . . . .	83
5.2	Candidate Lookup Methods . . . . .	83
5.3	Index and Analyzer . . . . .	85
5.4	Query Configurations . . . . .	87
5.5	Candidate Lookup Query Evaluation . . . . .	90
5.6	Summary . . . . .	92
<b>6</b>	<b>Entity Linking</b>	<b>93</b>
6.1	Introduction . . . . .	93
6.2	Entity Disambiguation . . . . .	94
6.3	Environmental Setup . . . . .	95
6.3.1	Experiment Hyperparameters . . . . .	96
6.3.2	Experiments' Tracking . . . . .	97
6.4	Experiments' Results . . . . .	98
6.4.1	Experiments' Set One: BERT Models . . . . .	98
6.4.2	Experiments' Set Two: Different Datasets . . . . .	100
	Remarkable Notes About the Results . . . . .	103
6.4.3	Experiments' Set Three: Disambiguation Evaluation . . . . .	104
	Overall Disambiguation Evaluation . . . . .	106

NED Component Disambiguation Evaluation . . . . .	111
6.5 Implementing Linguist APIs for Entity Linking . . . . .	114
6.6 Named Entity Linking in Software-related Text . . . . .	116
6.7 Summary . . . . .	119
<b>7 Conclusion and Future Work</b>	<b>121</b>
7.1 Introduction . . . . .	121
7.2 Conclusion . . . . .	121
7.3 Future Work . . . . .	124
7.4 Threats to Validity . . . . .	125

# List of Tables

1.1	Research questions . . . . .	10
3.1	Selected sources for review . . . . .	29
3.3	Included related work. . . . .	30
3.2	Inclusion and exclusion criteria . . . . .	34
3.4	Software-specific NER related work . . . . .	35
3.5	Comparison between neural network-based approach related work . . . . .	43
3.6	Comparison between hybrid approach related work . . . . .	49
4.1	Software-specific named entities description . . . . .	65
4.2	Words with tags example . . . . .	66
4.3	Counts of the flat, nested, and total of software-specific entities in the Wojood-NED corpus. . . . .	67
4.4	Statistics about the Wojood-NED software-specific linked entities . . . . .	67
4.5	List of excluded Wikidata types from the WikiGlossContext . . . . .	74
4.6	Statistics about the WikiGlossContext corpus . . . . .	75
4.7	The Wikidata extracted fields' description . . . . .	77
4.8	Splitting WikiGlossContext induced by the cross-related method . . . . .	78
4.9	Splitting WikiGlossContext induced by the False-Local method . . . . .	78
4.10	Combined datasets' description . . . . .	80
5.1	Query configuration statistics . . . . .	92
6.1	NED task environment setup . . . . .	96

6.2	Hyperparameters values . . . . .	97
6.3	Achieved results (%) after fine-tuning three different Arabic BERT models on WikiGlossContext <sub>cross-related</sub> dataset (Table 4.8) . . . . .	99
6.4	Achieved results (%) after fine-tuning Arabertv02 using different datasets (Table 4.10) using ArabGlossBERT test . . . . .	101
6.5	Achieved results (%) after fine-tuning Arabertv02 using different datasets (Table 4.10 on WikiContextGloss and ArabGlossBERT tests . . . . .	102
6.6	A summary of the pre-trained models' information . . . . .	103
6.7	Achieved results(%) per class after evaluating the two components (Candidate lookup and NED) on the Wojood-NED corpus using Wikidata description as a gloss . . . . .	107
6.8	Achieved results(%) per class after evaluating the two components (Candidate lookup and NED) on the Wojood-NED corpus using Wikipedia description as a gloss . . . . .	107
6.9	Achieved results(%) per class after evaluating the fine-tuned models on the Wojood-NED test corpus using Wikidata description as a gloss . . .	111
6.10	Achieved results(%) per class after evaluating the fine-tuned models on the Wojood-NED test corpus using Wikipedia description as a gloss . .	113

# List of Figures

1.1	Example of the entity linking process . . . . .	4
1.2	Knowledge graph representation of software-specific named entities, and their connection to different named entity types . . . . .	5
1.3	Aliases for Object-Oriented Programming Q79872 in Wikidata in the Arabic language. . . . .	8
2.1	knowledge-graph overview (Evans, 2022) . . . . .	14
2.2	Knowledge graph applications.(Zou, 2020) . . . . .	17
2.3	Data structure of item Q251 in Wikidata. . . . .	18
2.4	Wikidata growth (Group, 2022) . . . . .	19
2.5	Wikidata SPARQL query service UI with an example query . . . . .	20
2.6	BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings (Devlin et al., 2018) . . . . .	22
2.7	Overall pertaining and fine-tuning task for BERT (Devlin et al., 2018) .	23
3.1	Literature review methodology . . . . .	29
3.2	Published papers per year . . . . .	30
3.3	NER example . . . . .	35
3.4	General architecture for neural entity linking (Alam et al., 2022). . . .	42

4.1	The presented architecture has three main components : i) NER Tagging that extract named entities from a free-text ii) Candidate Lookup returns all possible candidates for a given ambiguous named entity iii) Entity Disambiguation: BERT-based transformer model to choose the suitable named entity from a set of candidates. . . . .	60
4.2	Sentences distribution . . . . .	67
4.3	Examples of flat entity of mentions of different types. . . . .	68
4.4	NED corpus building and annotating steps . . . . .	70
4.5	Glosses and contexts length in words . . . . .	71
4.6	Aliases as found in Wikidata for the item Q79872 . . . . .	71
4.7	Item Q79872 aliases expansion . . . . .	72
4.8	Example of <i>False</i> pairs generated using cross-relating. . . . .	75
4.9	Example of a negative pairs generation based on the false-local method. . . . .	76
5.1	The Arabic analyzer configuration . . . . .	85
5.2	The English analyzer configuration . . . . .	86
5.3	Query body configuration . . . . .	88
5.4	Achieved results of applying search using the fine-tuned query . . . . .	91
6.1	Tokens to ids representation of the sequence . . . . .	94
6.2	The overall accuracy change for the model $M_5$ when using Wikidata glosses (Table 6.7) vs using Wikipedia glosses (Table 6.8) . . . . .	109
6.3	Cosine-similarity between contexts and glosses . . . . .	110
6.4	The accuracy change comparison per class for the $M_5$ model results (Table 6.8 vs Table 6.10) . . . . .	112
6.5	Example 1: Qualcomm has introduced a new chipset for smartwatches. . . . .	115
6.6	Example 2: Mark Zuckerberg founded Facebook . . . . .	116
6.7	Example 3: YouTube announced that it has started rolling out picture-in-picture support . . . . .	116

# List of Abbreviations

<b>KG</b>	<b>K</b> nowledghe <b>G</b> raph.
<b>KB</b>	<b>K</b> nowledghe <b>B</b> ase.
<b>NED</b>	<b>N</b> amed <b>E</b> ntity <b>D</b> isambiguation.
<b>NER</b>	<b>N</b> amed <b>E</b> ntity <b>R</b> cognition.
<b>NEL</b>	<b>N</b> amed <b>E</b> ntity <b>L</b> ookup.
<b>EL</b>	<b>E</b> ntity <b>L</b> inking.
<b>ER</b>	<b>E</b> ntity <b>R</b> cognition.
<b>SDLC</b>	<b>S</b> oftware <b>D</b> evelopment <b>L</b> ife <b>C</b> ycle.
<b>SRS</b>	<b>S</b> oftware <b>R</b> equirement <b>S</b> pesification.
<b>NLP</b>	<b>N</b> atural <b>L</b> anguage <b>P</b> rocessing.
<b>OSIAN</b>	<b>O</b> pen <b>S</b> ource <b>I</b> nternational <b>A</b> rabic <b>N</b> ews <b>C</b> orpus.
<b>LSTM</b>	<b>L</b> ong <b>S</b> hort <b>T</b> erm <b>M</b> emory.
<b>BI-LSTM</b>	<b>B</b> Idirectional <b>S</b> hort <b>T</b> erm <b>M</b> emory.
<b>SE</b>	<b>S</b> oftware <b>E</b> ngineering.
<b>CRFs</b>	<b>C</b> onditional <b>R</b> andom <b>F</b> ields.
<b>OOV</b>	<b>O</b> ut <b>O</b> f <b>V</b> ocabulary.
<b>BERT</b>	<b>B</b> idirectional <b>E</b> ncoder <b>R</b> epresentations from <b>T</b> ransformers.
<b>POS</b>	<b>P</b> art <b>O</b> f <b>S</b> peech.
<b>WDQS</b>	<b>W</b> ikidata <b>S</b> PARQL <b>Q</b> uery <b>S</b> ervice.
<b>RDF</b>	<b>R</b> esource <b>D</b> escription <b>F</b> ramework.
<b>CSV</b>	<b>C</b> omma <b>S</b> eperated <b>V</b> alue.
<b>UI</b>	<b>U</b> ser <b>I</b> nterface.

<b>QA</b>	<b>Q</b> uestion <b>A</b> nswering.
<b>ML</b>	<b>M</b> achine <b>L</b> earning.
<b>CNN</b>	<b>C</b> onvolutional <b>N</b> eural <b>N</b> etwork.
<b>MLM</b>	<b>M</b> asked <b>L</b> anguage <b>M</b> odel.
<b>NSP</b>	<b>N</b> ext <b>S</b> entence <b>P</b> rediction.
<b>PPR</b>	<b>P</b> ersonalised <b>P</b> age <b>R</b> ank.
<b>WSD</b>	<b>W</b> ord <b>S</b> ence <b>D</b> isambiguation.
<b>MTS</b>	<b>M</b> achine <b>T</b> ranslation <b>S</b> ystem.
<b>WN</b>	<b>W</b> ord <b>N</b> et.
<b>AWN</b>	<b>A</b> rabic <b>W</b> ord <b>N</b> et.
<b>TF-IDF</b>	<b>T</b> erm <b>F</b> requency- <b>I</b> nverse <b>D</b> ocument <b>F</b> requency.
<b>MSA</b>	<b>M</b> odern <b>S</b> tandard <b>A</b> rabic.
<b>DA</b>	<b>D</b> ialectal <b>A</b> rabic.



## *Abstract*

This thesis introduces an entity-linking module for Arabic-named entity disambiguation and linking for software-related texts. The entity-linking process consists of three main components (NER, candidate lookup, and NED). To achieve the first component (NER), we used Wojood, a NER API developed at Birzeit University. However, as Wojood does not support software-specific entities, we extended the Wojood corpus by annotating part of it (the Quora module) with six additional software-specific tags commonly used in the software engineering domain. Second, to implement the candidate lookup component, we downloaded the Wikidata dump and extracted the first sentence from Wikipedia for the Arabic Wikidata nodes connected to Wikipedia using the site links. Then, we used the Elasticsearch framework and customized the querying parameters to enable proper and effective lookup. The candidate lookup component was evaluated using 598 unique named entities' surface forms and achieved (70%) accuracy. Finally, the named entity disambiguation was treated as a sentence-pair binary classification task. Firstly, we built a large corpus of about 1M *True* pairs from Wikidata and Wikipedia called **WikiGlossContext**. The *False* pairs were generated based on the *True* pairs using two different methods (cross-relate and local). We used these datasets to fine-tune different Arabic pre-trained BERT models. We also experimented with different data variations by combining **WikiGlossContext** with the **ArabGlossBERT**<sub>cross-related</sub> and the **ArabGlossBERT**<sub>false-local</sub> datasets. As a result, we fine-tuned eleven different models using three different Arabic BERT models and different data variations using the WikiGlossContext dataset and the combined datasets. The model that was fine-tuned using **WikiGlossContext** achieved the highest classification accuracy (97%). However, the model is over-fitted because of the lexical overlap between the extracted pairs. Finally, to further evaluate the fine-tuned model on an external dataset, we performed a disambiguation evaluation on the **Wojood-NED** corpus using software-specific entities. The disambiguation evaluation was done using glosses of two different knowledge bases. First, we used glosses from the Wikidata description field, and the second time, we used glosses from the Wikipedia summary.

The results showed that the best model achieved the highest linking accuracy (84%) among the other models. More importantly, the best-achieved accuracy of the two components together (the candidate lookup and the NED) was only (74%).

## ملخص

تقدم هذه الأطروحة نموذجاً لربط الكينونات المسماة (named entities) الواردة في النصوص بالكينونات (nodes) المقابلة في الشبكة المعرفية ويكيبيانات (Wikidata)؛ وتركز الأطروحة على الكينونات التي تتعلق بالبرمجيات مثل أسماء لغات البرمجة، والتطبيقات، وأنظمة التشغيل، وغيرها. نموذج الربط المقترح يحتوي على ثلاثة أجزاء رئيسية يخدم كل منها الآخر. يهدف الجزء الأول (NER) لاستخراج الكينونات المسماة من النصوص، أما الجزء الثاني (Candidate lookup) فيهدف إلى استرجاع مجموعة من التعاريف المرشحة (candidates) الموجودة في الويكيبيانات والتي قد تحتوي على التعريف الصحيح للكينونة المسماة. الجزء الثالث (NED) يهدف إلى تحديد أي من الكينونات المرشحة هي الكينونة الصحيحة (فك الالتباس الدلالي)، وبالتالي ربطها بالشبكة المعرفية ويكيبيانات. لتنفيذ الجزء الأول (NER)، قنا باستخدام وجود، وهو واجهة برمجة مستخدمة لاستخراج الكينونات المسماة من النصوص، تم تطويرها في جامعة بيرزيت. نظراً لأن وجود لا يدعم الكيانات المسماة الخاصة بالبرمجيات، فقد قنا بإضافة ستة أنواع من الكينونات إلى مدونة وجود (الجزء الذي تم جمعه من موقع كورا)، خاصة الكينونات التي تُستخدم بشكل شائع في مجال هندسة البرمجيات. أطلقنا على هذا الجزء من مدونة وجود اسم (Wojood-NED). بخصوص استرجاع الكينونات المرشحة (Candidate lookup)، قنا باستخراج جميع الكينونات الموجودة في الشبكة المعرفية ويكيبيانات (Wikidata)، خاصة الكينونات التي لها وصف باللغة العربية. ولكل كينونة (node) في الويكيبيانات قنا باستخراج أول جملة من موقع الويكيبيديا. بعد ذلك، استخدمنا تقنية (Elasticsearch) وضبط الاستعلام لتمكن من استرجاع أفضل مجموعة مرشحة من الكينونات. تم تقييم استرجاع الكينونات المرشحة (Candidate lookup) باستخدام 598 كينونة مسماة فريدة وحقق دقة تعادل 70%. أخيراً، ومن أجل تحديد فك الالتباس الدلالي وتحديد أي الكينونات المرشحة هي الكينونة الصحيحة (NED) تم التعامل مع هذه المشكلة كمسألة تصنيف ثنائي الجملة (sentence pair binary classification). أولاً، قنا ببناء مدونة من حوالي مليون زوج من الجمل الصحيحة، تم استخراجها من الويكيبيانات والويكيبيديا، أطلقنا عليها اسم (WikiGlossContext). قنا بعد ذلك باستخراج أزواج جمل خاطئة آلياً من الأزواج الصحيحة باستخدام طريقتين مختلفتين (الارتباط المتبادل، والاختيار المحلي). وفي النهاية، استخدمنا جميع الأزواج الصحيحة والخاطئة لتدريب نماذج لغوية باستخدام تقنيات التعلم العميق (BERT). قنا بتدريب عدة نماذج، بكميات وتشكيلات مختلفة من البيانات لتحديد النموذج الأفضل دقة. حقق النموذج الأول والذي تم تدريبه باستخدام (WikiGlossContext) الارتباط المتبادل أعلى دقة (97%). ولكن البيانات مجزأة بشكل زائد (overfitted) وذلك لتقارب المعنى بين السياقات والمرشحات. بناء على ما تقدم، ولتحديد النموذج الأفضل دقة، أجرينا تقييمًا للتوضيح باستخدام كيانات محددة وسياقات من مجموعة (Wojood-NED) وهي مجموعة بيانات خارجية لم يتم

تدريب النماذج على بيانات شبيهة بها. تم إجراء تقييم لدقة ربط الكينونات باستخدام تعاريف من اثنتين من القواعد المعرفية. أولاً، استخدمنا تعاريف من حقل وصف ويكيبيانات، وفي المرة الثانية استخدمنا تعاريف من ملخص ويكيبيديا. أظهرت النتائج أن النموذج الأفضل حقق أعلى دقة ربط (84%) مقارنة بالنماذج الأخرى. والأهم من ذلك، تم حساب الدقة الكلية للجزئين معاً (Candidate lookup and NED) باستخدام النماذج المختلفة وكانت أعلى دقة فقط (74%).

*Dedicated to my family, for their unconditional love and support*



# Chapter 1

## Introduction

In this chapter, a general overview will be presented about Arabic named-entity disambiguation and linking. Firstly, the motivation of our work is presented by starting with the importance of named entity disambiguation in software-related texts using knowledge graphs (KGs). While the problem statement and the research questions will be discussed in sections 1.2 and 1.3 respectively. Finally, the main findings, contributions, and document outline will be presented at the end of this chapter.

### 1.1 Motivation

The World Wide Web is considered a valuable source of information, as it hosts a huge amount of unstructured data such as social networking and question answering websites, bug repositories, application reviews, scientific papers, learning platforms, news articles, archived mailing lists (Malyshev et al., 2018a). Recently, websites such as StackOverflow and Quora have proven their existence as a highly valuable source of knowledge for both researchers and software developers (Derczynski et al., 2014), as these platforms play a significant role in problem-solving and knowledge sharing for workers in the domain of software engineering and development. The primary objective of content reuse on these websites is to seek discussions about a specific software-related entity, for example, (a programming language, framework, or programming paradigm), to discover valid usage patterns, solutions to bugs, or alternatives (Ye et al., 2016).

The richness of software-related information is continuously growing at an exponential rate on such platforms. User-created content on these platforms has become a vital root of knowledge. However, most of the user-generated content is in the form of unstructured data, e.g., free text, photos, and videos. With regard to the market survey (Gravili et al., 2018), between 2009 and 2020, the amount of digital data will grow by a factor of 44, but the staffing and investment to manage it will grow by a factor of just 1.4. With the present direction, data related to software-aspects (we mean by software-related aspects in this thesis, the texts related to the software development life cycle that contains software-specific named entities defined in Table 4.1, such as user stories that are used to capture features in Agile, software requirements, bugs descriptions, discussions about software-related issues, to mention a few) will magnify considerably at a faster rate than ordinary digital data. Transacting with such a major mismatch is a major defy. In this context, one of the propositions to this issue is to promote tools for information extraction, with the aim of converting unstructured data into structured data (Marrero et al., 2013).

The abundance of socio-technical data about software-related aspects on the web motivates the research community to invest more in developing frameworks for information retrieval, information extraction, and search engines with an aim to get the value of this unstructured content in a way that positively impacts the software development life cycle and aids software engineers in better doing their jobs. For example, the authors Malik et al., 2021 said that “in the software requirement Extracting the main entities in an software requirements specification (SRS) document helps in sorting the data and identifying necessary information, which can be important for the development process, quality assurance, operations, and maintenance in the software industry.” While in software bug-specific, the authors C. Zhou et al., 2020 said that “existing studies have some limitations on mining bug data. First, the overall structural features and inherent semantic information of the bug report are ignored during the data processing” viz., available information retrieval tools often neglect the semantics of the words and



harm the accuracy of classifying bugs. Besides, the same authors C. Zhou et al., 2020 said that “developers still have to spend a lot of time reviewing and analyzing these bug reports in order to extract the information they want. These limitations will reduce the accuracy of information retrieval and affect the efficiency of bug repair tasks” . Since the software engineering-related content on the World Wide Web, and formal documents such as SRS are written in natural language and tend to be ambiguous, it is important to be first pre-processed to be understood. Ambiguity means that the reader interprets the written piece of information in different ways, or different readers interpret it in many different ways. The main objective of the pre-processing phase is to reveal ambiguity and improve document understanding. For this aim, natural language processing (NLP) tools are used for information extraction from free text such as app reviews, and bugs repositories (Maalej et al., 2016; Maalej & Nabil, 2015), to mention a few. However, Ryan, 1993 stated “NLP is not appropriate to be used as the resulting reliability of the system is doubtful”. According to the survey on the businesses requiring software conducted at the Università di Trento in Italy (Mich et al., 2004), the authors found that most of the documents ready for software analysis are provided by the end-user are obtained by conducting interviews. Furthermore, 71.8% of these documents are written in a common natural language, while 15.9% is written in structured natural language, but only 5.3% of them are written in the formalized language (Mich et al., 2004).

One of the major challenges researchers face when mining software-related content is that the text needs to be well-understood. However, text semantics are typically obvious to humans, but not to machines (Rizzo & Troncy, 2012). For example, given the sentence “I hate messenger and WhatsApp they are bad programs”. This sentence might be obvious to humans - but not machines. Machines do not know if “messenger” means a prophet or a postman or if it is the name of a software application. In this research, we will focus on this issue (understanding the semantics of a given text). This will be achieved by recognizing six specific software named-entity types ( programming

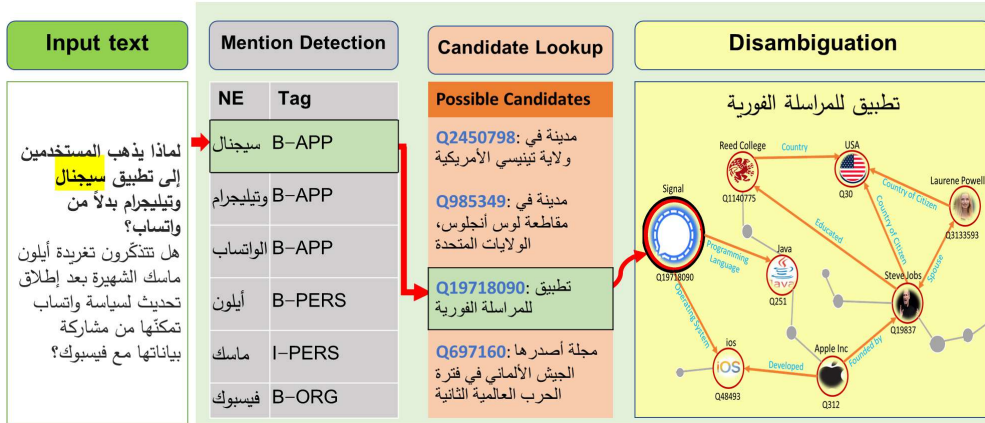


FIGURE 1.1: Example of the entity linking process

language, software standard, framework, software standard, programming paradigm and platform) in a given sentence and linked each named entity with its corresponding node in Wikidata. Figure 1.1 illustrates, by example, the stages of each component involved in the entity linking task, as the entity *سيجنال*/Signal in this example is linked to the **Q19718090** *تطبيق للمراسلة الفورية*/free encrypted communications app <sup>1</sup>. As the first step, we have to be able to recognize named entities - known as named entity recognition (NER) - in the software-related context and classify these entities into a set of predefined categories <sup>2</sup>. Second, for each named entity from the first step, the candidate lookup component is used to look up all possible matches for that entity on the knowledge graph. Finally, the named entity disambiguation component is used to determine which candidate from the returned list is the correct one based on the context that contains that named entity.

The objective is to arrange the extracted insights and information into knowledge bases that define the relationship between software-related entities together, in a manner that

<sup>1</sup><https://www.wikidata.org/wiki/Q19718090>

<sup>2</sup>The NER implementation is beyond this thesis objectives. However, we used the Wojood-NER (Jarrar et al., 2022) API that is published by Birzeit university. And we manually annotated Wojood-NED which is part of Wojood, with additionally six software-specific tags and used this corpus for evaluation purposes. For full details, please refer to chapter 4

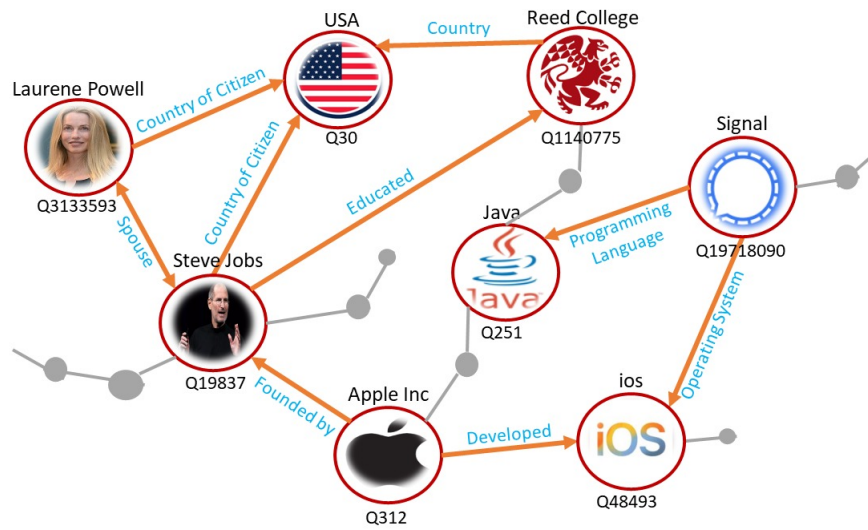


FIGURE 1.2: Knowledge graph representation of software-specific named entities, and their connection to different named entity types

facilitates finding different software entities with high accuracy. Such a way of organizing knowledge about software-specific named entities could be represented as "knowledge graphs" (Meij et al., 2014). Knowledge graphs which have many applications can help in entity disambiguation by linking a specific software named entity with its representative node in a knowledge graph, and hence, helps reach related content about a specific software-named entity with other entities. This information includes relations to another related software-related content in a triple format representation (subject, predicate, object). Figure 1.2 demonstrates a simple example of how named entities are stored in a knowledge graph, and how relations between entities are maintained, for example (Apple Inc, Developed, iOS) is a triple that reads "Apple Inc. developed iOS".

Entity Linking (EL) is defined as matching the mentions aka (Named Entities, surface forms) with their corresponding nodes on the knowledge graphs (Möller et al., 2021), such as Wikidata (Vrandečić & Krötzsch, 2014), Freebase (Bollacker et al., 2008), DBpedia (Auer et al., 2007), or Yago4 (Pellissier Tanon et al., 2020). Wikidata differs from the aforementioned knowledge bases (KBs) in such that it depends on the community

to add and update its content<sup>3</sup>, the rest of the KGs such as Freebase and Yago which are one-time generated KGs and hence, became outdated. While DBpedia remains updated with a lag of one-month<sup>4</sup>. Hence, it is more suitable for recent modern industrial applications such as smart assistance (Möller et al., 2021). Furthermore, Wikidata nodes are multilingual. That is, the same mention has labels, descriptions, and aliases in many languages. These factors attract more researches to link with Wikidata. The presence of named entity aliases helps in candidate generation modules, and hence in the entity linking task as it increases the possibility of entity surface forms. This feature is not available, for example, in DBpedia. The existence of the item description on Wikidata, which is a short sentence that is expressed in natural language helps in entity disambiguation for similar items with the same label by providing a local context of the item. NED is substantial for Information Retrieval (IR) and semantic analysis to make precise analysis of base entities rather than fuzzy mention chains (Hachey et al., 2013). Moreover, named entity disambiguation (NED) can assist in developing applications such as entity-based research. Entity linking helps to supplement search results with further semantic information, resolve query ambiguity, and to limit the search space (Alam et al., 2022), text summarizing, and news analysis (Hoffart et al., 2014).

In this research, we are interested in linking named entities in the Arabic language with its Arabic matching nodes in Wikidata. Arabic is the official mother tongue of millions of people in the Middle east. Therefore, it is important to develop tools that can help avoid ambiguity in the software engineering phases such as requirements, development, bugs, and testing. Moreover, Arabic is widely used in community discussions related to software issues using emails, blogs, and software management tools, to mention a few. Experience has proven that ambiguity of user-written related content in the Arabic language is a severe problem that decreases the production of high-quality software and increases scheduling and affects budget during the mismatch in understanding the ambiguous mentions and intent (Elazhary, 2016). Arabic has the highest population

---

<sup>3</sup><https://stats.wikimedia.org/#/metrics/wikidata.org>

<sup>4</sup><https://release-dashboard.dbpedia.org/>

growth rate on the Internet between 2000 and 2013 with a growth rate of about 5000%. Thus, unorganized Arabic content on the Internet is growing rapidly. For example, in March 2014, Arabic users on Twitter contributed on average with 17.5 million tweets per day <sup>5</sup>. However, it is still considered a resource-poor language (Gad-Elrab et al., 2015). The structured and semi-structured content on knowledge graphs and knowledge bases such as Wikidata and Wikipedia lag behind when compared to the English language.

## 1.2 Problem Statement

Resolving ambiguity caused by software-related content manually is a laborious, time-consuming, error-prone process, and thus costly (Berry et al., 2000). Besides, current research tools intended to extract and classify information from software-related content lack accuracy and neglect semantics (C. Zhou et al., 2020). This leads to confusion, wasted effort, and rework. Due to the ambiguity in the intent of internet users in QA websites, App stores, user stories, etc., and the difficulties arising from the morphological nature of the Arabic words and understanding their semantics which are listed in detail below. This research introduces an entity linking components to resolve semantic ambiguity, mainly in software-related texts. The aim of these components is to link software-specific named entities with their corresponding nodes in Wikidata to resolve their semantic ambiguity.

**Software-related named entity disambiguation limitations:** When writing about software-related aspects (e.g., user story, app review, bug report, or security issue) especially on social-technical websites (e.g., Quora or StackOverflow), users do not always describe their intent in a manner that follows the standardized naming conventions and description formats. Moreover, the user feedback usually contains unstructured text which compromises mixed languages embayed with code, abbreviations, slang, dialects, software-specific vocabulary, and vague language, which makes the NER task more difficult especially for the Arabic language. Hence, affects the NED process as errors

---

<sup>5</sup><http://www.arabsocialmediareport.com/>

propagated from NER into NED.

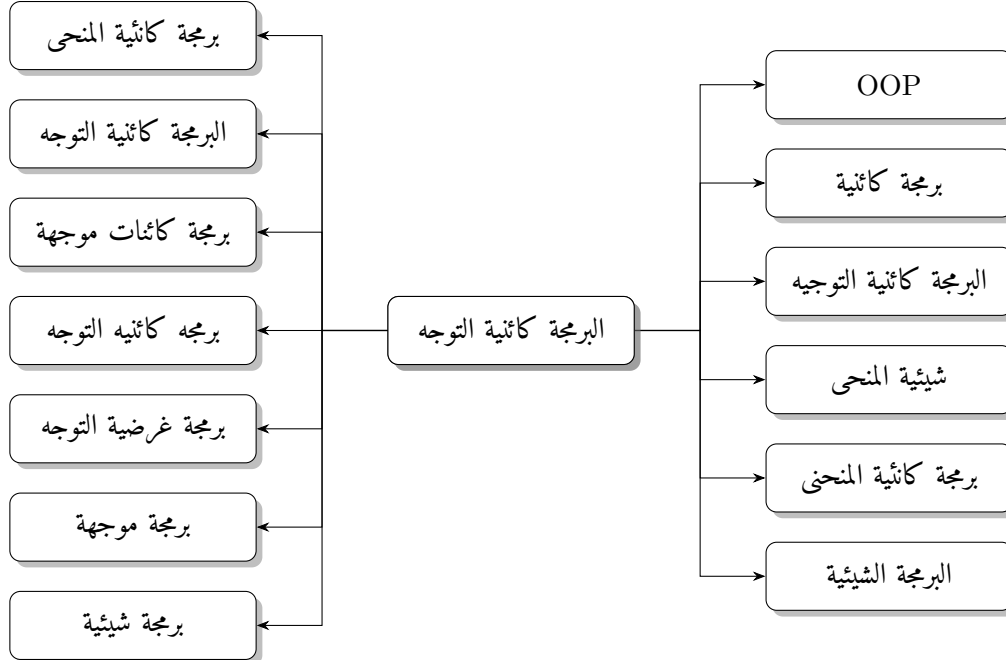


FIGURE 1.3: Aliases for Object-Oriented Programming Q79872 in Wikidata in the Arabic language.

**Challenges of Arabic language:** Arabic named entities disambiguation and linking is considered to be a challenging process. Arabic is a morphologically rich language, it has a different character set and writing rules from Latin languages. So, the techniques used for English tokenization and lemmatization are not valid for Arabic free-text. This is in addition to the issues arising with the presence of translated and transliterated entities (Elazhary, 2016). As modern text compromises translated and transliterated named entities, with non-standard spelling, for example, *جوجل*/Google is written as *غوغل*/Google , *قوقل*/Google. This applies to most of the translated technical terms. Figure 1.3 shows thirteen different aliases used in Arabic for the concept *البرمجة كائنية التوجه*/OOP - which means "object-oriented programming". This can lead to confusion if the concept is unknown, recognized by the reader, or not used consistently. The common challenge that faces the NED process of the Arabic language is the lack of resources on the knowledge bases for the Arabic language. Arabic entities in Wikidata

do not have enough coverage and quality, in a manner that is consistent with the fast-growing Arabic contact. Furthermore, the lack of benchmark datasets available for the Arabic Language comes in second. Where the lack of research that covers many knowledge graphs, especially Wikidata is also a significant reason.

### 1.2.1 Tasks Definition

In this research, the entity disambiguation (NED) process is defined as follows: given an input sentence, which is an ordered set of words  $W = w_1, w_2, w_3, \dots, w_n$  of length ( $n$ ) tokens; and given a set of named entities in  $W$  denoted by  $NE = (ne_1, ne_2, ne_3, \dots, ne_k)$ , where ( $k \leq n$ ) with their tags denoted by  $T = (t_1, t_2, t_3, \dots, t_k)$ . And given a set of entities in a KG denoted by  $\mathcal{E}$ . The NED process contains two main sub-tasks (candidate lookup and entity disambiguation) which are defined as the following:

- **Task 1:** (Candidate Lookup) Given an entity label  $ne_x$  with its tag  $t_x$ , the main objective is to return a minimum set of candidate entities  $C(ne_x) = (e_1^x, e_2^x, e_3^x, \dots, e_n^x | e_i^x \in \mathcal{E})$  from KG to possibly match with the given entity.
- **Task 2:** (Entity Disambiguation) given a named entity  $ne_x$  in  $W$  and a set of candidate nodes  $C(ne_x)$  in KG, the goal is to decide which candidate from the set of candidate nodes  $C(ne_x)$ , the entity nodes  $ne_x$  matches, i.e., same as,  $ne_x$ .

## 1.3 Research Questions

This research will review existing studies to provide in-depth details about Arabic named-entities' disambiguation and linking, including concept matching. The main goal is to advance the scientific literature with more accurate ways to disambiguate and link entities in a given context (i.e., a sentence about a software-related issue) with nodes in knowledge graphs, like, Wikidata. In order to achieve this goal, three research questions are formulated and presented in Table 1.1. The answers to these questions, which are directly linked to the objective of this research, can provide missing gaps and challenges, and contribute to resolving them.

TABLE 1.1: Research questions

#	Question
<i>RQ<sub>1a</sub></i>	Given a sentence about a software-related text (e.g., requirement, review, bug report), how understanding the semantics of this sentence improves understanding and classifying these issues?
<i>RQ<sub>1b</sub></i>	Given the lack of annotated Arabic corpora (i.e. datasets labeled with software-specific named entity classes) in the software-related aspects, what software-specific named entity classes are required to enable understanding the semantics of these named entities?
<i>RQ<sub>2</sub></i>	Given a word in a sentence (whether this word refers to a named entity or a concept, i.e., unnamed), what is the most accurate way (using e.g., state-of-the-art language models and different datasets) to disambiguate the semantics of this word with a node in a knowledge graph like Wikidata?
<i>RQ<sub>3</sub></i>	Given a named entity in a sentence (especially in software-related discussion forums), what is the best performing fine-tuned Arabic Bert model to link this entity with a node in Wikidata?

## 1.4 Research Contributions

We have presented an entity linking mechanism for Arabic-named entity disambiguation and linking of three main components (NER, candidate lookup, and the NED component)<sup>6</sup>. The key contributions of this research can be summarized as follows:

<sup>6</sup>In this thesis, the NER component was not implemented. However, as said earlier, we used Birzeit NER API named Wojood (Jarrar et al., 2022), and manually annotated Wojood-NED, part of Wojood with additional six software-specific named entities for evaluation purposes, without the need to refine-tune the NER model



- **Wojood-NED** corpus: which is part of the previously annotated Wojood corpus (Jarrar et al., 2022). We took about ~50k tokens from Wojood, those that were originally extracted from Quora (a discussion forum about software-related issues), and previously annotated using 21 classes of entities. We used this part of Wojood (we call it Wojood-NED) and enriched it with additional six software-specific classes and linked them with Wikidata using Wikidata Q-identifier.
- The candidate lookup was implemented by building a local data store from the Wikidata and Wikipedia dumps. A custom analyzer was built to index the extracted data using the Elasticsearch framework (Kuc & Rogozinski, 2013). Then, a boolean query was fine-tuned to retrieve the most accurate candidates for the named entity. The query was tested on named entities from Wojood-NED. The results show that the overwhelming majority (68%) of the correct named entities came in the first order candidate by the query.
- A corpus called **WikiGlossContext** was constructed for the sentence-pair classification task. The *True* pairs for each Arabic named-entity were extracted from the Wikidata description and the corresponding Wikipedia definition. At the same time, the *False* pairs were generated based on the *True* pairs with two different methods (cross-relate and false-local). As a result, **WikiGlossContext<sub>cross-relate</sub>** contains 1,106,408 (*True* and *False*) pairs and the **WikiGlossContext<sub>false-local</sub>** contains 1,142,280 (*True* and *False*) pairs.
- We used multiple datasets combinations (of WikiGlossContext and ArabGlossBERT (Al-Hajj & Jarrar, 2021)) in order to fine-tune different Arabic BERT models and compare between them.

## 1.5 Structure of Thesis

**Chapter 2** This chapter provides comprehensive background information and highlights the implicit basic principles and concepts that serve as the basis for addressing challenges, and provides a solid background and concepts necessary to understand the

rest of the thesis.

**Chapter 3** In this chapter, prior studies that are closely related to the named entity disambiguation and concept matching are discussed in-depth, with a concentration of studies talked about Arabic named entity disambiguation as well as those studies that targeted the software-specific named entity disambiguation and recognition to reveal the research gap.

**Chapter 4** In this chapter, the research methodology used, based on the literature review, the gap and shortcomings in Arabic studies, in particular, are addressed. Where we presented an entity linking components for Arabic-named entity's disambiguation and linking, including data processing and corpora building and annotation.

**Chapter 5** In this chapter, the candidate lookup component is illustrated, including local store building, analyzer building and data indexing, query fine-tuning, and evaluation.

**Chapter 6** In this chapter, Entity linking and disambiguation are presented. The needed experimentation to fine-tune the BERT model was conducted, and the results were reported and discussed. Moreover, the models' evaluation and selection were illustrated, and answers to the research questions are provided. Finally, the implementation of the components' APIs are provided and examples are demonstrated.

**Chapter 7** This chapter provides a conclusion, future works and threats to validity.

## Chapter 2

# Background

### 2.1 Introduction

To handle the problem of named entity disambiguation for the Arabic language in the software-related domains, and leveraging context from KGs as illustrated in chapter 1. A comprehensive and thorough approach is necessary to obtain insights that cover different perspectives. This chapter provides comprehensive background information and highlights the implicit basic principles and concepts that serve as the basis for addressing challenges. In section 2.2 we illustrate the concepts related to knowledge graphs in general and describe in detail the background knowledge and concepts about Wikidata in section 2.2.2. In addition, the state-of-the-art neural Arabic language models based on transformers will be referred to in section 2.3.

### 2.2 Knowledge Graphs

The term knowledge graph was coined in the 1980s by a group of researchers from both the [university of Groningen](#) and the [university of Twente](#) (James, 1991). The term was then used to describe their proposed knowledge-based system that combines knowledge from multiple resources for representing natural language (James, 1991; Sri Nurdianti & Hoede, 2008). Their proposed KG has a limited set of relations. Besides, their main

focus in this KG was modeling human interactions which contradict the currently accepted concept knowledge graphs (Ehrlinger & Wöß, 2016).

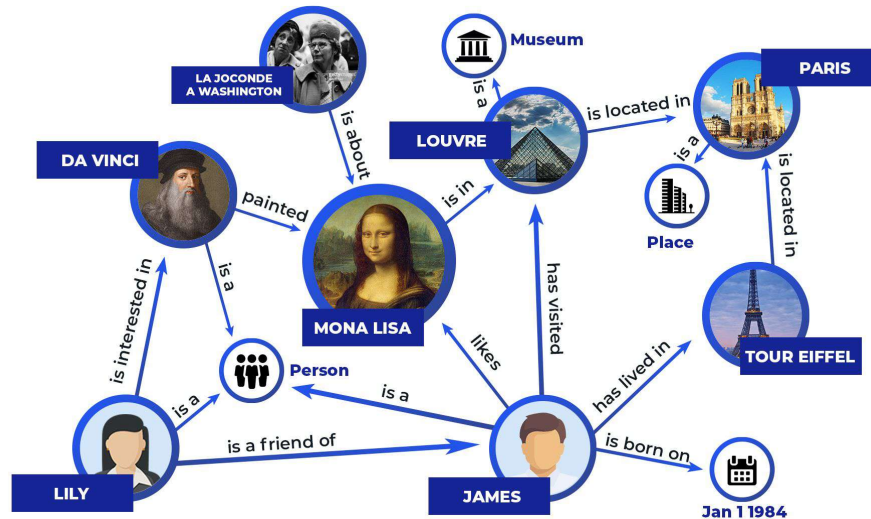


FIGURE 2.1: knowledge-graph overview (Evans, 2022)

Google introduced the concept of the modern knowledge graph definition in 2012 as a semantic improvement of the search engine feature to allow users to search for things also known as real-world objects (Singhal, 2012). Since that the term knowledge graph was associated with a group of applications, such as YAGO (Suchanek et al., 2007), DBpedia (Auer et al., 2007), Wikidata (Vrandečić & Krötzsch, 2014), Firebase, Google’s Knowledge Vault, Yahoo semantic search tool, Microsoft’s Satori, in addition to Facebook’s entity graph. Albeit these applications have major differences in architecture, purpose, and technology, they have something in common, which is, that they all use the Linked Data concept (Ehrlinger & Wöß, 2016) that emerged in 2009, and aims to link the different databases semantically to produce a large connected graph. Figure 2.1 illustrates the knowledge-graph concept and how it links entities, concepts, and their relations in a graph manner that is based on the triple (subject, predicate, object). For example, *Da VINCI painted MONA LISA*. The two entities **Da VINCI** and **MONA LISA** are linked using a property **painted**.

Knowledge graphs have attracted attention in several areas. The fact that these databases are systematically structured helps machines interpret them, and hence build more intelligent machines. The application domains of the knowledge graphs are depicted in Figure 2.2 as Xiaohan Zou assembled the different applications mentioned in the literature and then introduced them in a survey paper (Zou, 2020).

### 2.2.1 Knowledge Graphs Applications

Based on (Zou, 2020), the knowledge graphs applications domains have spanned multiple areas, this includes building recommender systems. Recent studies have begun to consider KGs as a source of collateral information. Relationships with different types of knowledge graphs help amend the accuracy of the recommender and raise the variety of the recommended items. They also offer the possibility of interpretation for recommendation systems. Generally, most of the present ways for constructing KG-based recommendation systems are embedding-based and path-based approaches. Information retrieval, besides they are efficient in question and answering systems, where bots replace humans such as XiaoIce (L. Zhou et al., 2020), Microsoft Cortana for Windows, and Apple Siri. The need for machines to understand the semantics of the words and the intent of the query is vital for the success of such QA systems. Knowledge bases such as Freebase and Dbpedia are used by researchers for QA systems using many techniques such as semantic parsing, information retrieval, and embedding-based. Recently researchers shifted toward deep learning and neural networks.

In another aspect, many domain-specific applications made use of the knowledge graph semantic structure and start to draw attention in many fields such as cybersecurity, biomedical, and financial to mention a few. Besides, the applications of knowledge graphs extend to other applications such as social media and geoscience. Unlike the health care domain. Although in the field of security, knowledge graphs are important to identify, predict and detect attacks and fraud. Some researchers concentrate on

building cybersecurity knowledge bases (Jia et al., 2018; Qi et al., 2018), but more research is needed on how to make use of the knowledge graphs reasoning to update KGs with new findings in this domain.

In the field of information retrieval, large companies have taken the lead by taking advantage of knowledge graphs to improve the services of search engines for their services, for example, Google uses Google knowledge graph and Facebook carries out entity queries over graphs. KGs that contain structured and linked information about real-world entities help improve the understandability of the query's semantics to better interpret and retrieve the correct results based on these facts. There are many potential methodologies to use the semantics of KGs in various components, such as query and document representation, besides the search engine ranking.

### 2.2.2 Wikidata

Wikidata (Vrandečić & Krötzsch, 2014) is a community-driven knowledge graph that was launched in October 2012 by the Wikimedia group. Wikidata is an editable, free multilingual that assembles an enormous amount of structured data to provide support for Wikipedia, Wikimedia, etc. At first, the Wikidatas' functionality was limited, as editors are only authorized to add items and link them to Wikipedia articles. But, in January 2013, three Wikipedias, Hungarian, Hebrew, and Italian, started to connect with Wikidata. Meantime, the community also communicates to Wikidata by contributing more than three million items. The English Wikipedia followed in February. While all Wikipedia editions were linked to Wikidata in March. As of July 2020, Wikidata has approximately 87 million structured data articles across diverse domains. 73 million elements are interpreted as entities because of the presence of an `is_instance` property. Wikidata is huge compared to DBpedia, as the latter only has around 5 million entities (Pellissier Tanon et al., 2020). The `is_instance` property comprises a much wider range of entities than DBpedia. Wikidata has about 8.5 million persons whereas DBpedia has only about 1.8 million (October 2020). Therefore, the difference



FIGURE 2.2: Knowledge graph applications.(Zou, 2020)

in size is clear (Möller et al., 2021).

Recently, Wikidata has gained publicity as a target database for linking entities (Kolisas et al., 2018; Sorokin & Gurevych, 2018a; Weichselbraun et al., 2018). Figure 2.4 illustrates the fast-growing community of Wikidata, as the sub-figure 2.4a shows the size of active editors (editors with five or more edits) during the period from Jan 2012 until Oct 2021 with an average of 11K edits. While the sub-figure 2.4b shows the

number of new pages created during the same period with 96M as average. Wikidata stores data in a collection of items, each item has a unique identifier in the form of **Q-identifier** e.g (Q34211), a **label** that stores the name of the entity, a **description** about the item to disambiguate it from namesakes, and **aliases** alternate names for the entity for each language. The figure <sup>1</sup> 2.3 illustrates items structure.

The screenshot displays the Wikidata interface for item Q251, 'Java'. At the top, the item is identified as 'Java (Q251)' with a red box around the Q-identifier and an arrow pointing to 'Item Identifier'. Below this, the item is described as an 'object-oriented programming language'. The interface is divided into several sections:

- Languages:** A table showing the item's labels in different languages. The English label is 'Java' and the Arabic label is 'جاڤا'.
- Description:** The description in English is 'object-oriented programming language', and in Arabic, it is 'لغة برمجة كائنية التوجه'.
- Aliases:** A section for 'Also known as' with the Arabic label 'الجاڤا'.
- Statements:** A list of statements about the item, including:
  - Instance of: multi-paradigm programming language (Item: Q12772052)
  - Property: P31: JVM language (Item: Q56062429)
  - Subclass of: programming language (Item: Q9143)
  - Property: P279: programming language (Item: Q9143)
  - Part of: Java platform (Item: Q1713118)
- Site Links:** A list of links to external resources, primarily Wikipedia in various languages. The Arabic entry 'جاڤا (لغة برمجة)' is highlighted with a red box.

FIGURE 2.3: Data structure of item Q251 in Wikidata.

The Wikidata knowledge graph is not easily accessible by end-users because they need knowledge of query languages such as SPARQL and a deep understanding of the underlying structure of the KGs. Wikidata provides a publicly SPARQL endpoint (a web service that accepts SPARQL queries). SPARQL is the W3C standard query language of knowledge graphs represented in RDF. A SPARQL endpoint is an interface that users (human or application) can access to query an RDF dataset using the SPARQL query language. For human users, this endpoint could be a standalone or web-based application. For an application, this endpoint takes the form of a set of APIs that a

<sup>1</sup>The figure is taken from <https://en.wikipedia.org/wiki/Wikidata>





FIGURE 2.4: Wikidata growth (Group, 2022)

connected agent can use. In fact, a SPARQL Endpoint is an HTTP presence point capable of receiving and processing SPARQL PROTOCOL requests. SPARQL is an HTTP-based protocol to perform SPARQL operations against data over a SPARQL endpoint. HTTP payloads are sent using GET, POST, or PATCH methods.

### The Wikidata Query Service

The Wikidata SPARQL query service (WDQS) <sup>2</sup> was published in 2015 as a service on the main Wikidata website. This service was built on top of both the graph database and the BlazeGraph <sup>3</sup>. The service is used to query the Wikidata KG live with latency near 60 sec (Malyshev et al., 2018b). It provides a well-documented query help <sup>4</sup>, and a set of predefined and helpful examples about SPARQL queries. In addition, it provides a query builder tool integrated into WDQS, that helps people with no experience in SPARQL language to query the Wikidata easily and fast. It also provides a feature of visualizing the data, listing properties, producing short URLs of the query page, and exporting results in multiple formats i.e CSV, to mention a few. Figure 2.5 shows the WDQS use interface (UI) <sup>5</sup>.

<sup>2</sup><https://query.wikidata.org/>

<sup>3</sup><https://blazegraph.com/>

<sup>4</sup>[https://www.wikidata.org/wiki/Wikidata:SPARQL\\_query\\_service/Wikidata\\_Query\\_Help](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/Wikidata_Query_Help)

<sup>5</sup>Live example of the mentioned query can be found using the following URL: <https://w.wiki/4Ccz>



FIGURE 2.5: Wikidata SPARQL query service UI with an example query

## 2.3 Neural Arabic Language Models

The Transformer is a newly emerged novel architecture in NLP. It was introduced to solve the problem of sequence to sequence classification problem. It is based on the self-attention mechanism and was introduced by the authors Vaswani et al. in 2017, in their popular paper "Attention is all you need". In this section, Arabic versions of transforms based on BERT (Devlin et al., 2018) will be overviewed.

### 2.3.1 BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformer. A new language representation model was published recently by Google AI researcher team (Devlin et al., 2018). Unlike other NLP models (Peters et al., 2018; Radford et al., 2018) which are unidirectional language models. BERT bridges the limitation gap in these models by introducing an NLP model that looks at the sentence from both sides (left

and right). For instance, the OpenAI GPT model (Radford et al., 2018) represent the sentence from left-to-right direction only. In this architecture, each token can attend to the previous tokens only in the transformer (Vaswani et al., 2017) self-attention layers. This used architecture is suitable for fine-tuning sentence-level downstream tasks and could be unwholesome for fine-tuning downstream token-level tasks.

### **BERT Architecture**

The BERT architecture is based on the original transformers (Vaswani et al., 2017) implementation with only an encoder stack of layers. The BERT has mainly two models of architecture:

1. BERT base: Identical in size with the OpenAI GPT for performance comparison objectives. It contains 12 stacked decoder layers, with 768 hidden layers, and 12 self-attention layers, with 110M total parameters.
2. BERT large: a huge model that contains 24 stacked decoder layers, with 1024 hidden layers, and 16 self-attention layers, with 340M total parameters.

### **BERT Input/Output**

As BERT was designed to handle many downstream tasks, to support this, the input to the BERT model can handle one sentence and tuple of sentences, i.e (Question, Answer) in one token sentence. BERT uses word piece embedding (Wu et al., 2016) along with 30,000 token vocabularies. In which the first token in the sentence is a special classification token ([CLS]) and it is used in the final hidden representation for classification tasks. While pair of sentences are separated by a special token ([SEP]). Each sentence in the pair of sentences is distinguished using a learned embedding associated with each token in each sentence to indicate to which sentence the token belongs. Hence, for every token, its input representation is calculated using the summation of the corresponding token segments and the position embedding. Figure 2.6 illustrates the input representation process.

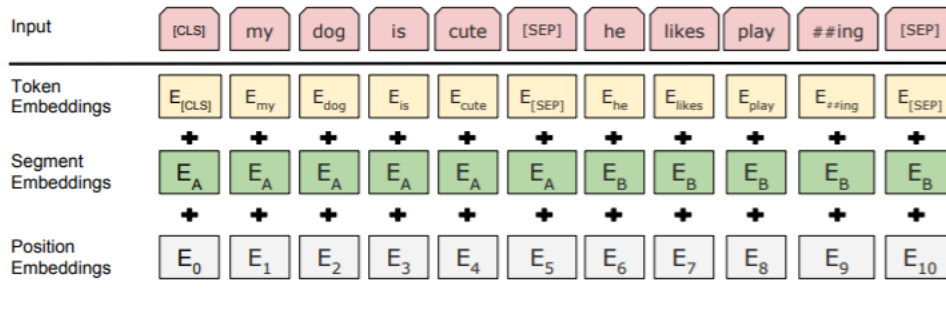


FIGURE 2.6: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings (Devlin et al., 2018)

## BERT Pre-training

BERT model was pre-trained on two unsupervised downstream tasks (illustrated in first part of Figure 2.7).

1. **Task #1: Masked Language Model (MLM)** In this task, 15% of all the tokens in a sequence were masked by a special token [MASK] at random. To overcome the problem of fine-tuning and training of the model, the [MASK] token is used 80% of the time, while the rest is replaced by a random token 10% of the time. The rest 10% are kept unchanged.
2. **Task #2: Next Sentence Prediction (NSP)** BERT was trained on a corpus that has 50% of the entries are labeled with (isNext) and 50% are labeled with (NotNext), these later sentences were chosen randomly from the corpus. This is essential in some downstream tasks that need understanding the relation between two sentences such as question answering (QA) and natural language inference (NLI).

The BERT model was trained on the BookCorpus (Zhu et al., 2015) that have 800M words, and the English Wikipedia (2,500M words) extracted from free-text only. Tables, lists, etc are ignored.

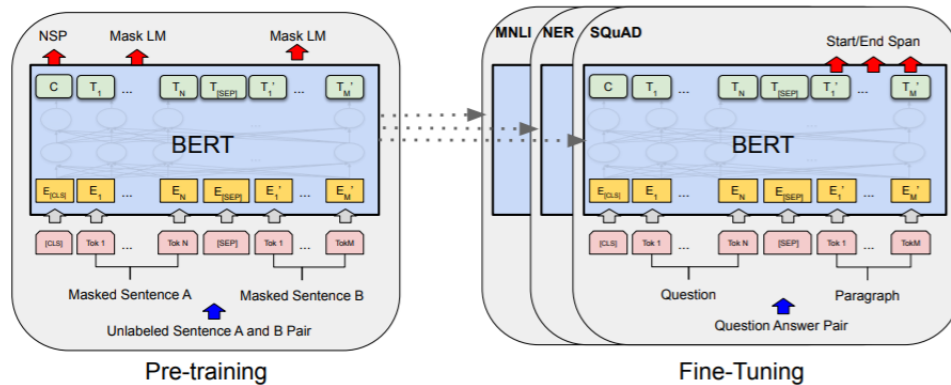


FIGURE 2.7: Overall pre-training and fine-tuning task for BERT (Devlin et al., 2018)

### BERT Fine-tuning

Due to the BERT architecture that is mainly based on transformers, BERT can be easily fine-tuned to accomplish many downstream tasks for natural language processing for tasks involving single sentence or sentences-pairs. The self-attention layer is used to encode a concatenated self-attention effectively. These downstream tasks include:

1. Classification tasks such as sentiment analysis. These tasks are similar to the next sentence classification, in the output layer, a classification layer is added for the [CLS] token.
2. Question Answering (QA) tasks. The BERT model takes two sequences of sentences and is required to mark the answer in the second sequence.
3. Named entity recognition (NER) tasks, such as classifying the tokens of a given sequence into predefined classes such as ( date, person, location, etc), the output vector of every token is fed into a classification layer to predict the NER label.

#### 2.3.2 BERT Models for Arabic

In this subsection, five commonly used Arabic BERT models were overviewed.

### **BERT multilingual**

Multilingual BERT (Devlin et al., 2018) is a pre-trained model on top of 104 languages, including the Arabic language, and was introduced by Google research AI. The model was trained on the largest Wikipedia (MLM) task. In this way, the model learns the internal representation of languages in the training dataset to be used to extract useful features for downstream tasks.

### **AraBERT**

AraBERT (Antoun et al., 2020) is a pre-trained model specifically for Arabic language. This model is based on the same architecture in the BERT base model (Devlin et al., 2018), with 12 decoders, 12 attention heads, 768 hidden dimensions, and 110M parameters. Due to the morphological nature of the Arabic language, an additional pre-processing step is done.

In the same manner, the AraBERT model was trained on the MLM task for 15% of the input tokens were selected for replacement. From these, 80% of the tokens were replaced with [MASK], while 10% were replaced randomly, and 10% kept the same. Besides, the NSP task was also employed to help the model understand the relation between the sentences, which is vital for many downstream tasks such as QA.

To overcome the small Arabic Wikipedia dump compared to English, the authors manually scraped news articles from websites. Besides, they used public Arabic corpora, the first contains 1.5 billion words (El-khair, 2016), with 5 million articles, these articles are extracted from 10 popular websites and cover 8 countries. The second is OSIAN (The Open Source International Arabic News Corpus) (Zeroual et al., 2019) and contains 3.5 million articles that span 31 news sources from 24 Arabic countries. In total, the corpus size after eliminating duplicates is 70M sentences. Compared to multilingual BERT, AraBERT achieved state-of-the-art results for many downstream tasks and was much smaller in size (300MB).

### QARiB

The authors Abdelali et al. trained 5 BERT models that differ in size from their training sets, a mixture of formal and dialect Arabic models. The goal is to develop Arabic dialects and social media support in addition to MSA. QARiB models achieved good results in many tasks. The model is trained using two corpora for modern Arabic and dialectal Arabic:

1. **Arabic Gigaword 4th Edition 2:** contains 9 Premium news sources. The total data consists of 2.7 million documents and 32 million sentences (1B words).
2. **Abu Al-Khair Group** (El-khair, 2016): It was scrapped from news websites from various Arab countries during the period December 2013 and June 2014. The corpus comprises 58 million sentences (1.5 billion words).
3. **Open Subtitles** (Lison & Tiedemann, 2016): An aggregate collection of 2.6 billion sentences from a huge database of movie and TV subtitles in over 60 languages. The Arabic part of the collection was chosen, which compromises 83.6 million sentences (0.5 billion words). Statements are mostly conversational, and sections of text are usually smaller in length than sentences in articles.

### CAMeLBERT

CAMeLBERT (Inoue et al., 2021) is a multivariate NLP model to study the effect of corpus size and fine-tuned tasks in the Arabic language. To achieve this goal, the authors build four models like the following:

1. A model for the modern standard Arabic.
2. A model for dialectal Arabic.
3. A model for classical Arabic.
4. A model that combines data from the three models.

The importance of the volume of pre-training data was investigated by developing further models prior to training for accurate aggregation of the MSA variant. The different variants of the model were compared against each other, as well as eight publicly available ones, and turned into five NLP tasks covering 12 datasets. The authors took advantage of this insight to define the optimal system. Their results showed that the AraBERTv02 (Antoun et al., 2020) excels on average and prevails in six out of 12 sub-tasks. The CAMELBERT-Star model came in second place overall on average, as it prevails in four out of 12 sub-tasks.

### **ARBERT & MARBERT**

MARBERT (Abdul-Mageed et al., 2021) is an extensive and persuasive pre-training language model that focuses on both the Arabic dialect (DA) and Modern Standard Arabic (MSA). To train MARBERT, the authors tested 1B of Arab tweets from a large internal data set of approximately 6 million tweets. They included tweets with at least 3 Arabic words, based on string matching, regardless of whether the tweet contains a non-Arabic string. That is, there are no restrictions on the Arabian Sea only. The dataset contains 128 GB (15.6 Million) of texts. They used the same network architecture as ARBERT (Bert's Rule), but without the Next Sentence Prediction Goal (NSP) because the tweets were short.

## **2.4 Summary**

In this chapter, the needed background knowledge related to this thesis was reviewed. In section 2.2 the knowledge graph was reviewed in general, its definition as stated in the literature, and its most important applications in recent research. After that, a background about Wikidata KG is presented. The services provided by Wikidata such as the query service GUI interface are presented. While in section 2.3, we reviewed the new emerging approaches for Arabic NLP such as BERT Arabic distributions.



## Chapter 3

# Literature Review

### 3.1 Introduction

Although entity linking with KGs is getting a lot of attention in many application domains, it is still emerging in the software-related text such as requirements, bugs, test cases, to mention a few. Available studies focus on NER as the first step in the entity linking process. Moreover, most of the research done in software-related data mining focuses on getting insights, such as extracting functional and non-functional requirements from app reviews. Besides, classifying bugs from the bug repositories, building NER tools that extract some domain-specific named entities such as app version, API, programming languages, etc. But, to our knowledge, there are no studies that tried to continue the task of the entity linking (EL) by linking the extracted entities to their matching nodes in different KGs (i.g. Wikidata ). Because of the lack of studies that talked about named entity disambiguation for software-specific named entities, we expand our literature review to include studies that have mentioned NER for software-related text, such as bugs identification, and cybersecurity. Moreover, we include studies that mentioned named entity disambiguation or linking in the cultural heritage for both English and Arabic languages without any limitation to the target KG.

As a start, section 3.3 summaries the state-of-art approaches for NER studies for

software-specific named entities. While section 3.4 shed the light on approaches followed by the research community for named entity linking and disambiguation. In sections 3.5, and 3.6, concept matching and word sense disambiguation summaries are provided in a nutshell respectively.

Apart from the application domain, there are several techniques and methodologies followed by researchers to advance the state-of-art in the field of named entity disambiguation and recognition. In this literature review, the related work is classified based on the approach used into five main categories: Rule-based approaches (Section 3.4.1), Word embedding (Section 3.4.2), Neural-based approaches (Section 3.4.3), Hybrid approaches (Section 3.4.4) and Statistical approaches (Section 3.4.5).

This chapter provides a targeted synopsis of the state-of-the-art techniques linked directly to the research problem and questions described in Chapter 1. Prior studies that are closely related to the named entity disambiguation and concept matching will be discussed in-depth, with a concentration on Arabic named entity disambiguation as well as those studies that targeted the software-specific named entity disambiguation and recognition. With an aim to reveal the gap in the previous studies regarding Arabic NED in the SE field.

At the outset, the first section(3.2) discusses the successive stages of this research, as these stages show exactly how the literature review was conducted to reveal the research gap.

## 3.2 Planning and Conducting The Review

This section is intended to define the guidelines and protocols for how the literature review will be conducted. Some steps are adopted by Kitchenham and Charters because it provides a well-defined process for identifying, evaluating, and interpreting relevant studies available for a given set of research questions. It facilities to summarize the

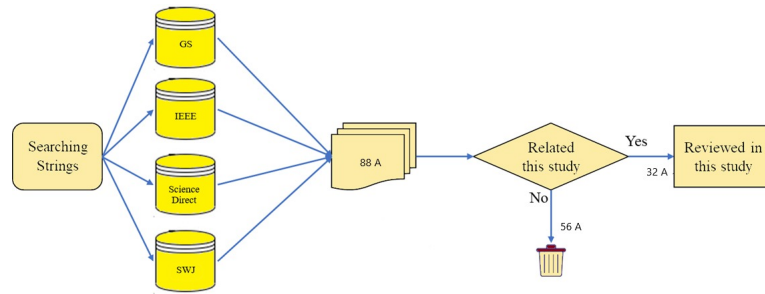


FIGURE 3.1: Literature review methodology

existing evidence concerning a technology, identify gaps in current research in order to suggest areas for further investigation, and provide a framework/background in order to appropriately position new research activities.

The objective of this study is to draw the landmarks of Arabic-named entity disambiguation in the software-related texts as an emerging research area and to conduct a thorough overview of the work researchers has done so far. The review protocol designed for this study (figure 3.1) consists of a search strategy, criteria for inclusion and exclusion, and highlighting missing gaps. The strategy for collecting relevant research is composed of two main elements: search terms and data sources. The potential relevant research was collected from many scientific digital databases and journals. Table 3.1 illustrates the search resources used along with their URLs.

TABLE 3.1: Selected sources for review

Source	Type	URL
Google Scholar (GS)	Database	<a href="https://scholar.google.com/">https://scholar.google.com/</a>
IEEE Xplore	Database	<a href="https://ieeexplore.ieee.org/Xplore/home.jsp">https://ieeexplore.ieee.org/Xplore/home.jsp</a>
sciencedirect	Database	<a href="https://www.sciencedirect.com/">https://www.sciencedirect.com/</a>
Semantic Web Journal (SWJ)	Journal	<a href="https://content.iospress.com/journals">https://content.iospress.com/journals</a>

**Inclusion/Exclusion criteria:** The filtering of studies goes through multiple iterative stages. Where the initial stage is to collect all relevant studies by querying online databases using a well-constructed search string, after which the results are filtered again according to the relevancy of the title and abstract. For those studies identified,

the second round begins, based on the inclusion/exclusion criteria given in Table 3.2. Finally, for those papers that are selected, in-depth reading is carefully expanded to include the introduction, and the conclusion to be filtered and re-selected.

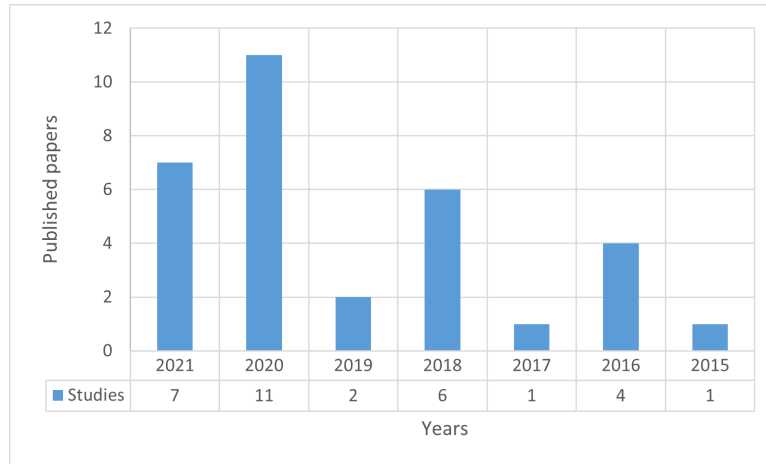


FIGURE 3.2: Published papers per year

Applying the aforementioned inclusion/exclusion criteria resulted in 32 research papers that have been classified into groups based on the approach followed. The selected related work is presented in Table 3.3. Figure 3.2 shows papers distribution per published year. According to the applied selection criteria, from the 32 studies were selected as related work, seven studies were published in 2021, eleven studies were published in 2020, two studies were published in 2019, and six in 2018, while the rest (six studies) span the period (2015-2017).

TABLE 3.3: Included related work.

#	Ref	Article
1	Bouziane et al., 2021	ARALD: Arabic Annotation Using Linked Data.
Continued on next page		

**Table 3.3 – continued from previous page**

#	Ref	Article
2	(Kannan Ravi et al., 2021)	CHOLAN: A Modular Approach for Neural Entity Linking on Wikipedia and Wikidata.
3	(McCrae & Cillesen, 2021)	Towards a Linking between WordNet and Wikidata.
4	(Al-Hajj & Jarar, 2021)	ArabGlossBERT: Fine-Tuning BERT on Context-Gloss Pairs for WSD.
5	(El-Razzaz et al., 2021)	Arabic Gloss WSD Using BERT.
6	(Esmeir, 2021)	SERAG: Semantic Entity Retrieval from Arabic knowledge Graphs.
7	(Al-Hajj & Jarar, 2021)	ArabGlossBERT: Fine-Tuning BERT on Context-Gloss Pairs for WSD.
8	(Makris et al., 2020)	Text Semantic Annotation: A Distributed Methodology Based on Community Coherence.
9	(Tabassum et al., 2020)	Code and Named Entity Recognition in StackOverflow.
10	(Sakor et al., 2020)	Falcon 2.0: An entity and relation linking tool over Wikidata.
11	(Banerjee et al., 2020)	PNEL: Pointer Network based EndToEnd Entity Linking over Knowledge Graphs.
12	(Harandizadeh & Singh, 2020)	Tweeki: Linking Named Entities on Twitter to a Knowledge Graph.
13	(Botha et al., 2020)	Entity linking in 100 languages.
14	(Mulang' et al., 2020)	Encoding knowledge graph entity aliases in attentive neural network for wikidata entity linking.

Continued on next page

Table 3.3 – continued from previous page

#	Ref	Article
15	(Huang et al., 2020)	Entity Linking for Short Text Using Structured Knowledge Graph via Multi-Grained Text Matching.
16	(C. Zhou et al., 2020)	Improving software bug-specific named entity recognition with deep neural network.
17	(Bouziane et al., 2020)	Annotating Arabic Texts with Linked Data.
18	(Perkins, 2020)	Separating the Signal from the Noise: Predicting the Correct Entities in Named-Entity Linking.
19	(Li et al., 2019)	Feature-Specific Named Entity Recognition in Software Development Social Content.
20	(Delpeuch, 2020)	Opentapioca: Lightweight entity linking for wikidata.
21	(Malyshev et al., 2018a)	NERSE: Named Entity Recognition in Software Engineering as a Service.
22	(Raiman & Raiman, 2018)	Deeptype: Multilingual entity linking by neural type system evolution.
23	(Sorokin & Gurevych, 2018b)	Mixing Context Granularities for Improved Entity Linking on Question Answering Data across Entity Categories.
24	(Cetoli et al., 2018)	Named entity disambiguation using deep learning on graphs.
25	(C. Zhou et al., 2018)	Recognizing software bug-specific named entity in software bug repository.
26	(Gasmi et al., 2018)	LSTM recurrent neural networks for cybersecurity named entity recognition.
27	(Phan et al., 2017)	NeuPL: Attention-based semantic matching and pair-linking for entity disambiguation.

Continued on next page

Table 3.3 – continued from previous page		
#	Ref	Article
28	(Al-Qawasmeh et al., 2016)	Arabic named entity disambiguation using linked open data.
29	(Ye et al., 2016)	Software-specific named entity recognition in software engineering social content.
30	(Hadni et al., 2016)	Word sense disambiguation for Arabic text categorization.
31	(Alian et al., 2016)	Arabic word sense disambiguation using Wikipedia.
32	(Gad-Elrab et al., 2015)	Named entity disambiguation for resource-poor languages.

To organize this chapter, the selected related work was classified into categories based on the approach used in the entity linking process. Classifying the papers into groups helps in revealing similar and different characters of the used approach to reveal the research gap. From those papers, we distinguished five approaches for the entity linking process: 1. Rule-based. 2. Word embedding. 3. Neural. 4. Hybrid. 5. Statistical Models. The majority of the research follows the neural-based approach.

### 3.3 Named Entity Recognition in Software-related Text

In this section, literature about software-specific named entities recognition was reviewed. Although each one of included studies focused on a specific domain in the software development life-cycle, such as bugs and testing phase (C. Zhou et al., 2020; C. Zhou et al., 2018), development phase (Li et al., 2019), security (Gasmi et al., 2018). But they all intersect with many types (or called tags) of named entities. These tags include but are not limited to Platform, Programming language, Software Standard,

TABLE 3.2: Inclusion and exclusion criteria

Criteria	Description
I1.	A study that investigates NER/NED in the software-related domains, or NED on the domain of cultural heritage was included.
I2.	In addition to I1, a study of type empirical research, such as case studies, surveys, experiments, and ethnographic studies is included.
I3.	Studies that investigate concept matching or ontology to ontology linking are included.
I4.	Studies that have parts or relies on previous studies, only recent studies were included.
I5.	Studies that investigate Arabic word sense disambiguation based on the transformers approach are only included.
E1.	Studies with only abstracts are excluded.
E2.	Studies with less than five pages were excluded.

I: stands for Inclusion Criteria.

E: stands for Exclusion Criteria.

and Framework. Table 3.4 presents a comparison between the related literature in terms of the target language, the model type, the number of tags, the availability of the code, and the availability of API. The majority of the included studies used hybrid approaches by mixing more than one neural model such as LSTM with CRF (Malyshev et al., 2018a) and word2vec with BiLSTM (Li et al., 2019). Only one study (Tabassum et al., 2020) used the BERT model. The number of tags (classes) varied, and some of the studies have targeted the high abstract software-specific named entities such as programming language, platform, version, and so such as (Gasmi et al., 2018; Li et al., 2019; Ye et al., 2016). While others (Malyshev et al., 2018a; Tabassum et al., 2020; C. Zhou et al., 2020; C. Zhou et al., 2018) include deep software named entities at a low level of granularity such as functions, OOP-public methods, OOP packages, etc.

Figure 3.3 demonstrates an illustrative example of what NER tagging is, as in this example the word Instagram is extracted and tagged with Application, the same is for the word Python which is tagged as a programming language, while Django is tagged with a framework.

It is worth mentioning that all the reviewed studies that discussed extracting NER



Instagram Application is said to be another example of websites that are mainly built with both Python programming language and Django Framework .

FIGURE 3.3: NER example

tags in the English language, while we could not find similar NER studies in the Arabic language for extracting software-specific named entities. These studies constitute a starting point to extend their work to support the Arabic software-specific named entities extraction and continue the task of linking named entities with knowledge graphs.

TABLE 3.4: Software-specific NER related work

Ref.	Lang.	Model	#Tags	Code	API
SoftNER(Tabassum et al., 2020)	en	BERT	20	✓	✗
BNER(C. Zhou et al., 2020)	en	BiLSTM-CRF	16	✗	✗
SFF (Li et al., 2019)	en	word2vec,BiLSTM	5	✗	✗
DBNER(C. Zhou et al., 2018)	en	CRF, word embeddings	16	✗	✗
Gasmi et al. (Gasmi et al., 2018)	en	CRF,LSTM-CRF	7	✗	✗
S-NER(Ye et al., 2016)	en	CRF	5	✗	✗
NERSE(Malyshev et al., 2018a)	en	CRF,LSTM-CRF	22	✓	✗

en: English.

NERSE (Malyshev et al., 2018a) is a service that includes deep software named entities at low granularity level, with an aim to enrich the literature with more tags compared to other studies that are limited to the higher categories of software named entities (Gasmi et al., 2018; Li et al., 2019; Ye et al., 2016). To this extent, the authors rely on StackOverflow as an important source for software developers to categorize the content into 22 predefined software-specific named entities tags (*Object Oriented, Procedural, Scripting, Web Development, Other Types i.g. SQL, CPU instruction sets, Hardware architectures, Operating systems, OOP-packages, OOP-public methods, Non-OOP-Functions, Other-Events, Software tools, Software framework, Software libraries, software applications, Data formats, Standard Protocols, Software design patterns, Software acronyms, Software Roles, Software Operation*). This was done using NLP and ML techniques, using Conditional Random Fields (CRF) and Bidirectional Long Short-Term Memory - Conditional Random Fields (BiLSTM-CRF). They used the precision,

recall, f1-score evaluation metrics, and achieved (0.87, 0.82, 0.85) respectively for the CRF model, while they achieved (0.96, 0.94, 0.95) respectively for the BiLSTM-CRF model.

**BNER** (C. Zhou et al., 2018) is a bug-specific entity recognition system. The BNER system is developed using CRF and word embedding techniques. The BNER system makes use of the huge information found on bug repository systems that are used to track bugs, e.g., Bugzilla. These bug tracking systems can be used as centric knowledge information that can then help understand and fix these bugs. The authors of the BNER system classify bugs into three predefined main categories (component, specific, and general). The component category has seven subcategories (core, GUI, network, IO, driver, file system, and hardware), according to (Tan et al., 2014), while the specific category has five subcategories according to (Ye et al., 2016). Finally, the general category has four subcategories (defect test, common adjectives, common verb, and mobile). The authors extracted four types of features (contextual feature, gazetteer feature, orthographic feature, and embedding feature). To evaluate their system, the authors established a baseline corpus on two open-source projects, Mozilla (1400 bug reports) and eclipse (400 bug reports). The datasets were evaluated using the three metrics (precision, recall, and f1-score) with overall accuracy that reached more than 70% to more than 80%.

The authors of the study (Gasmi et al., 2018) presented a domain agnostic NER model that is independent of any domain-specific entities and applied this model to the cybersecurity field. Compared to other studies (Malyshev et al., 2018a; C. Zhou et al., 2018), this model does not need domain experts to implement feature engineering. The proposed model relies on long-short term memory LSTM and conditional random fields (CRFs). The evaluation of the NER model was limited to seven entity types (vendor, application, version, file, operating system, hardware, and edition) that are closely related to the cybersecurity domain. Their model achieved 2% better accuracy

of the CRFsuite (Okazaki, 2007), which is one of the most accurate CRF tools.

**S-NER** (Ye et al., 2016) is a named entity recognition system that is designed to classify a broad category of software entities for a broad range of widespread programming languages, platform, libraries, API, and software standards using semi-supervised machine learning CRF model. The dataset was constructed from StackOverflow posts, labeled and validated manually using annotators with the appropriate experience. The brown clustering technique is also used on the StackOverflow unlabeled data to assign words within the same context into the same clusters. To evaluate the **S-NER** system, three metrics were used (precision, recall, and f1-score) and the results were compared to a baseline system that was implemented using rule based approaches, a mix of empirical lexical rules and gazettters. The **S-NER** system outperforms the baseline by 30.3% , with overall f1-score 78.176%.

Later on, C. Zhou et al. extended their previous work (C. Zhou et al., 2018) by providing an enhancement to the **BNER** by adopting deep learning (attention-based BiLSTM-CRF) in combination with domain features such as POS and gazetteer feature into a new system that they called **DBNER** (C. Zhou et al., 2020). This new architecture helped them extract semantic information related to bug reports with a little feature engineering. On the other hand, they extend the corpus from the first study to include additional software projects, rather than Mozilla and Eclipse. Apache and Kernel were added to achieve more comprehensiveness. The **DBNER** was evaluated using the aforementioned four open-source projects. Compared to **BENR**, the precision, recall, and f1-score increased by (4.5%, 4.15%, and 4.07%) respectively, while for the cross-project experiment, the **DBNER** has achieved an increase in the f1-score by 5.8%.

**SFF** (Li et al., 2019) is a Software Function Feature specific named entity recognition tool over the CSDN (Chinese StackOverflow). They built web crawlers to scrape the HTML pages from the CSDN and then processed the crawled pages to extract text from

the questions and their answers. They manually annotated the dataset with six types of entities ( Programming language, Platform, API, tool library framework, software standard, and undefined function). The next stage is to process the extracted text into a shortened sentence that has the SFF particular entity, then, slice the sentence into words, get the words embedding using word2vec for each word. Finally, in the last stage, the BI-LSTM model is trained to understand the boundary of the SFF-specific entity. To solve the problem of Out-of-Vocabulary (OOV) words that are not in the starting dataset, the authors used deep learning techniques to train the model. The SFF-NER system was evaluated using three metrics (precision, recall, f1-score). The overall results of the system are (74.805%, 69.019%, 71.702%) respectively.

While **SoftNER** (Tabassum et al., 2020) is a NER system for computer programming entities. This system is built on BERT and trained on StackOverflow groups that have 15,372 sentences annotated with 20 fine-grained entity types (8 code-related entities, 12 natural language (NL) entities) code entities are( class, variable, inline code, function, library, value, data type, and html xml tag). while the nl entities contain ( application, ui element, language, data structure, algorithm, file type, file name, version, device, os, website, and user name). To address the problem of ambiguity in the software domain, many software terms may conflict with concepts or English terms. For example, the word 'go', means a programming language in the software domain or an English verb in other contexts. The authors designed the SoftNER model architecture in a way that leverages the sentential meaning and domain-specific entities using BERT embeddings as an input layer to extract embeddings. In addition to domain-specific embeddings for each named entity in the sentence. Then, an extra attention layer adds the three embeddings from the first layer for the named entity together using the attention mechanism. Finally, a linear CRF layer is used to predict the entity class using the embedding emitted from the previous layer. The **SoftNER** was trained on 152 million sentences from StackOverflow which is relatively large and diverse compared to other NER datasets that are either small or have limited named entity types. Overall,

The **SoftNER** achieves a 79.10% f1 score on StackOverflow and 61.08% f1 score on GitHub data on 20 software-related named entity types.

## 3.4 Named Entity Disambiguation and Linking

In this section, studies related to named entity linking and disambiguation are included.

### 3.4.1 Rule-based Approaches

Rule-based Approaches depend mainly on human hand-crafted rules, which are widespread in machine learning (Bringmann et al., 2011). They are used to find the common pattern aka regularities in data using the IF-THEN approach (Fürnkranz, 2013). In what follows, the studies that used this approach are reviewed.

**Falcon 2.0**<sup>1</sup> (Sakor et al., 2020) is a joint entity and relation linking tool over Wikidata that was introduced by Sakor et al. as an enhancement over **Falcon 1.0** (Sakor et al., 2019) tool that was developed to link entities over DBpedia. Falcon 2.0 tool extracts entities and relations for English short statements especially questions and provides a ranked list of the corresponding nodes over Wikidata. This tool is built on the previous Falcon tool using a rule-based approach. At the recognition phase, **Falcon 2.0** uses three modules, POS for tagging, tokenization, and compounding, then, N-Gram tiling. While, at the linking phase, the candidate list is produced to be matched and linked to the proper node on Wikidata using relevant rule selection and N-Gram splitting. **Falcon 2.0** outperforms **OpenTatoica** (Delpeuch, 2020) on three different datasets (**LC-QuAD 2.0**, **SimpleQuestion**, and **SimpleQuestion Uppercase Entities**) using three metrics (**precision**, **recall** and **F-score**). On the other hand, since Falcon 2.0 is designed for English short statements, it is thus not applicable for large noisy text rather than short questions. Besides, its independence from the changes made in the knowledge graph.

---

<sup>1</sup><https://labs.tib.eu/falcon/falcon2/>

Another study (Bouziane et al., 2021) used the rule-based approach to link Arabic-named entities with DBpedia. This is done using NLP techniques such as tokenization, normalization, speech, POS tagging, parsing for the input text, and machine learning to do the NER task. The developed system consists of three main modules: (a) candidate resources module, (b) semantic module, and (c) disambiguation module. The input text is first normalized to correct typing errors. The named entity recognition model is trained using the SVM (support Vector machine) a classical machine learning algorithm using the dataset and the tags **WikiFANEgold** from the study (Alotaibi & Lee, 2014) which is extracted from Wikipedia Arabic pages. The **WikiFANEgold** dataset contains 34,483 sentences and 114,632 words with 100 tags in the two-level taxonomy. The overall system performance was measured using a 100-sentence corpus which was manually annotated with DBpedia links, and the system achieved ( 0.91, 0.78, and 0.84) for the precision, recall, and f1-score respectively. This approach is based on DBpedia KB ( 5 million entities) which is relatively small compared to Wikidata ( 98 million entities). Besides, rule-based approaches often have a lower recall, and depend on many rules which are hard to list and need domain experts (Pellissier Tanon et al., 2020).

### 3.4.2 Word Embedding Approaches

Word embedding approaches used to mathematically represents the embedding of the words. word2vec, one hot encoding, TF-IDF and Fast-text are popular word embedding methods. In this section, the studies that use this approach were reviewed.

**PNEL** (Banerjee et al., 2020) is a pointer network model or solving end-to-end entity linking over Wikidata for short text. This model is inspired by the use of the pointer network in solving the convex hull and generalizing the traveling salesman problem. This is mainly based on word embedding and n-grams techniques. The short text is tokenized into multiple combinations as entities are taken separately by itself, a combination of the entity with its predecessor, the entity with its successor, and finally, the

entity with its predecessor and successor at the same time. All of these combination are then searched using the BM25 similarity. End-to-end forms can often enhance entity recognition performance. Almost all of the entity binding algorithms used in industry often use legacy entity identification methods separate from the entity binding process. Thus, this end-to-end entity-linking approach can be beneficial. However, because of its credence on including static graphs, extensive training will be required if Wikidata changes. The model was trained on **WebQSP**, **SimpleQuestions**, and **LC-QuAD 2.0** datasets. The achieved accuracy **PNEL-L** 0.803, 0.517, 0.629 for the precision, recall, and f1 respectively.

While **SERAG** framework (Esmeir, 2021) is a Semantic Entity Retrieval from Arabic knowledge Graphs. It was introduced by Esmeir and inspired by KEWER (Nikolaev & Kotov, 2020). For every entity in a query sentence, a document is created of directly linked textual information, then a list of random walks starts at that entity. Word2Vec is then used to enhance the ranking such that a standard two-stage ranking approach was used, for a given query, the first 1000 thousand entities are selected using BM25 and then ranked using embeddings. The proposed framework was evaluated against BM25 and achieved better results in all categories (Named entity queries, IR-style keyword queries, natural language questions, and seek a particular list of entities).

### 3.4.3 Neural-based Approaches

Most of the entity-linking research studies fall under this category. 56% of these studies are published between (2020-2021). Two studies (Botha et al., 2020; Raiman & Raiman, 2018) are multilingual, the rest are proposed to solve the entity-linking for the English named entities. Only three studies (Cetoli et al., 2018; Perkins, 2020; Raiman & Raiman, 2018) were limited to the task of entity disambiguation. While the rest summarized approaches in this section proposed a jointly end-to-end solution for entity linking. Table 3.5 presents a comparison between the related literature in terms of the

target language, target KG, the encoder type, the type of the proposed tool, the availability of the code, the availability of API, the usage of label aliases and the description.

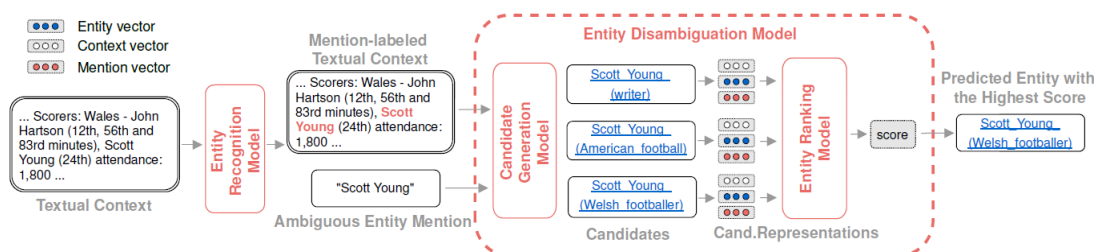


FIGURE 3.4: General architecture for neural entity linking (Alam et al., 2022).

Recently, deep learning approaches that are based on different types of neural networks have emerged as a silver bullet to the research community. These methods have shifted the performance towards a new level. The aim of this section is to provide a deep discussion of the latest generation of models that emerged from the year 2015 till the time of writing this thesis. Neural network models have attracted the attention of researchers in the field of natural language processing because of their ability to understand lexical semantics (Bengio et al., 2003; Collobert et al., 2011; Young et al., 2018). Besides, neural network-based models have shown considerable advances compared to traditional methods of machine learning (Chisholm & Hachey, 2015).

Many NLP systems use pre-trained deep language models such as Elmo (Peters et al., 2018), BERT (Devlin et al., 2018), and their different representations. Research in entity linking made benefits of these state-of-art-model as a method to introduce knowledge represented in KG, and how this helps in adopting the words in the free text to be represented to some tasks (Alam et al., 2022).

Figure 3.4 illustrates the general architecture of neural network entity linking. The pipeline is composed of two main parts, the NER model which extracts the NE from a free text, and the NED model that intern has two main parts, candidate generation to



retrieve all possible candidates of the NE from the KG, and the entity ranking model that gives these possible candidates the score, the entity with the highest score is the output from the model.

TABLE 3.5: Comparison between neural network-based approach related work

Ref.	Lang.	KG	Encoder Type	Type	Code	API	Label Aliases	Desc.
<b>CHOLAN</b> (Kannan Ravi et al., 2021)	en	Wikidata Wikipedia	BERT	EL	✓	✗	✓	✓
<b>Botha et al.</b> (Botha et al., 2020)	ml	Wikidata	BERT	EL	✓	✗	✗	✗
<b>Arjun et al.</b> (Mulang' et al., 2020)	en	Wikidata	Glove	EL	✓	✗	✓	✗
<b>Huang et al.</b> (Huang et al., 2020)	en	Wikidata	BERT	EL	✗	✗	✓	✓
<b>Perkins et al.</b> (Perkins, 2020)	en	Wikidata Wikipedia	ELMo	NED	✗	✗	✓	✗
<b>Deeptype</b> (Raiman & Raiman, 2018)	ml	Wikidata Wikipedia	Bi-LSTM CNN	NED	✗	✗	✓	✗
<b>Sorokin et al.</b> (Sorokin & Gurevych, 2018b)	en	Wikidata	CNN	EL	✓	✗	✗	✗
<b>Cetoli et al.</b> (Cetoli et al., 2018)	en	Wikidata	Bi-LSTM	NED	✓	✗	✓	✗
<b>NeuPL</b> (Phan et al., 2017)	en	Wikipedia	LSTM	EL	✗	✗	✗	✗

Ref.: Reference, Lang.: Language, Desc.: Description.

en: English, ml: Multilingual, EL: Entity Linking, NED: Named entity disambiguation.

One of the recent studies that was published in 2021 is **CHOLAN**<sup>2</sup> (Kannan Ravi et al., 2021), which is an end-to-end modular pipeline approach for entity linking. **CHOLAN** is divided into two transformers, the first distinguishes different surface forms for a specific text using the BERT model, and the second transformer is used to classify each entity mention through a predefined nominee list. This is done by feeding the second transformer with the entity mention generated from the first transformer, alongside with the local context (a sentence containing the entity-mention), and the entity Wikipedia description as an input to the second BERT transformer to disambiguate the entity mentions. To achieve the candidate generation module,

<sup>2</sup><https://github.com/ManojPrabhakar/CHOLAN>

the authors experimented with two methods. The first is using Falcon (Sakor et al., 2020) knowledge base, and the second using the component proposed by (Ganea & Hofmann, 2017). In their study, the authors linked to Wikidata, and Wikipedia. To link with Wikidata, the authors used the dataset named T-REx (Elsahar et al., 2018), by adapting part of T-REx used by Mulang' et al., 2020. It has 983,257 statements, and linked to 85,628 distinct Wikidata entities. But, for linking with Wikipedia, the CoNLL-AIDA dataset (Hoffart et al., 2011) was used. It has 8,448 linked entities from 946 documents. CHOLAN-Wikidata achieves (75, 76, 75.4) on (precision, recall, and f1-score) respectively. While CHOLAN-Wikipedia achieves (83.4, 76.8, 86.8) on (precision, recall, and f1-score) respectively

Another study (Huang et al., 2020) that aims to link entities in a short text to Wikidata. Huang et al. introduced an end-to-end entity linking approach that consists of three modules, the first is entity extraction similar to the NER system. In this module, the entity extraction is a tagging problem, the BIO schema (L. Ramshaw & Marcus, 1995) is used to tag the mentions with (**B**: at the beginning, **I**: inside, **O**: outside). They fine-tune the BERT model with a final hidden representation for each token into a classification layer based on the BIO classes. The second step in their system is the candidate search. They created Elasticsearch index from entity labels from Wikidata and for each entity, they apply Levenshtein and the exact match edit distance that is based on fuzzy matching. They also expanded the entity mention span by one token to create an error-prone entity search. Then the final search results of the adjacent mentions are added as well to the dataset. The third step is the entity ranking. In this step, three types of information about the entity mention from Wikidata are used: entity labels, description, and relations with other entities are used to rank entities via a multi-grained text matching. The multi-grained text matching process is done using many steps, the first is Character-level Similarity and achieved using CNN to measure the similarity between the entity mention and its candidate entity label on Wikidata. The result is a similarity matrix whose entire are calculated using cosine similarity.

The second is token-level similarity, in which the similarity between the text and the candidate entity Wikidata description is measured using the BERT-base model. Third, similarity based on Neighboring Entities is calculated to measure whether the entity candidate matches the text context. This is done using the BERT model to get semantic of the triple (subject, object, predict) of the entity concatenated to form a sentence. All calculated measures are accumulated and fed into two-layer perception. In this method, the Wikidata structure is integrated through the use of the entity triples in a manner that is similar to Mulang' et al. However, only one-hop of triples are used, and no hyper relational information is considered.

Unlike Huang et al., 2020 who proposed a solution for short text. The authors in (Mulang' et al., 2020)<sup>3</sup> proposed an end-to-end named entity linking over Wikidata for long, noisy titles using the Encoder-Decoder-Attention model for both entity recognition and disambiguation. The entity recognition is done using Glove while candidate generation is done in the same way in (Sakor et al., 2020). While entity ranking score is calculated using a model that takes as input the mention, entity label, and aliases. This model does not use global ranking, as the entities are matched with its Wikidata label and aliases. Thus it is not liable to KG change. The model is evaluated over the **T-REx** dataset that contains 4.65 million documents with 6.2 million sentences that are annotated using 11 million Wikidata triples. Their method achieved an 8% improvement in the performance over baseline and Opentapioca tool.

What distinguishes the tool (Botha et al., 2020) from its predecessors is that it proposed a solution for multilingual named entities over Wikidata. Their model covers 100+ languages and 20 million Wikidata entities in a corpus named **Mewsl-9**<sup>4</sup>. They used two BERT encoders. The authors make use of Wikipedia descriptions of their entities as input to the model, since, Wikidata nodes have links to Wikipedia. Wikidata

---

<sup>3</sup><https://github.com/mulangonando/Arjun>

<sup>4</sup>[https://github.com/google-research/google-research/tree/master/dense\\_representations\\_for\\_entity\\_retrieval/mel](https://github.com/google-research/google-research/tree/master/dense_representations_for_entity_retrieval/mel)

was a suitable KG since it is a Multilingual knowledge graph. The candidate generation of the mentions is done using dual BERT transformers to encode the context sensitive and the entities into the same vector space.

Perkins, 2020 as well introduced an entity linking tool over Wikidata. The NER part is done using the LSH algorithm and link frequency for entity labels. For those named entities recognized from the first step, the authors treated the entity linking as a classification problem, that is, for each candidate entity in the candidate list, the model classifies whether it is the true candidate for the named entity. This is done using deep learning techniques, as the context embedding is fed into the ELMo model, and the output is concatenated with the knowledge graph embeddings, to be then fed to a feed-forward-neural-network which intern maps the context embedding and the KG embedding as the final output.

While **Deeptype** (Raiman & Raiman, 2018) is another entity linking system on Wikidata that followed a novel approach in that it uses type information via Wikidata or Wikipedia. This system integrates symbolic information into the logic procedure of a neural network with a type system. In the first step of DeepType, the optimization is done using stochastic optimization.

Cetoli et al., 2018<sup>5</sup> used deep learning to solve the entity linking problem over Wikidata. The method takes text and graph embeddings. They used Bi-directional Long Short-Term Memory (Bi-LSTM) encoding of the graph triplets. While text embeddings are calculated by applying a Bi-LSTM over the Glove embeddings of all words in the text. It assumed that the candidate is available, so there is no candidate generation module in their work.

While in the authors in the study (Elnaggar et al., 2018) tried to solve the entity linking

---

<sup>5</sup> <https://github.com/contextscout/ned-graphs>

problem by transfer learning. They reused an open-source deep learning model that was built for similar tasks. The selection criteria of the model depend mainly on the f1-score and accuracy of the model. This model consists of three main parts. 1. the word embedding of the entities. 2. the context score is calculated based on the sentence containing the entity. 3. CRF model with parametrized potentials to disambiguate the entities. The authors defined the entity context with the left and right 100 words around the entity mention. But to avoid noise, the used algorithm only counts for the top 20 words from the entity context. The achieved f1-score is 98.90% and 98.01% on the legal small and large test datasets respectively.

Although there are many off-the-shelf named entity recognition (NER) tools freely available, Sorokin and Gurevych in their study (Sorokin & Gurevych, 2018b) targeted the question-answering (QA) text that did not find a suitable tool for short and noisy questions, this limitation motivates the authors to build entity linking tools that correspond with data needs. To do this, they made a single-stage system that performed the task of entity linking over Wikidata into a single model. The tasks of demonstrating entity recognition and disambiguation are performed in a single joint end-to-end neural network that operates at multiple context levels and does not depend on the manual features. They overcome the problem of noisy data by handling each level of granularity in a separate n-gram model. The levels include a high feature that is handled using token level using a dilated convolutions neural network model, whereas, low-level features are handled using character level dilated convolutions neural network of the entities n-gram. For each generated token, the Wikidata KG is searched to collect candidate matches. The best match is subsequently found by calculating the rating and entity score for each possible token. Their code is publicly available <sup>6</sup> as well as the datasets. Their entity linking system achieves state-of-the-art results with an 8% improvement in performance compared to previous QA systems.

---

<sup>6</sup><https://github.com/UKPLab/starsem2018-entity-linking>

**NeuPL** (Phan et al., 2017) is a deep neural network entity linking and disambiguation model. NeuPl measures the semantic matching between the context of the entity mention and the target entity by employing the long short-term memory (LSTM) and attention mechanism for entity disambiguation. Moreover, NeuPl introduces a fast Pair-Linking algorithm by scanning the pairs of mentions at most once in the whole document and takes into account positional information and word order. Therefore, two LSTM networks are used for left- and context-modeling. The model was trained on several datasets, namely, **Reuters128**, **ACE2004**, **MSNBC**, **DBpedia Spotlight (DBpedia)**, **RSS500**, **KORE50**, **Microposts2014 (Micro2014)**. The average accuracy among all datasets is 0.851%. Unlike transforms-based models such as BERT, LSTM is capable of capturing dependencies in one direction (Devlin et al., 2018).

#### 3.4.4 Hybrid Approaches

In this subsection, articles that used hybrid approaches were discussed deeply. Only two articles (Al-Qawasmeh et al., 2016; Gad-Elrab et al., 2015) have proposed NED solutions, one study (Al-Qawasmeh et al., 2016) of them targets Arabic, whereas the other is multilingual. While, only two studies have proposed an end-to-end entity linking solution (Bouziane et al., 2020). Table 3.6 presents a comparison between the related literature in terms of the target language, target KG, the approach used, the type of the proposed tool, the availability of the code and the availability of API.

Hybrid approaches combine the advantages of both rule-based and learning-based approaches. The final result in this approach is achieved by adding more than one machine learning technique together or using machine learning with handcrafted rules. Several hybrid named entity disambiguation and recognition tools were introduced.

Bouziane et al., 2020 used NLP techniques such as tokenization, normalization, and POS to implement the Arabic entity linking system to DBpedia labels and Wikidata.

TABLE 3.6: Comparison between hybrid approach related work

Ref.	Lang.	KG	Approch	Type	Code	API
<b>Bouziane et al.</b> (Bouziane et al., 2020)	ar	DBpedia, Wikidata	NLP	EL	✓	✗
<b>Tweeki</b> (Harandizadeh & Singh, 2020)	en	Wikidata	Spacy	EL	✓	✗
<b>Gasmi et al.</b> ANED(Al-Qawasmeh et al., 2016)	ar	DBpedia	NLP	NED	✗	✗
<b>Gad et al.</b> (Gad-Elrab et al., 2015)	ML	YAGO3	NLP	NED	✗	✗

en: English. ar: Arabic ML: Multilingual.

The proposed system takes Arabic text as input and combines two main modules, the first is a candidate resource that split the sentences and does tokenization and normalization of the words. Then, the candidate resources are identified from nominal phrases. Then, SPARQL query is used to retrieve the URI regarding DBpedia based on the `rdfs:label`. The system was evaluated on a small corpora (100 sentence labeled) with a precision of 0.86, a recall of 0.73, and an F-measure of 0.79. The authors did not illustrate the disambiguation technique when multiple results are returned from the SPARQL query.

Another tool that was designed for entity linking of tweets called **Tweeki**<sup>7</sup> (Harandizadeh & Singh, 2020) is an unsupervised tool for short text entity linking on Wikipedia. The ER part is implemented using a pre-existing tool (Gardner et al., 2018) that is based on a neural entity linking system (Spacy) to find a set of possible candidates in Wikidata and then find the best one based on the type’s compatibility (between NER and named entity linking (EL) systems). While the selection in the ED part was based on some rules definition. This was implemented by first creating the potential candidates and then calculating the probability of each candidate among Wikidata aliases. This is done using the Intrawiki (Ratinov et al., 2011) crawl that harvests anchor links over Wikipedia. To adjust this tool to Wikidata, the existing links between the two KGs (Wikipedia, Wikidata) are used to gather information about the entity aliases, labels and description, and the number of times the alias is used in Wikipedia. However, the module of candidate generation ignores the context of the mention in its original

<sup>7</sup><https://ucinlp.github.io/tweeki/>

tweets. To overcome this problem, the authors combine it with entity type filtering which then produces the right score probability of the link. The tool was evaluated over two datasets (TweekiData and TweekiGold).

**ANED** (Al-Qawasmeh et al., 2016) is a proposed tool for Arabic named entities disambiguation. This tool is built over an enhancement of the hybrid approach that is proposed in (Al-Smadi et al., 2015). The authors built an ANED ontology to overcome the limitation of knowledge bases that support Arabic entities linking. This ontology was designed to represent Arabic entities mainly Person, Location, and Organization. While its properties are based on Wikipedia info-box metadata and human modeling, the enhanced approach consisted of seven steps and uses query label expansion and text similarity techniques to disambiguate three types of entities (Person, Location, and Organization) over Wikipedia. The authors used contextual disambiguation, by extracting content from wiki pages and using a small window around the entity mention (3 before and 3 after), and counting the vector of term entities frequency in each candidate. Then, the highest total term frequency is selected. To evaluate their work, the **ANERcorp12** corpus was used, which is manually annotated with URIs and entity types. The overall accuracy achieved 84%. Unlike Bouziane et al.; Esmeir, the authors intruded only the NED component to disambiguate limited entity types to Wikipedia.

Another multilingual tool (Gad-Elrab et al., 2015) that targets poor resource languages in general, including Arabic. The tool consists of three steps necessary for the NED process: **NED Catalog**: The authors have used the same approach from (Yosef et al., 2014) study to build a NED methodology for the proposed language, so they combined the English and the Language under test Wikipedia's to capture famous entities from the English language with local entities of the Language under test. **Named-Entity Dictionary**: To extend the name entities of the proposed systems, the authors have proposed three approaches: **External Resources**: This is done by excerpted anchor texts that are written in the language under test using Google word-to-concept



(GW2C) repository which intern link to the Wikipedia article. Those extracted names were further automatically processed to be filtered and cleaned. **Statistical Machine Translation:** The authors developed a Type-aware machine translation to translate English entities to corresponding Arabic that is trained on the character level using a parallel set of entity names. **Entity Description:** The standard approaches were used to extract contexts Wikipedia Anchor text, Wikipedia Categories, titles, and pages linking to the entity. As most of the data is available in English, the authors used Statistical Machine Translation to translate categories. The methodology was evaluated on **LDC2014T05** and **LDC news** document. The average precision, and non-null entity precision as follows (73.23, 71.34, 94.69) respectively. The results of the LDC web dataset average precision, and non-null entity precision as follows (68.16, 60.10, 93.86) respectively.

### 3.4.5 Statistical Approaches

In this subsection, The paper that fall under this category used mathematical models.

**OpenTapioca**<sup>8</sup> (Delpeuch, 2020) is a lightweight end-to-end entity linking system that is trained on Wikidata KG, it adapts a heuristic-based model of three approaches for Entity disambiguation of named entity mentions in a text to Wikidata. The three approaches are (Local compatibility using Bayesian methodology, Topic similarity using a bag of word model, Mapping coherence using random walk. This system works on three Entity types (humans, organizations, and locations). The system was evlauated on the **RSS-500 dataset**. **ISTEX**. The height accuracy achieved is 0.870%

---

<sup>8</sup><https://opentapioca.org/>

### 3.5 Ontology Concept Matching

An ontology consists of concepts and relationships between them (Jarrar, 2005; Jarrar & Meersman, 2008). The problem of linking between two ontologies is similar to the problem of linking between two knowledge graphs. In this section, the literature related to ontology concept matching is discussed. Research that focused on linking different knowledge bases nodes that refer to the same concept.

To link WordNet and Wikidata, in their study (McCrae & Cillessen, 2021), the authors used the hapax linking and natural language processing. They focused on linking the elements on the two resources that only accrue ones in a single noun synset in the Wordnet and a corresponding unique label in Wikidata. This concept was called *hapax legomenon*. To reduce a large number of links and increase the precision, the authors took the following heuristics into consideration before evaluation. First, the Wikidata entities with **Q** number over **10,000,000** are considered less significant and unlikely to be mentioned in English WordNet. Second, All entities that have a **Wikipedia disambiguation page** or **Wikimedia disambiguation page** were filtered out. Finally, All entities whose definitions follow the pattern of 1-3 words followed by the word [*by*] and then 1-4 words were filtered out and excluded.

As mentioned earlier, the hapax linking matches only resources that do not have disambiguation, but in order to achieve the complete linking by including resources that they disambiguate, the authors extended their methodology by using the Naisc system (McCrae & Buitelaar, 2018). The first step is to collect relevant facts about the WordNet concepts, such as definitions, labels, and synset links. And from the Wikidata, the English labels, the definitions were also extracted and the links between the synset.

Then the same set of heuristics aforementioned was applied in order to prepare the data for the Naisc methodology that consists of the following steps: The hapax links were identified and accepted. The definitions were compared using the **Jaccard similarity**

at both the word-level and character-level. The similarity of each element was analyzed based on the Personalised Page-Rank PPR algorithm, using the Fast-PPR implementation. The three scores' results were normalized in the range [0,1] using percentile ranking. A simple average of the three scores (character-level Jaccard, word-level Jaccard, and PPR) was used to rank each potential match. The bijective assumption, that each entity in the WordNet has only one corresponding match in Wikidata was used. This assignment problem was solved using the Hungarian Algorithm, but due to the large dataset, a simple greedy algorithm was used. The evaluation job is assigned to two annotators to validate the 100k links predicted by their system. The agreed accuracy between them ranges from (65-66 %) with a Cohen's Kappa equals to 0.934 of the automatic linking.

The authors in the study (McCrae, 2018) aims to link concepts in the WordNet with Wikipedia articles that describes that matched concept. This was done by creating a gold-quality mapping between all concepts (7,742) in the WordNet and the Wikipedia with an aim to introduce a gold standard link discovery.

While the study (Hadni et al., 2016) used the knowledge-based method to word sense disambiguation. Their methodology starts from preprocessing a corpus text, then extracting the local context using bag-of-words (BoW), after that Words are mapped into concepts using Arabic WordNet (AWN). If the matching is found, the Arabic WSD based similarity measures are calculated, then a term to term Machine Translation System (MTS) from Arabic WN to English WN. But if there are matching concepts in AWN, a translation from AWN to WN is performed and the mapping is done by searching the corresponding concepts in the WN. The closest concept is based on the semantic similarity scale. After that, these concepts will be translated into Arabic using MTS and the text document is represented as a vector of concepts.

Alian et al., 2016 presented a new approach to demystifying Arabic words by using

Wikipedia as a lexical source for demystification. The closest context to the fuzzy word is determined using two techniques, the vector space model and cosine similarity between the word context and senses retrieved from Wikipedia. Three experiments were performed to evaluate the suggested approach, in which the two experiments used the first sentence retrieved for each meaning from Wikipedia, but they used a different vector model for space while the third experiment uses the first paragraph of meaning was retrieved from Wikipedia. The results of the experiments showed that using the retrieved first clause is better than using the retrieved first sentence and that using the Tf-Idf VSM is better than using the initial frequency VSM.

Because of the (BoW) limitations (Zhao & Mao, 2017) in modern applications, (Makris et al., 2020) presented a new supervised knowledge-based approach that uses community discovery algorithms for textual annotation with Wikipedia entities, establishing an unmatched concept of community cohesion as a measure of local contextual cohesion fit. This methodology was an empirical evaluation that revealed deeper inference on the connectedness and cohesion of the local in the Wikipedia graph generally holds significant improvements by focusing on improving the accuracy of less common annotations. The suggested approach is convenient for vast adoption, achieving strong demystification performance.

### 3.6 Word Sense Disambiguation

The word sense disambiguation task aims to differentiate words' meanings (senses) from the context of the word. This process is similar to named entity disambiguation but differs in the fact from the disambiguation process with the latter is using knowledge graphs. In this domain, we explore the most recent papers in this sense and what approaches are followed by the authors to achieve this uneasy task. This section includes the most recent literature related to word sense disambiguation for the Arabic

languages and based on using the transformers-based approaches.

**ArabGlossBERT** (Al-Hajj & Jarrar, 2021) treated the WSD task as sentence pair binary classification problem. The dataset consists of  $\sim 167k$  pairs, i.g., context-gloss positive and negative pairs which were extracted from the Arabic Ontology (Jarrar, 2011, 2021) and the Lexicographic database at Birzeit University (Jarrar, 2020; Jarrar & Amayreh, 2019a). The dataset was split into training and testing. The test set was selected such that, every pair in the test set should not appear in the training set. Three versions of the Arabic pre-trained BERT model were fine-tuned to accomplish this task. The authors achieved promising results with an accuracy of 84%.

Similar approach is used in (El-Razzaz et al., 2021). The authors fine-tuned two Arabic BERT models on a relatively small dataset of pair-gloss. The dataset consists of 5k lemmas, with balanced labels (15k True, 15k False). Their approach achieved 89% on f1 score. However, as mentioned in (Al-Hajj & Jarrar, 2021), the accuracy of this research is not reliable. The authors Al-Hajj and Jarrar have repeated the experiments done in (El-Razzaz et al., 2021) to interpret the f1 result, but they found that the majority of the sentences that were used in the testing phase, were also used in the training phase of the experimentation.

### 3.7 Highlight the Research Gap

In this section, we highlight the research gap from the reviewed papers. Moreover, challenges and open issues in the software-specific named entity disambiguation in the Arabic language are reported for future insights and directions.

- Systems that are built using a rule-based approach often rely on domain experts that manually handcrafted rules and dictionaries. They are domain-specific and hard to apply to other domains. Besides, they are slightly costly and not portable. Furthermore, they need human expertise along with programming skills

and knowledge in the language (Sarawagi, 2008). Because of that, rule-based approaches are outdated nowadays. Researchers shift to machine and deep learning approaches. In the time period of this study, only two studies (Bouziane et al., 2021; Sakor et al., 2020) adapted this technique.

- Most of the Arabic NED literature targets knowledge bases derived from Wikipedia, such as DBPedia, and Yago. In these databases, information is gathered automatically by harvesting information from the infoboxes and categories located on Wikipedia pages, which are not editable. Wikidata, which is a Multilingual data graph that is editable, has recently obtained a reputation as a target database for entity linking (Delpeuch, 2020). For the Arabic language, only some studies have taken into consideration linking entities to Wikidata.
- NED is a challenging process as there are many entities that can have more than one type depending on its context and the morphological richness of the language. For example, the word Jaguar can be referred to as an animal, Car brand, or a movie name (Al-Qawasmeh et al., 2016). This process gets harder when it comes to the Arabic language due to many reasons, some of which is related to the nature of the Arabic language itself, as the Arabic language is a complex rich morphological language, "there is no capitalization, agglutination, optional short vowels, free word order, lack of uniformity in writing style" (Al-Smadi et al., 2019). The second major reason that makes named entity recognition harder is the lack of resources for the Arabic language, in addition to the lack of benchmark datasets for evaluation purposes (Darwish et al., 2021).
- When it comes to software-specific named entity linking, there are no studies that propose nor introduce a mechanism or a technique to link named entities in this to any knowledge graph, although some studies (C. Zhou et al., 2018) have mentioned using KGs as a way of entity disambiguation in this field. (Li et al., 2019; Sorokin & Gurevych, 2018b; Tan et al., 2014; Ye et al., 2016; C. Zhou et al., 2018) have introduced NER systems for software-specific named entities.

- There is a lack of annotated datasets for the Arabic language. This is due to the lack of research on this field in the Arabic language especially Wikidata. Only one study (Bouziane et al., 2020) targeted the NED on DBpedia and Wikidata. Moreover, as mentioned before, all research done on Arabic NED targeted culture heritage with the main focus on three entity types (Person, Location, Organizations). This adds an extra challenge towards creating domain-specific data to include new entity types that serve our purpose to disambiguate entities related to the software-specific entities, such as programming languages, versions, etc.
- Although most of the studies that proposed solutions to entity linking for the English language have clearly adopted deep learning algorithms such as BERT, studies that proposed solutions for the entity linking in the Arabic language are way behind as they still adopt traditional artificial intelligence techniques and rule-based approaches. Transformers-based models such as BERT have revolutionized the NLP research since BERT can understand the semantics of words. This, in turn, encouraged us to adopt applying the transformers-based approaches (mainly BERT) to address the research problem.

### 3.8 Summary

In this chapter, we have introduced a literature review of named entity linking and concept matching landscape. In section 3.2 we illustrated the methodology of conducting this literature review in detail. This study follows the (Kitchenham & Charters, 2007) guidance. The search strategy, search strings, online data sources were discussed and documented carefully. While in section 3.2, the inclusion and exclusion criteria related to this study were made clear. We followed our literature review by discussing the state-of-the-art methodologies applied in entity linking divided by the approach followed. While, the research gap, challenges, and limitations were discussed thoroughly in section 3.7. Table 3.3 summarizes the selected related work, while Tables 3.5 give a

quick comparison of the work done by each group of papers that belongs to the same group.



## Chapter 4

# Research Methodology

The literature review in the previous chapter highlighted the gap in named entity disambiguation and linking for the Arabic language. Recall that we highlighted the research gap in chapter 3, section 3.7. The studies that proposed solutions for named entity disambiguation for Arabic language used classical machine learning techniques and rule-based approaches. However, rule-based approaches depend on domain experts; Hence, they are very costly. Moreover, most of the studies linked extracted named entities to KGs that are derived from Wikipedia, such as DBpedia and Yago. The major concern with these KGs is that they are not editable. In this research, we aim to bridge this gap and build on previous efforts by introducing two Arabic corpora. In this chapter, our research methodology is presented. Section 4.1 illustrates the entity disambiguation and linking process. Section 4.2 justifies the need for annotated corpora. Sections 4.3 and 4.4 present our two annotated corpora, Wojood-NED and WikiGlossContext, respectively.

### 4.1 The Entity Disambiguation and Linking Process

In this section, we overview the general design of the named-entity disambiguation and linking components: (1) the NEL tagging component, (2) the entity lookup component, and (3) the named entity disambiguation component. In what follows, we give a quick overview of each component; the full details of the NEL and the NED are presented in

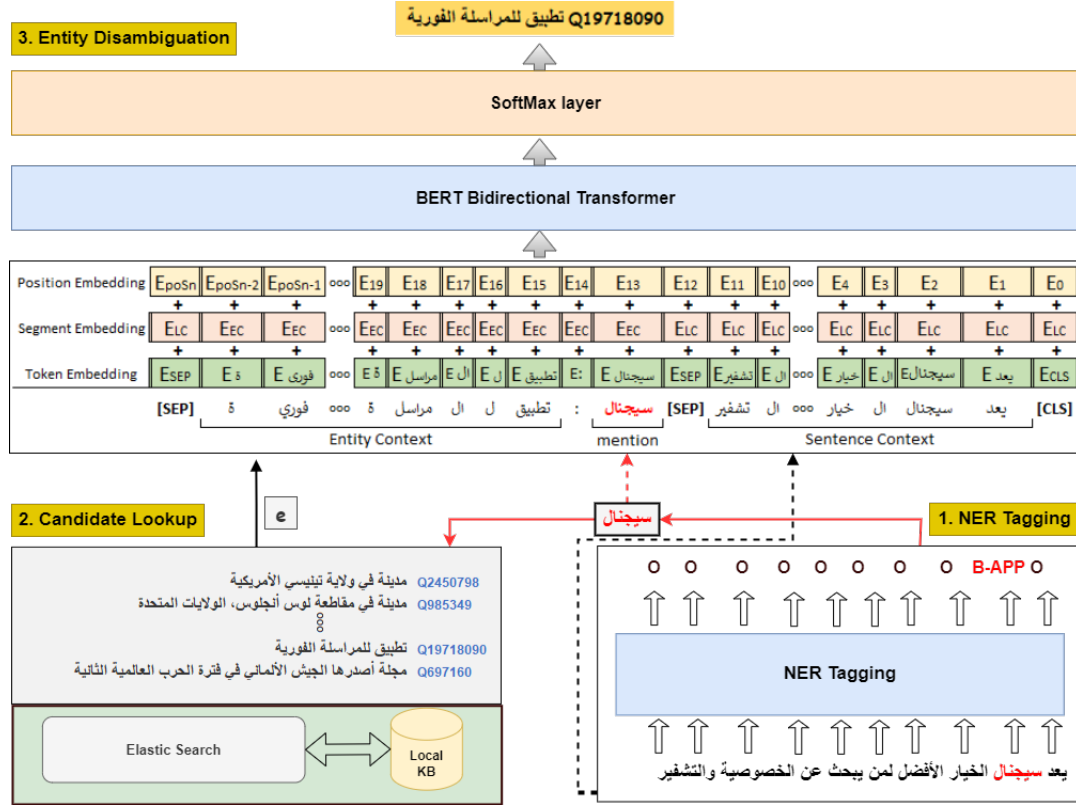


FIGURE 4.1: The presented architecture has three main components : i) NER Tagging that extract named entities from a free-text ii) Candidate Lookup returns all possible candidates for a given ambiguous named entity iii) Entity Disambiguation: BERT-based transformer model to choose the suitable named entity from a set of candidates.

separate sections 4.3 and 4.4, respectively. The candidate lookup component will be presented in chapter 5.

- 1. NEL tagging:** The main functionality of this component is to extract named entities from a free text. The acronym NEL will be used in the thesis to denote our named entity lookup component. Building the NER tagging model is beyond this thesis's objectives. We did not fine-tune the BERT model for the NER task. But to achieve our goal, we used Wojood, a NER API developed at Birzeit University (Jarrar et al., 2022)<sup>1</sup>. Nevertheless, as Wojood does not support

<sup>1</sup>The model is deployed at the following URL as a web service <https://ontology.birzeit.edu/Wojood/>

software-specific entities, we extended the Wojood corpus by annotating part of it (Quora module) with six additional software-specific tags that are commonly used in the software-related text, such as bugs and testing phase (C. Zhou et al., 2020; C. Zhou et al., 2018), development phase (Li et al., 2019), security (Gasmi et al., 2018) The new tags are used for evaluation purposes only in this thesis. Then, we will train a new NER BERT model on the new corpus as future work. Generally, the NER model takes a sentence as input, and for each word in the sentence, the model recognizes a tag to which the word belongs. As shown in Figure 1.1, (سيجنال /Signal) was recognized as an application (B-App).

2. **Candidate lookup:** Candidate lookup is also known as "candidate generation" or "mention detection". The goal of this component is that, given a search query, the component returns all possible candidate nodes from Wikidata. We need this lookup component because a named entity (e.g سيجنال /Signal) could be also written in different forms (سيجنل /Signal, سيغنل /Signal, سيغنال /Signal). The goal is to find all possible nodes in Wikidata that have similar forms - although they may have different meanings (see Figure 1.1). The input to this component is a named entity (i.e., search query), and the output is a list of all candidate Wikidata nodes.

To implement the candidate lookup component we downloaded the Wikidata dump and for each Arabic Wikidata node, we extracted the first sentence from Wikipedia (as both are connected to using the site-links) - similar to the methodology followed by (Sakor et al., 2020). Then we install the Elasticsearch framework (Kuc & Rogozinski, 2013) and fine-tuned the query to enable proper and effective lookup (the details of the candidate lookup component are presented in chapter 5, sections (5.3, 5.4, and 5.5)).

3. **Entity disambiguation and linking:** The goal of this component is to select the correct Wikidata node from a set of candidate nodes. That is, given a named entity in a sentence, our goal is to select the corresponding entity (i.e., node)

from the Wikidata knowledge graph - from a list of candidate entities retrieved by the previous candidate look-up component. As illustrated in Figure 4.1, the first component recognizes the named entity (سيجنال /Signal) in a sentence, then the second lookup component returns a set of candidate nodes from Wikidata. This third component decides (i.e., disambiguates) which of the candidates is the correct Wikidata nodes. It returns the node Q\_identifier (Q19718090). Although entity disambiguation and entity linking might be addressed as different problems, we consider them as one problem in this thesis. This is because the disambiguation of a named entity is to find the corresponding Wikidata node, and as we know the node URL (i.e., the Q\_identifier) then disambiguation becomes the same as linking.

As mentioned earlier in the section (3.4), researchers followed different approaches to tackle entity disambiguation. Some used rule-based approaches (Bringmann et al., 2011; Fürnkranz, 2013), others used word embeddings (Banerjee et al., 2020; Esmeir, 2021), statistical approaches (Delpeuch, 2020), transformers-based approaches (Cetoli et al., 2018; Perkins, 2020; Raiman & Raiman, 2018), and hybrid approaches (Al-Qawasmeh et al., 2016; Bouziane et al., 2020; Gad-Elrab et al., 2015). The most recent research has used transformers-based approaches such as BERT, Glove, Elmo, and Bi-LSTM. In this thesis, we will experiment with different Arabic BERT models (science BERT (Devlin et al., 2018) is state-of-the-art) in order to evaluate the best-performing model for the sentence-pair classification task. This is similar to (Kannan Ravi et al., 2021), which used an architecture based on BERT for the NED task to link entities in the English language with Wikidata and Wikipedia, and treated the NED problem as sentence-pair classification problem. That is, our entity disambiguation problem is transferred to a sentence-pair binary classification problem, and hence, the suitable BERT architecture will be used (as illustrated in Figure 4.1). The special [CLS] token at the final layer which produces raw predictions will be fed into a

ranking system using a soft-max layer to convert raw predictions into probabilities for each of the two classes (*True*, *False*) and a ranking system will determine which of the fed pairs have the maximum probability of the class *True*.

## 4.2 The Need For Annotated Corpora

Arabic is considered a poor-resource language (Haff et al., 2022; Jarrar et al., 2017) and lacks annotated corpora suitable to train the models needed to accomplish the NER and NED components. Based on the literature review in chapter 3, we were unable to find corpora annotated with Arabic software-specific tags. Besides, there are a few available benchmark datasets for the sentence-pair binary classification tasks. Only a small dataset of gloss-context pairs was used in (El-Razzaz et al., 2021), which includes about 5k lemmas (15k positive, and 15k negative) pairs. A larger dataset described in (Al-Hajj & Jarrar, 2021) contains 167k pairs of gloss-context labeled pairs (26k lemmas with 33k glosses), extracted from 150 lexicons at Birzeit University (Jarrar & Amayreh, 2019b). However, the glosses in this dataset define linguistic concepts rather than named entities. To overcome the aforementioned limitation, two corpora (Wojood-NED, and WikiGlossContext pairs) were built. The details for building and annotating each of the corpora are presented in the following sections.

## 4.3 The Wojood-NED Corpus

This section presents the Wojood-NED corpus, which is part of the previously annotated Wojood corpus (Jarrar et al., 2022), then enriched with additional types of named entities. We took about ~50k tokens from Wojood, those that were originally extracted from Quora (a discussion form about software-related issues), and annotated using 21 classes of entities. We used this part of Wojood (we call it Wojood-NED) to

enrich with more types of entities and link with wikidata<sup>2</sup>. In addition to the 21 classes of entities, we introduced six software-specific classes (listed in Table 4.1). Moreover, named entities in this corpus were manually linked with their corresponding Wikidata Q-identifier. In the following sub-sections, 4.3.1, and 4.3.2, we present the details of Wojood-NED tagging and linking with Wikidata. These details contribute to answering the research question  $RQ_{1b}$  that stated: **( Given the lack of annotated Arabic corpora (i.e. datasets labeled with software-specific named entity classes) in the software-related aspects, what software-specific named entity classes are required to enable understanding the semantics of these named entities?)** by illustrating the process of Wojood-NED corpus tagging and linking.

### 4.3.1 Software-specific NEL Tagging

The named entity lookup (NEL) tagging process aims to manually label the named entities mentioned in the text. The Wojood-NED corpus consists of **2,650** sentences, with **50,449** tokens that are already labeled with 21 nested tags. However, for the sake of our research, we introduced new six software-specific tags (listed in Table 4.1), which are not supported by Wojood. We used these tags to manually annotate the corpus for evaluation purposes. As we did not fine-tune a new BERT model that is able to recognize software-specific tags. These new six tags are inspired by previous research, especially (Li et al., 2019; Ye et al., 2016; C. Zhou et al., 2020; C. Zhou et al., 2018).

---

<sup>2</sup>The Wojood-NED corpus, which is part of Wojood, was crawled from Quora by the author of this thesis and then given to the Wojood authors to annotate manually with 21 tags. Before crawling Quora we referred to the robots.txt file (<https://www.quora.com/robots.txt>). We also contacted the Quora team for permission via (<https://help.quora.com/hc/en-us/requests/new>) as robots.txt file noted (“If you operate a search engine and would like to crawl Quora, please please visit our contact page <<https://help.quora.com/hc/en-us/requests/new>>. Thanks.”) Please note that only public questions and answers are crawled, we didn’t gather sensitive information about users who posted the questions such as usernames. In addition, many studies use the same technique and crawl public data from websites such as news, and QA websites.

We limited the choice of the software-specific named entities to the six mentioned types due to three reasons. Firstly, the Arabic language is not used at the level of software development programming languages. Hence, it is not suitable for fine-grained named entities, such as functions, libraries, data types, etc. Secondly, the Wojood-NED corpus as mentioned earlier is taken from a software-related discussions on Quora. The data nature directed the selection of the tags since the corpus is rich in the selected six tags mentioned in Table 4.1. However, new tags can be introduced if necessary, according to the use case available at hand, and if there is a new source of data that is rich with the newly introduced named entities. Finally, to our knowledge, this is the first study that extracts Arabic software-specific named entities, with an aim to understand technical software-related documents such as SRS, app reviews, user stories, etc.

TABLE 4.1: Software-specific named entities description

Named Entity	Start Tag	Inside Tag	Description
Programming Language	B-PL	I-PL	Programming Languages like java, c#,c++
Platform	B-PLATFORM	I-PLATFORM	Platforms and OSs like windows, Linux, Android, and x86.
Application	B-APPLICATION	I-APPLICATION	software applications like Instagram, Facebook, and Whatsapp.
Framework	B-FRAMEWORK	I-FRAMEWORK	Software tools, programming libraries, like MS Word, flask, Bootstrap, and software servers.
Programming Paradigm	B-PP	I-PP	The method used to group programming languages based on the programming style. For example OOP, Functional, etc.
Software Standard	B-SD	I-SD	like data format, JSON, HTML, and CSV.

Each token in the corpus was manually tagged using the IBO2 format (L. A. Ramshaw & Marcus, 1999) for nested tags, viz. an entity begins with B (Beginning) tag. If the entity has more than one token then the second and the following words are tagged with an I (Inside) tag. The *O* (Outside) tag is used for each word that does not belong to any of the predefined classes of entities. In the example shown in Table 4.2), لغة جافا/Java language is tagged with programming language (PL). The first word is marked as B-PL and the subsequent word جافا /Java is marked as I-PL. Words like تعلم/learn, مع /with,

and بيئة/environment are marked with *O* since they are not named entities. The second sentence in the Table illustrates nested named entities which are defined as entities that are embedded inside another entity. In this case, both entities are annotated with the proper tag sorted from outside to inside. For example, خان أكاديمي /Khan Academy is first tagged with B-WEBSITE then B-PERS because the B-WEBSITE is the outside entity (top-most entity).

TABLE 4.2: Words with tags example

Word	Tag
تعلم /learn	O
لغة /programming	B-PL
جافا /Java	I-PL
Java	B-PL
مع /with	O
بيئة /environment	O
التطوير /development	O
أندرويد/Android	B-FRAMEWORK
ستوديو/Studio	I-FRAMEWORK
android	B-FRAMEWORK
studio	I-FRAMEWORK
مثل /like	O
خان/Khan	B-WEBSITE B-PERS
أكاديمي/Academy	I-WEBSITE
أو/or	O
أكاديمية /Academy	B-WEBSITE
حسوب /Hasoub	I-WEBSITE

As a result, there are 27 classes of entities in the Wojood-NED corpus, which are the 21 classes in the original Wojood, and the six tags we introduced in this research. Figure 4.2 illustrates the sentences distribution lengths parentage. The median length of the sentences is 14 tokens, while the mean percentage of the lengths of the sentences is 19 tokens per sentence.

Table 4.3 lists each of the seven classes of software-specific named entities with the number of occurrences in each class. The **WEBSITE** class is the most frequent in the corpus with 562 entities, followed by the programming language (**PL**) class with



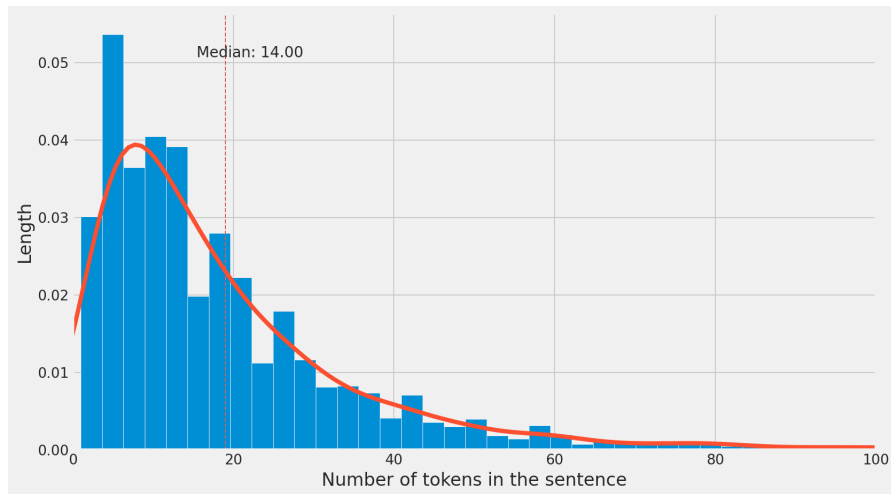


FIGURE 4.2: Sentences distribution

TABLE 4.3: Counts of the flat, nested, and total of software-specific entities in the Wojood-NED corpus.

Tag	Count Flat	Count Nested	Total	Wikidata links
PP	28	0	28	28
PLATFORM	41	1	42	42
APPLICATION	50	0	50	34
SD	92	0	92	89
FRAMEWORK	289	4	293	265
PL	550	3	553	480
WEBSITE	562	1	563	228
Total	1,612	9	1,621	1,166

TABLE 4.4: Statistics about the Wojood-NED software-specific linked entities

	Count
Count of all entity mentions	1,166
Count of unique entity mentions (aliases)	281
Count of unique node IDs (for entity mentions)	152

550 entities. In general, the number of nested entities is small (9) compared to the flat ones (1,612). Moreover, Table 4.4 illustrates that from 1,361 linked named entity, there are 276 unique links (in terms of Wikidata Q-identifier), and 598 unique named entity label.

تعليم لغة جافا	Java	مع بيئة التطوير أندرويد ستوديو
PL	PL	FRAMEWORK
Q251	Q251	Q13233410

FIGURE 4.3: Examples of flat entity of mentions of different types.

### 4.3.2 Linking Wojoood-NED to Wikidata

Given the Wojoood-NED corpus (described in the previous sub-section), our goal in this section is to link each annotated entity with its corresponding Wikidata node, if exists. That is, given the named entities in the corpus, including the six software-specific classes, we aim to link each named entity with the Wikidata knowledge graph.

There are 10 classes of entities that we did not link with Wikidata because they are irrelevant for our research purposes, which are: EVENT, DATE, TIME, CARDINAL, ORDINAL, PERCENT, QUANTITY, UNIT, MONEY, and CURR. As a result, the classes of entities we linked in the corpus are: ORG, OCC, PLATFORM, FRAMEWORK, WEBSITE, NORP, PERS, PL, APPLICATION, SD, LANGUAGE, GPE, PP, and LOC. Besides, named entities that do not have a corresponding matching node on Wikidata are not linked. For example, the frameworks "Mongo express" and "MERN"<sup>3</sup> are not found on Wikidata. Person names for non-popular people usually do not have Wikidata nodes and Websites such as *مدونة بلوهوست* /Bluehost blog does not have corresponding nodes. In this thesis, although Wojoood-NED is annotated with 21 entity classes. Only software-specific named entities will be considered in the evaluation of the NED model. Because we are focusing on linking named entities in software-related texts.

This linking process was performed manually. For each named entity in the corpus, we query the Wikidata knowledge graph using our lookup module, then the corresponding node is selected. Figure 4.3 shows how named entities have been linked to Wikidata

<sup>3</sup><https://www.mongodb.com/mern-stack>

using the Wikidata Q-identifier number. The entity *لغة جافا/Java* is linked to its corresponding node in Wikidata (Q251). The named entity *أندرويد ستديو/Android Studio* is also linked to the Wikidata node (Q13233410).

## 4.4 WikiGlossContext Pairs Corpus

As discussed earlier, our methodology to disambiguate and link entities automatically follow the same methodology used for word-sense disambiguation described in (Al-Hajj & Jarrar, 2021). The idea is to fine-tune a BERT model on the so-called *context-gloss pair binary classification* task (Kannan Ravi et al., 2021). In other words, given a gloss (i.e., meaning definition) for a target word, and given a context (i.e., a sentence in which the target word appears), BERT is trained to judge whether it is *True* (or *False*) that the given gloss represents the meaning of the target word in the given context. The dataset used to train BERT for this task consists of about 167K context-gloss pairs (Al-Hajj & Jarrar, 2021). Each pair was labeled with *True* or *False*

To extend the ArabGlossBERT dataset with more labeled context-gloss pairs (for the purpose of linking entities with Wikidata), this section describes the process of building and labeling a corpus of sentence pairs (which we call WikiGlossContext). The sentence pairs corpus was extracted from Wikidata and Wikipedia. The extracted pairs were labeled with *True*, while the *False* pairs were generated based on these *True* pairs.

### 4.4.1 Gloss-Context Pairs Extraction

The following steps, illustrated in Figure 4.4, are performed for gloss-context pairs extraction:

**First - Extract raw gloss-context pairs:** The first step in building the corpus was to download the latest version of the Wikidata dump<sup>4</sup>, then the needed fields were

<sup>4</sup> Wikidata dump (16-Dec-2021 01:03) <https://dumps.wikimedia.org/wikidatawiki/entities/>

extracted (see fields 1-13 in Table 4.7) and saved in CSV files<sup>5</sup>. Our goal is to extract a gloss and a context for each node in Wikidata. For each node in the Wikidata graph, we extracted the gloss (from the description field in Wikidata) and a context (the first sentence from the node’s Wikipedia page). In other words, the Wikidata Arabic description field is considered a gloss, and the Wikipedia’s summary (the first sentence in a Wikipedia article) is considered a context. To get a context from Wikipedia for a Wikidata node, we used the Wikipedia site-link that is found in the item information on the Wikidata node<sup>6</sup>.

Wikidata nodes that do not have Arabic description were excluded. We also excluded the nodes that do not have a corresponding site-link to Wikipedia. As a result, nodes with Wikidata descriptions and Wikipedia descriptions are considered a gloss-context pair and given *True* label (see fields 14-15 in Table 4.7). Figure 4.5 shows the distribution of words frequencies for the extracted Wikidata glosses and Wikipedia contexts.

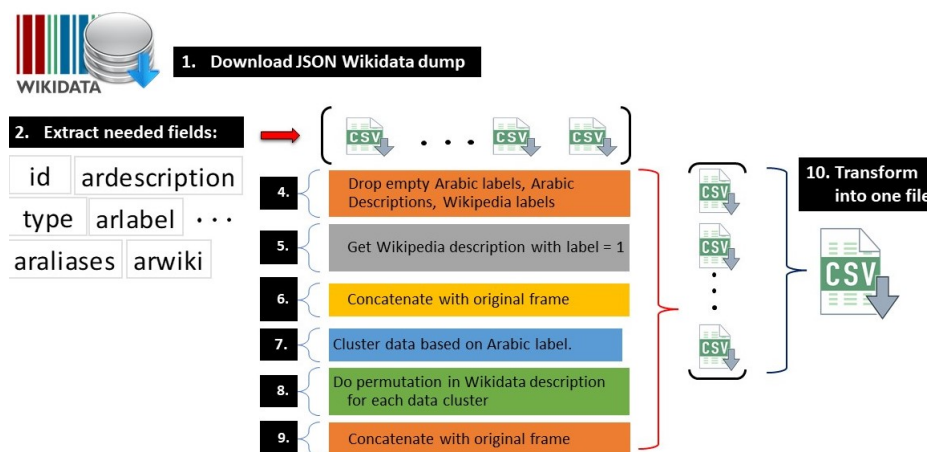


FIGURE 4.4: NED corpus building and annotating steps

<sup>5</sup>9,532 file, each file contains 10,000 item, number of items 95,318,999

<sup>6</sup>The `Wikipedia` Python package is used to retrieve one sentence length summary for the article. To ensure that the correct match of the label is returned, the `summary` method parameter `auto_suggest` is set to `False`, and the exact Wikipedia `label` of the named entity which is extracted from Wikipedia sit-link is passed to the `summary` method

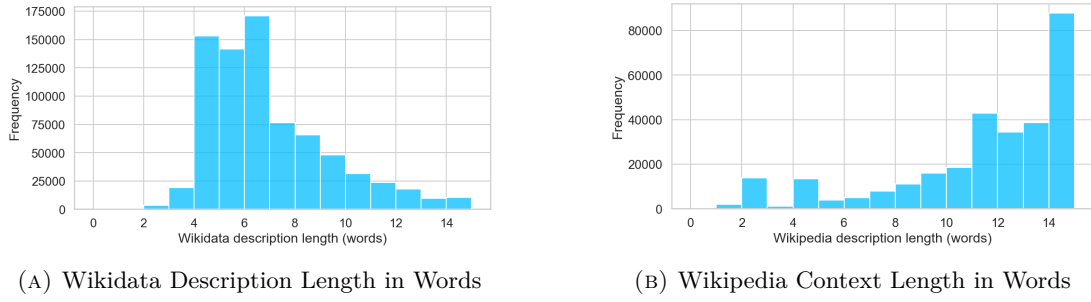


FIGURE 4.5: Glosses and contexts length in words

**Second - Clean and normalize nodes' labels:** Given that Wikidata nodes have Arabic aliases (i.e., alternative labels) in addition to the default label filed, these aliases are stored in a comma-separated string format in the *"also known as"* filed in Wikidata (see Figure 4.6 and Figure 2.3). In order to treat such aliases as synonyms of node labels, some pre-processing and normalization are required. For example the item (برمجة كائنية التوجه/Object-oriented programming) that has the Wikidata Q-identifier (Q79872) has thirteen aliases (see Figure 4.6), these aliases are normalized and considered synonyms, i.e., we assign them the same context-gloss pairs as shown in Figure 4.7.

Wikidata_IQ	WikidatArabicLabel	WikidataAliases
Q79872	برمجة كائنية التوجه	برمجة شيئية, برمجة كائنية المنحى, برمجه كائنيه التوجه, OOP, برمجة كائنات موجهة, برمجة كائنية المنحى, برمجة كائنية, برمجة كائنية التوجيه, برمجة موجهة, برمجة غرضية التوجه, البرمجة كائنية التوجه, البرمجة الشيئية, شيئية المنحى

FIGURE 4.6: Aliases as found in Wikidata for the item Q79872

- 1. Normalization of node labels:** Some Arabic node labels and their aliases are the same. For example, the node label احمد/Ahmad has aliases such as أحمد/Ahmad with hamza. The normalization step (for smart matching of Arabic words, See (Jarrar et al., 2018)) is necessary to unify these cases. The Arabic label of each node needs to be normalized in order to enhance their consistency and to be used by the lookup model. The following normalization steps are performed:

Wikidata_ID	WikidataArabicLabel	Context	Gloss	Label
Q79872	برمجة كائنية	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	برمجة كائنية: نوع من أنماط البرمجة	1
Q79872	برمجة موجهة	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة شيئية	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	البرمجة كائنية التوجه	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة كائنات موجهة	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة غرضية التوجه	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة كائنية المنحني	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872		OOP	OOP: نوع من أنماط البرمجة	1
Q79872	برمجة كائنية التوجه	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة كائنية المنعَى	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	شيئية المنعَى	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	البرمجة الشيئية	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة كائنية التوجيه	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1
Q79872	برمجة كائنية التوجه	برمجة ذات نمط كائنية التوجُّه أو شيئية المنعَى (بالإنجليزية: OOP - object-oriented programming)	نوع من أنماط البرمجة	1

FIGURE 4.7: Item Q79872 aliases expansion

- Removing sub-strings inside parentheses. Such sub-strings in Wikidata are typically used for disambiguation, not part of the Arabic label, for example, the item ('Q99840504') has the label *ءالبصيرة (الأحساء)* /Baseera (Al-Ahsa). This becomes *البصيرة* /Baseera. 818,261 node labels were affected by this normalization step.
- The following special characters were removed from node labels: `.[%$#@! ?]`
- All Arabic diacritics including Shaddah, as well as small Quranic annotation signs and tatwelah were removed.
- Underscores were replaced with white space.
- All extra spaces were removed.
- All forms of alif (ا، آ، إ) are unified into (ا).

2. **Data Elimination:** The data elimination process is needed to remove irrelevant, poor, or duplicate node labels. For example, after normalizing the node label *أحمد* /Ahmad to *احمد* /Ahmad without hamza, two node labels with the same surface form are found, one of them should be deleted. In addition, given that some Wikidata nodes are irrelevant, especially those that are not named entities (listed in Table 4.5) should be excluded. The following data elimination steps are performed semi-automatically (using scripts):

- Short Wikidata and Wikipedia descriptions (i.e. less than four words long) were excluded as they do not contribute usefully to the BERT fine-tuning

process. The length was learned while auditing and reviewing the corpus manually, as sentences less than four words usually do not have meaningful descriptions, and in most cases are the same as the node label.

- Labels written in languages other than Arabic (e.g., English and Chinese) are excluded.
- Labels that are a symbol or a single character in any language are excluded.
- Incomplete descriptions are excluded. Indeed, some of the descriptions were not complete. This is due to a limitation of the Python package used to retrieve a one-sentence length description from Wikipedia based on period. Some abbreviations that contain a period (.ﺩ/Dr.) came in the sentence and are thus considered the end of the sentence.
- Duplicate labels (as a result of the previous normalization phase) were removed.
- Any node that is an *instance-of* one of the nodes listed in Table 4.5 is considered irrelevant and thus excluded. This resulted in excluding 249,425 nodes, which are either not named entities or irrelevant for the purposes of this research.

**Third - Enrich glosses:** This step aims to enrich Wikidata description in order to be used as glosses. Some Wikidata descriptions are short and do not contain the node label(s). Such poor descriptions may affect the fine-tuning performance of BERT. To overcome this issue, we follow the same methodology in (Al-Hajj and Jarrar, 2021), which suggested adding the lemma into the gloss; such that, the gloss becomes a concatenation of the node label, node type(s), and the description fields. As such, the gloss is generated accordingly to the following pattern: let the Wikidata Arabic node label (*label*), the Wikidata types (*type<sub>1</sub>, type<sub>2</sub>, ..., type<sub>n</sub>*), and the Wikidata description is denoted by *description*, then the gloss will be in the following format: *label : type<sub>1</sub>, type<sub>2</sub>, ..., type<sub>n</sub> , description*. For example, the node (فيسبوك/Facebook) has the following Arabic description (خدمة تواصل)

TABLE 4.5: List of excluded Wikidata types from the WikiGlossContext

Wikidata Type	Description
Q577	Year
Q4167836	Wikimedia category
Q11266439	Wikimedia template
Q244751	583 BC
Q3186692	calendar year
Q235680	common year starting and ending on Friday
Q36330215	Wikimedia location map template
Q15647814	Wikimedia administration category
Q4167410	Wikimedia disambiguation page
Q235680	common year starting and ending on Friday
Q36330215	Wikimedia location map template
Q15647814	Wikimedia administration category
Q19828	leap year.
Q14204246	Wikimedia project page.
Q42032	country code top-level domain.
Q41713761	Arabic letter
Q3863	asteroid
Q645924	classical Kuiper belt object
Q13406463	Wikimedia list article
Q21199	natural number
Q13366129	odd number
Q49008	prime number

(اجتماعي/social networking service), and has two types:(منظمة، موقع ويب). Hence, its gloss becomes (فيسبوك/Facebook: منظمة، موقع ويب، خدمة تواصل اجتماعي) (an organization, a website, a social networking service). This enrichment makes glosses more meaningful and thus helps BERT to form clearer embeddings between related words.

**Fourth - Generate *False* context-gloss pairs:** In the previous steps, we extracted 730K context-gloss pairs for Wikidata nodes. That is, given a node, we were able to extract a gloss (from Wikidata) and a context (from Wikipedia). All extracted pairs in this way are considered *True*. In this step, we aim to generate *False* pairs, based on the *True* pairs. Both *True* and *False* pairs will be used for fine-tuning BERT models. Recall that a *False* context-gloss pair means that this



TABLE 4.6: Statistics about the WikiGlossContext corpus

	Count
<i>True</i> pairs (extracted)	731,509
<i>False</i> pairs (generated)	374,899
Total	1,106,408

gloss does not represent the meaning of the node mentioned in this context. To generate *False* pairs automatically, we used two alternative methods:

- (a) **Cross-Relating:** Based on the 730K *True* pairs extracted in the previous step, the *False* pairs were then generated. This is done as follows: *True* gloss-context pairs for a given node label are cross-related. For example, let  $(context1-gloss1)$  and  $(context2-gloss2)$  be the two *True* pairs of the same node label, then  $(context2 - gloss1)$  and  $(context1 - gloss2)$  are generated and labeled with *False*. This done only node labels that belong to different nodes. For example, as shown in Figure 4.8, بايثون/Python appears with in two *True* context-gloss pairs (i.e., for two different Wikidata nodes), these we cross-related the context and glosses to generate *False* pairs. As a result of that, about 374,899 *False* pairs were generated. Table 4.6 provides some statistics about the corpus.

True pairs			False pairs		
Context 1	Gloss 1	Label	Context 1	Gloss 2	Label
بايثون هي لغة سهلة التعلم مفتوحة المصدر قابلة للتوسع.	بايثون: لغة برمجة عالية المستوى لغة مفسرة، لغة برمجة، عالية.	True	بايثون هي لغة سهلة التعلم مفتوحة المصدر قابلة للتوسع.	بايثون: أصنوفة، فصيلة من الثعابين الغير سامة.	False
Context 2	Gloss 2	Label	Context 2	Gloss 1	Label
تتواجد أفعى اليايوشون في أفريقيا وآسيا وأستراليا.	بايثون: أصنوفة، فصيلة من الثعابين الغير سامة.	True	تتواجد أفعى اليايوشون في أفريقيا وآسيا وأستراليا.	بايثون: لغة برمجة عالية المستوى لغة مفسرة، لغة برمجة، عالية.	False

FIGURE 4.8: Example of *False* pairs generated using cross-relating.

- (b) **False Local:** Another approach for generating *false* pairs is to choose the *false* gloss that is closest to the *true* one for the named entity. In this method, the closest gloss is determined by measuring the cosine-similarity of the BERT representations of the glosses. Then the gloss with the highest similarity is determined as the *false* pair. This method aims to generate

a robust model by fine-tuning it using difficult samples (i.e. the *true* and the *false* glosses for the named entity are close to each other as they have the highest similarity) and hence the model is trained to distinguish *true* pairs from *false* pairs to a given named entity which are similar or with subtle changes. For example, Figure 4.9 shows examples of *false* pairs generated based on the False-local method. For example, the *false* gloss of the named entity (نوكيا لوميا 630/Nokia Lumia 630) is selected based on the cosine similarity of the BERT embedding of the *true* gloss (نوكيا لوميا 630: هاتف) (نوكيا لوميا 630: A Nokia Smartphone With Windows Phone) with other glosses. Therefore, (نوكيا لوميا 720/Nokia Lumia 720) is selected as a negative gloss. Note that the gloss in this case is similar to the *true* one (they have the same definition with only different version numbers). In this approach, the ratio values were used (1:1). That is, for each *True* context-gloss pair, only one *False* pair was generated.

Context	Gloss	Label
كروميوم (بالإنجليزية: Chromium) هو متصفح ويب مفتوح المصدر، يأخذ متصفح جوجو	كروم: متصفح ويب، Google متصفح ويب من تطوير جوجل	0
كروميوم: متصفح ويب، منصة مفتوحة المصدر تعمل عبر متصفح الانترنت (بالإنجليزية: Chromium) هو متصفح ويب مفتوح المصدر، يأخذ متصفح جوجو	كروميوم: متصفح ويب، منصة مفتوحة المصدر تعمل عبر متصفح الانترنت	1
أخبار جوجول (بالإنجليزية: Google News) هو برنامج مبي على الويب، تقدمه شركة جوجو	أرشيف أخبار جوجول: موقع ويب، أرشيف	0
أخبار جوجول (بالإنجليزية: Google News) هو برنامج مبي على الويب، تقدمه شركة جوجو	أخبار جوجول: موقع وتطبيق يقوم بتجميع الأخبار	1
نوكيا لوميا 630 (بالإنجليزية: Nokia Lumia 630) هو هاتف ذكي من نوكيا يعمل بنظام ويندوز فون	نوكيا لوميا 630: هاتف ذكي من نوكيا يعمل بنظام ويندوز فون	1
نوكيا لوميا 630 (بالإنجليزية: Nokia Lumia 630) هو هاتف ذكي من نوكيا يعمل بنظام ويندوز فون	نوكيا لوميا 720: هو هاتف من نوكيا يعمل بنظام ويندوز فون	0
تحالف أمن الحوسبة السحابية "The Cloud Security Alliance" هي منظمة غير ربحية	تحالف أمن الحوسبة السحابية: منظمة غير ربحية، منظمة دولية	1
تحالف أمن الحوسبة السحابية "The Cloud Security Alliance" هي منظمة غير ربحية	أمن الحوسبة السحابية: تكنولوجيا، أمن المعلومات	0
سيليكون جرافيكس (بالإنجليزية: Silicon Graphics, Inc.) هي شركة تصنيع حواسيب ع	سيليكون جرافيكس: عمل تجاري، شركة أمريكية	1
سيليكون جرافيكس (بالإنجليزية: Silicon Graphics, Inc.) هي شركة تصنيع حواسيب ع	سيليكون باور: عمل تجاري، شركة	0

FIGURE 4.9: Example of a negative pairs generation based on the false-local method.

## 4.5 Splitting into Training, Validation and Test Datasets

This section describes how we divided our corpus into training, validation, and testing sets and the criteria used to avoid information leak between the mentioned sets. Recall that the *False* pairs in the WikiGlossContext were generated using two methods (cross-relating and false-local). In the following subsections, we present the data splitting produced for each of the methods.

TABLE 4.7: The Wikidata extracted fields' description

#	Filed	Description
1	<b>id</b>	The Wikidata Unique identifier. (e.g. Q28865 )
2	<b>type</b>	The Wikidata instance of property (P31). Stored in a comma-separated value (e.g. Q899523, Q1268980,...,Q3839507)
3	<b>arlabel</b>	The Arabic label for the entity (e.g. بايثون/Python)
4	<b>enlabel</b>	The English label for the entity (e.g. python)
5	<b>araliases</b>	The also-known as Arabic labels for the entity. Stored in a comma-separated value (e.g. البايثون، ألبيثون، ييثون، بايثون)
6	<b>enaliases</b>	The also-known as English labels for the entity. Stored in a comma-separated value (e.g. py, python 2, python 3)
7	<b>ardescription</b>	The Arabic description filed for the entity (e.g. لغة برمجة، عالية المستوى .مفتوحة المصدر قابلة للتوسيع/ general-purpose programming language )
8	<b>endescription</b>	The English description filed for the entity (e.g. general-purpose programming language)
9	<b>maincategory</b>	The topic's main category property (P910). Stored as comma-separated value (e.g. Q7136128)
10	<b>arwiki</b>	The Arabic Wikipedia label for the entity (e.g. بايثون (لغة برمجة)/Python(Programming Language))
11	<b>enwiki</b>	The English Wikipedia label for the entity (e.g. Python)
12	<b>arwikiquote</b>	The Arabic entity label for the entity on Arabic Wikiquote
13	<b>enwikiquote</b>	The English Wikiquote label for the entity (e.g. Python)
14	<b>wikiDescription</b>	The Wikipedia Description for the entity in the Arabic Language (e.g. بايثون هي لغة برمجة، عالية المستوى سهلة التعلم مفتوحة امصدر قابلة للتوسيع، تعتمد . (أسلوب البرمجة الكائنية )/Python is a high-level, interpreted, general-purpose programming language.
15	<b>Label</b>	<b>True</b> if the Wikidata Arabic description and the Wikipedia Arabic description is for the same entity, otherwise <i>False</i>

#### 4.5.1 Splitting WikiGlossContext Induced by The Cross-related Method

The dataset should be split (into train, validation and test sets) carefully to prevent information leaks. The dataset contains one or more contexts for each Arabic label, thus it cannot be split arbitrarily. Contexts used in the training should not be used in the test dataset. Thus, the test dataset was selected taking into account the following criteria: Every context selected in the test set should not be selected in the training set for the same Arabic label. This is done by firstly, splitting the labels using stratify option into the train, and testing to maintain the same proportions of samples in each of the *True* and *False* classes. Besides, to ensure that pairs that belongs to one label only belongs to one of the datasets (e.g. all pairs for the (جاڤا /Java) label are found in train or test datasets, but not both). Then, the training dataset is split into train and

validate sets with a percentage (90:10), so the whole dataset is split into 80% train, 10% validation, and 10% test. Table 4.8 provides statistics about these sets.

TABLE 4.8: Splitting WikiGlossContext induced by the cross-related method

<b>Dataset</b>	<b>Pairs</b>	<b>Count</b>	<b>Total</b>
Training	<i>True</i>	592904	899,705
	<i>False</i>	306,801	
Validation	<i>True</i>	65,879	99,968
	<i>False</i>	34,089	
Test	<i>True</i>	72,726	106,735
	<i>False</i>	34,009	
		<b>Total</b>	<b>1,106,408</b>

#### 4.5.2 Splitting WikiGlossContext Induced by The False-Local Method

In this method, we split the WikiGlossContext pairs into three balanced sets (train, validate, and test). To avoid information leaks between pairs in the three sets, the *True* pairs were, first, split into train, validation, and test sets with a ratio (70%, 10%, 20%), respectively. Then, for each set, *False* pairs were generated separately by selecting a *False* pair with the highest similarity to the *True* one using the method illustrated in section 4.4.1. Table 4.9 provides statistics about these sets.

TABLE 4.9: Splitting WikiGlossContext induced by the False-Local method

<b>Dataset</b>	<b>Pairs</b>	<b>Count</b>	<b>Total</b>
Training	<i>True</i>	399,797	799,594
	<i>False</i>	399,797	
Validation	<i>True</i>	56,544	113,088
	<i>False</i>	56,544	
Test	<i>True</i>	114,799	229,598
	<i>False</i>	114,799	
		<b>Total</b>	<b>1,142,280</b>

### 4.5.3 Combined Datasets

In this section, we extend the ArabGlossBERT (Al-Hajj & Jarrar, 2021) dataset with additional context-gloss pairs from our WikiGlossContext dataset. More specifically, we propose different dataset combinations (as shown in Table 4.10). The (**D0**) dataset is the original ArabGlossBERT dataset. In **D1**, **D2**, **D3** and **D4**, we added a fixed number of *True* pairs (23k, 40k, 200k, and 400k) to the ArabGlossBERT training set, respectively. We did not add the *False* pairs into these because the original ArabGlossBERT already has a large number of *False* pairs. The test dataset is kept the same ArabGlossBERT test. Keeping the same testing dataset is important to compare whether our enrichment to the original dataset improves the performance. In **D5** (ArabGlossBERT+WikiGlossContext<sub>cross-related</sub>), we added all the context-gloss pairs from WikiGlossContext<sub>cross-related</sub> into ArabGlossBERT training set, and then the dataset is divided into three sets (training, validation and testing) with ratio (80%, 10%, and 10%) respectively. However, in **D6** (ArabGlossBERT+WikiGlossContext<sub>false-local</sub>), we added all the context-gloss pairs from WikiGlossContext<sub>false-local</sub> into ArabGlossBERT training set, and then the dataset is divided into three sets (training, validation and testing) with ratio (80%, 10%, and 10%) respectively. Although, in this combination we produced different test dataset. Finally, D7 and D8 are the WikiGlossContext<sub>cross-related</sub> and WikiGlossContext<sub>false-local</sub> datasets that were described in the tables (4.8, and 4.9) respectively.

TABLE 4.10: Combined datasets' description

	<b>Dataset</b>	<b>Pairs</b>	<b>Count</b>	<b>Total</b>	
<b>(D0).</b> ArabGlossBERT (Al-Hajj & Jarrar, 2021)	Training	<i>True</i>	55,373	151,893	
		<i>False</i>	96,520		
	Testing	<i>True</i>	4,842	15,206	
		<i>False</i>	10,364		
				<b>Total</b>	<b>167,099</b>
	<b>(D1).</b> ArabGloss+Wiki23KTrue	Training	<i>True</i>	70,923	157,791
<i>False</i>			86,868		
Validation		<i>True</i>	7,881	17,533	
		<i>False</i>	9,652		
Testing		<i>True</i>	4,842	15,206	
		<i>False</i>	10,364		
			<b>Total</b>	<b>190,530</b>	
<b>(D2).</b> ArabGloss+Wiki40KTrue	Training	<i>True</i>	85,835	172,703	
		<i>False</i>	86,868		
	Validation	<i>True</i>	9,538	19,190	
		<i>False</i>	9,652		
	Testing	<i>True</i>	4,842	15,206	
		<i>False</i>	10,364		
			<b>Total</b>	<b>207,099</b>	
<b>(D3).</b> ArabGloss+Wiki200KTrue	Training	<i>True</i>	229,835	316,703	
		<i>False</i>	86,868		
	Validation	<i>True</i>	25,538	35,190	
		<i>False</i>	9,652		
	Testing	<i>True</i>	4,842	15,206	
		<i>False</i>	10,364		
			<b>Total</b>	<b>367,099</b>	
<b>(D4).</b> ArabGloss+Wiki400KTrue	Training	<i>True</i>	499,629	676,456	
		<i>False</i>	176,827		
	Validation	<i>True</i>	55,515	75,162	
		<i>False</i>	19,647		
	Testing	<i>True</i>	4,842	15,206	
		<i>False</i>	10,364		

Table 4.10 – continued from previous page

	Dataset	Pairs	Count	Total
			<b>Total</b>	<b>766,824</b>
(D5). ArabGloss+WikiGlossContext <sub>cross-related</sub>	Training	<i>True</i>	642,740	1,036,409
		<i>False</i>	393,669	
	Validation	<i>True</i>	71,416	115,157
		<i>False</i>	43,741	
	Testing	<i>True</i>	72,726	106,735
		<i>False</i>	34,009	
			<b>Total</b>	<b>1,258,301</b>
(D6). ArabGloss+WikiGlossContext <sub>false-local</sub>	Training	<i>True</i>	460,542	958,117
		<i>False</i>	497,575	
	Validation	<i>True</i>	51,172	106,458
		<i>False</i>	55,286	
	Testing	<i>True</i>	114,799	229,598
		<i>False</i>	114,799	
			<b>Total</b>	<b>1,294,173</b>
(D7). WikiGlossContext <sub>cross-related</sub>	Training	<i>True</i>	592904	899,705
		<i>False</i>	306,801	
	Validation	<i>True</i>	65,879	99,968
		<i>False</i>	34,089	
	Test	<i>True</i>	72,726	106,735
		<i>False</i>	144,535	
			<b>Total</b>	<b>1,106,408</b>
(D8). WikiGlossContext <sub>false-local</sub>	Training	<i>True</i>	399,797	799,594
		<i>False</i>	399,797	
	Validation	<i>True</i>	56,544	113,088
		<i>False</i>	56,544	
	Test	<i>True</i>	114,799	229,598
		<i>False</i>	114,799	
			<b>Total</b>	<b>1,142,280</b>

## 4.6 Summary

In this chapter, we explained the entity disambiguation and linking process. The main components were presented in section 4.1. In this chapter, we justify the need for annotated corpora in section 4.2. Then, the process of annotating and linking WoJood-NED is illustrated in the section 4.3.1. The WikiGlossContext corpus building is illustrated in section 4.4. Finally, the training, validation, and testing datasets are described in section 4.5.



## Chapter 5

# Candidate Lookup

### 5.1 Introduction

In this chapter, the candidate lookup component will be explained. Candidate lookup is the second component in the entity linking process that is responsible for returning the candidate nodes (from Wikidata) for a given named entity. In section 5.2, we present popular methodologies for retrieving candidates from knowledge graphs. In this research, we extend the approach proposed in (Sakor et al., 2020) by developing a local data store for the Arabic language and building a search engine on top of it using the Elasticsearch framework.

### 5.2 Candidate Lookup Methods

Candidate lookup could be done using many techniques, some of the popular methods, which we did not use, are:

1. **Surface form matching:** the list of possible candidates can be generated from all the possible surface forms of the mentions in a specific text. This is done using NLP techniques such as Levenshtein distance, n-grams, and normalization (Zwicklbauer et al., 2016).

2. **Aliases expansion:** a dictionary of named entity aliases can be constructed from the named entity "also named as" field in Wikidata. One competitive advantage of this method is improving the recall of the candidate lookup generation compared with sub-strings, which cannot catch similar cases (Alam et al., 2022).
3. **Prior probability computation:** generate the candidate list based on empirical probability. This is the pre-computed probability of correspondence between certain mentions and entities  $p(e|m)$ . An entity map can be generated from Wikipedia hyperlinks (Ganea & Hofmann, 2017), cross wikis (Spitkovsky & Chang, 2012) and YAGO (Hoffart et al., 2011).
4. **SPARQL query:** the named entity can be passed to a SPARQL endpoint that matches it with the *Label* filed in Wikidata. This method is used by many research that targeted Arabic language (Al-Smadi et al., 2015). One major limitation of this method is that the entities can be referred to by its aliases rather than its label. Moreover, if the named entity is not perfectly match the label then the result query will miss the named entity. Besides, many heuristics have to be included in order to overcome the mentioned limitations.

In the candidate lookup component, we have implemented a method other than the above-listed methods. We built a local store, extracted from Wikidata nodes, similar to the methodology followed by (Sakor et al., 2020). But instead of including only labels and aliases of the nodes, the local store includes the following fields: (1) the Wikidata Arabic labels, (2) their aliases, (3) the corresponding Wikidata English labels, (4) the Wikipedia site-links, (5) the Wikidata and Wikipedia descriptions for each node, and (6) the Wikidata Q-identifier. These fields will be indexed using the Elasticsearch framework <sup>1</sup> which uses the BM25 algorithm for ranking (Logeswaran et al., 2019). The Elasticsearch was then customized to best find the possible candidates for each named entity. The Elasticsearch supports the lookup of both Arabic and English search queries, though the retrieved results are only in Arabic. This means that if a

---

<sup>1</sup><https://www.elastic.co/>

named entity mentioned is in English inside and Arabic sentence, our entity linking components can look up and disambiguate this entity.

## 5.3 Index and Analyzer

The analyzer role is to inform Elasticsearch on how to index and search text. Elasticsearch comes with a wide range of built-in analyzers, which can be applied on any index without extra configuration<sup>2</sup>. In our case, a custom analyzer was built to support multilingual term search<sup>3</sup>. The analyzers' types were combined in a custom analyzer for Arabic and English.

```
1 PUT /Arabic_Analyzer
2 {
3   "settings": {
4     "analysis": {
5       "filter": {
6         "arabic_stop": {
7           "type": "stop",
8           "stopwords": "_arabic_"
9         },
10      "arabic_stemmer": {
11        "type": "stemmer",
12        "language": "arabic"
13      }
14    },
15    "analyzer": {
16      "rebuilt_arabic": {
17        "tokenizer": "standard",
18        "filter": [
19          "lowercase",
20          "decimal_digit",
21          "arabic_stop",
22          "arabic_normalization",
23          "arabic_keywords",
24          "arabic_stemmer"
25        ]
26      }
27    }
28  }
29 }
```

FIGURE 5.1: The Arabic analyzer configuration

<sup>2</sup><https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>

<sup>3</sup>The full implementation of the Analyzer can be found in the GitHub repository [https://github.com/eng-aomar/Thesis/blob/main/notebooks/wikidata\\_indexing.ipynb](https://github.com/eng-aomar/Thesis/blob/main/notebooks/wikidata_indexing.ipynb)

```

1 PUT /English_Analyzer
2 {
3   "settings": {
4     "analysis": {
5       "filter": {
6         "english_stop": {
7           "type": "stop",
8           "stopwords": "_english_"
9         },
10        "english_stemmer": {
11          "type": "stemmer",
12          "language": "english"
13        },
14        "english_possessive_stemmer": {
15          "type": "stemmer",
16          "language": "possessive_english"
17        }
18      },
19      "analyzer": {
20        "rebuilt_english": {
21          "tokenizer": "standard",
22          "filter": [
23            "english_possessive_stemmer",
24            "lowercase",
25            "english_stop",
26            "english_keywords",
27            "english_stemmer"
28          ]
29        }
30      }
31    }
32  }
33 }

```

---

FIGURE 5.2: The English analyzer configuration

The code listed in the Figures (5.1, 5.2) show the configuration of the custom analyzer.

Below is the description of how they analyze input queries:

- The standard analyzer is used to split the text into word boundaries. For example, (The Java language) is split into [The, Java, Language], and (لغة جافا/The Java language) is split into [جافا/Java, لغة/language].
- The stop-word filter removes stop words. For example, in the English analyzer, (the Java) becomes [Java] by removing (the). The same as in the Arabic analyzer, the stop-word (بـ/using) in the named entity(بـ جافا/using Java) becomes ([جافا/Java]).
- The Arabic stemmer (a built-in stemmer from the Elasticsearch) is applied to the tokenized words in the Arabic language. The same idea is applied to the tokens

emitted from the English tokenizer. For example the token [Foxes] is stemmed into [Fox], and the [مهندسون/Engineers] is stemmed into [مهندس/Engineer].

The filter in our configurations (line 18 and 22) is applied to the tokens emitted from the tokenizer to tidy up the tokens. In line 6, the filter removes the stop-words<sup>4</sup> from the Arabic language, while in line 10, the Arabic stemmer is applied to the tokenized words in the Arabic language. The same idea is applied to the tokens emitted from the English tokenizer, Figure 5.2, lines six and ten, while the possessive steamer (line 14) removes the trailing possessive 's, (e.g., Harvard's becomes [Harvard]).

These steps are important in the search process, as for example, named entities that belong to the NORP class are plural such as (مبرمجون، أطباء، مهندسون/Engineers, programmers, doctors), and thus cannot be found in Wikidata if exact matched. Hence, they have to be stemmed first into (مبرمج، طبيب، مهندس/Engineer, Programmer, Doctor), then passed as a query. The same applies to stop-words, as named entities that contain stop-words such as (كأسمبلي/As Assembly) is hard to find in Wikidata if the stop-word (ك/As) is not removed. Elasticsearch framework provides these mechanisms without the need to use heuristics to handle such cases.

## 5.4 Query Configurations

Elasticsearch queries' results are ordered according to relevancy so that records that are more relevant to the search query are given higher ranks and hence appeared on top. However, the relevancy depends more on the use case, and therefore, relevancy may depend on the application domain and needs to be tweaked to work as intended. Elasticsearch at first minimizes the number candidate results before scoring the records, this is done by enforcing a Boolean test that contains only matched records to the search query. Elasticsearch uses BM25 as a default scoring algorithm, so the score is calculated for each record and then ordered based on the rank value. In general, there are three main principles that play a role in record scoring:

<sup>4</sup>The Arabic stop-words are built-in in the Elasticsearch, but a custom stop-words could be used

1. **Term frequency (TF)**: Search term appears in the field we are looking at.
2. **Inverse document frequency (IDF)**: The more records that include a search term in the field we are looking for, the less significant that term will be.
3. **Field length**: records with short search term filed are more relevant than long fields (i.e. has many words).

```

1  body = {
2    "query": {
3      "bool": {
4        "should": [
5          {
6            "multi_match": {
7              "query": query,
8              "fields": fields,
9              "fuzziness": 1
10           }
11         },
12       ],
13       {
14         "multi_match": {
15           "query": query,
16           "fields": fields,
17           "operator": "and",
18           "fuzziness": 1
19         }
20       },
21     },
22     {
23       "match_phrase": {
24         "arlabel": {
25           "query": query,
26           "boost": 5
27         }
28       }
29     },
30     {
31       "match_phrase": {
32         "enwiki": {
33           "query": query,
34           "boost": 1
35         }
36       }
37     }
38   }
39 }

```

---

FIGURE 5.3: Query body configuration

Sakor et al., 2020 have used a simple match query to search Wikidata labels, after extending their local store with label aliases. However, a multi-term matching query (i.e. a query that contains multiple words) will use the *OR* operator by default which

will return records containing any of the terms in the query, as the *OR* operator does not take into account the position of the words. For example, the query contained the named entity (لغة جافا /Java language) may match many records that contain the word (لغة/Language or جافا/Java), although some of the matched records that contain (لغة/Language) alone may be only partially relevant. In addition, the totally matched records to the search query may give a score less than records that contain the search query as a sub-string. Elasticsearch is flexible when it comes to query building. It can offer an exact match of the search term using *AND* operator instead of the default (*OR*), but using *AND* operator will limit the search space to only results that match exactly the search term. For example if the term is (لغة جافاسكريبت /JavaScript language), searching Wikidata using the exact term will give no results since there is no match with the exact label or its aliases in Wikidata. To overcome this limitation, we build a query that favors exact matches if exists and ranked it higher, but also returns other relevant candidates. This is achieved by using "Boolean query" to combine *OR* and *AND* operators, and multi-match query to search within multiple fields. That is, fields including Wikidata aliases and Site-link labels in Arabic and English, without the need to search in each field separately. The more-matches-is-better approach is followed when using the "Should" clause in the Boolean query so that each clause's score will contribute to the final score of each returned result. To favour clauses in the query, we boosted (i.e., a number is given to favour the clause, the higher is the more favoured, line 26, and 34) individual clauses regarding the clause we want to be ranked higher. For example, match with *AND* operators is given a higher boost value, so it appears first in the results. Moreover, the fuzziness parameter in the query configuration 5.3 is used to handle typos which are common in websites' discussions such as Quora, Stack-overflow, etc. For example, the word (بايثون/Python) can be misspelled with (بيثون/Python), and the word( asp.net) with( asb.net). Using the fuzziness attribute, Elasticsearch can handle such cases by taking into account the possibility of characters missing or appeared in the wrong order (e.g., مترجم/Translated is misspelled with مترج/Translated). The value is determined when building the query. In our case, one character only is considered

to be reordered. This value is chosen to limit the number of records returned, and at the same time handle common typos which are popular.

## 5.5 Candidate Lookup Query Evaluation

In this section, we provide an evaluation of the Candidate lookup component based on the query configuration described in section 5.4. This evaluation is done using 598 unique named entities (unique surface forms) from the Wojood-NED corpus (e.g. بايثون/python, and بيثون/python are two unique surface forms for the same named entity "python"). This was done by passing the named entities to the query and limiting the query size to the top seven results. Then, glosses retrieved from Wikidata were labeled to *True* if the retrieved Wikidata IQ-identifier for the named entity matched the Wikidata IQ-identifier found in Wojood-NED for that named entity, otherwise, glosses were labeled with *False*.

The bar charts in Figure 5.4 depict the results of the query evaluation that was performed using 598 unique named entities. The first chart 5.4a shows the percentage of the correct match (i.e. the *True* candidates) per order in the candidate list returned by the query. Overall, the majority of correct matches (68%) came in the first order in the candidate list. Followed by (16%) of the correct matches which came in the second order. While only (15%) of the correct matches occupied the bottom five positions (i.e third, fourth, fifth, sixth, and seventh). In general, the size of the candidate list returned by the query varied between at most four candidates and at least one unique candidate. However, the correct match came in advanced orders, most notably first in the list of candidates with a high percentage.

In terms of query size, the second chart 5.4b shows the percentage of candidate lists' size returned by the query. The majority (28%, and 24%) of the named entities passed to the query have four and five candidates respectively. Moreover, (14%) of the named



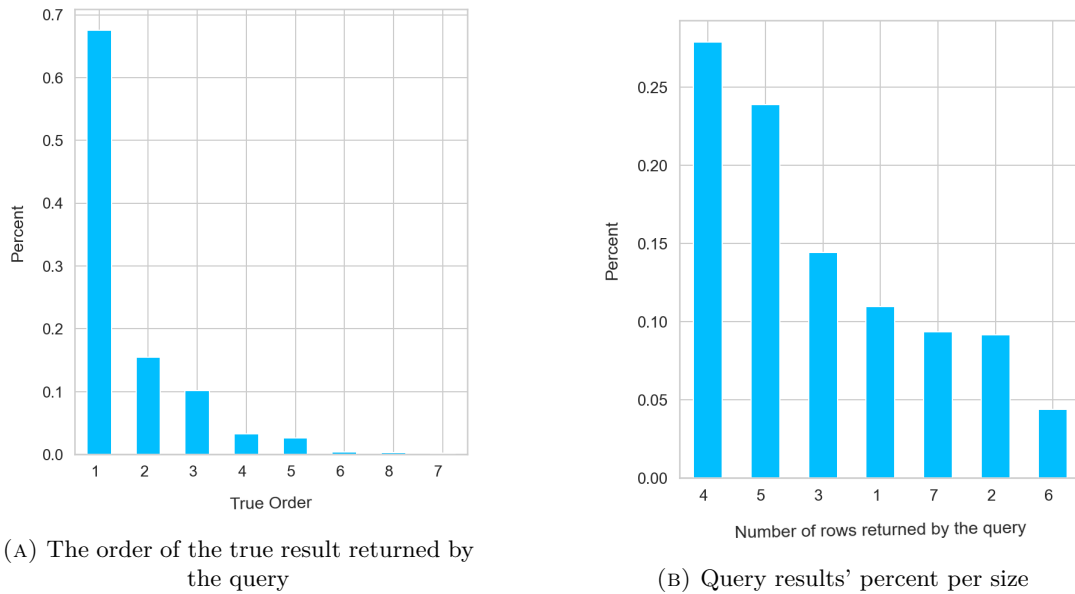


FIGURE 5.4: Achieved results of applying search using the fine-tuned query

entities have three candidates. While (11%) of the named entities have unique candidates returned by the query. Only (9%) of the named entities have seven or two candidates returned by the query. While only 4% of the named entities returned six candidates.

Table 5.1 shows statistics about the results of querying 598 unique named entities using the query configuration presented in Figure 5.3. The query configuration returned the correct match with an accuracy of 419 (70%). While the rest 117 (30%) of the named entities do not have a correct match within the list of candidates.

To understand the reasons behind this percentage (i.e. 30% of the queries do not have correct match within the candidate list), we further analyzed the results and we found that the majority 114 (80%) of the named entities in this category do not have a matching node on Wikidata. While 27 (15%) of the results contains typos that misled the search component, for example, the word Google was written as *گوگل*/Google.

TABLE 5.1: Query configuration statistics

Category		Count
<b>Has correct match</b>		<b>419 (70%)</b>
<b>Does not have correct match</b>	Typos	27 (15%)
	Does not have Wikidata node	144 (80 %)
	Have Wikidata node	8 (5%)
	<b>Total of no match</b>	<b>179 (30%)</b>
<b>Count of unique named entities</b>		<b>598</b>

However, *كوكل*/Google is not one of the aliases for the named entity *جوجل*/Google. Finally, the minority 8 (5%) of the node have Wikidata nodes but the query could not return the correct (i.e. the named entity gloss) match within the query result. In our Query configuration, we handled typos by using the fuzziness attribute (refer to section 5.4). However, this attribute can not handle all types of user typos. Besides, increasing the number associated with this attribute can increase the irrelevant results, and cause error propagation to the next component (NED).

**Outlook:** The overall accuracy can be enhanced by supporting the Arabic content on Wikidata and using auto-correction tools to handle typos of the end users which is a common problem in the QA websites.

## 5.6 Summary

In this chapter, the candidate lookup module was explained. The sections (5.3, 5.4) respectively explain the configuration process of the Elasticsearch framework. Finally, section 5.5 shows that the configuration is efficient for the quires. In the next chapter, the process of fine-tuning and evaluating BERT models to select the best-performing model for the NED component will be illustrated.

## Chapter 6

# Entity Linking

### 6.1 Introduction

Entity linking (EL) is the process of deciding which node in a knowledge graph (e.g. Wikidata) corresponds to a given named entity in the sentence. For a recognized named entity in the text, possible candidate nodes are looked up from the kG (Wikidata in our case). These nodes are then fed to the NED component to determine the correct node. The problem of NED is treated in this thesis as WSD problem. In this chapter, the NED problem is described in section 6.2, and the environmental setup needed to conduct our experimentation using BERT is explained in section 6.3. After all, the results of experiments are reported and discussed in section 6.4. To select the best performing model from all the fine-tuned models, we did a disambiguation evaluation for entity linking, which is explained in subsection 6.4.3. After that, the best-performing model was deployed as a linguistic API and used in the implementation of the two components, and a demo was presented for the linking process in section 6.5. Finally, in section 6.6, a couple of scenarios were presented to demonstrate how the entity linking components can be used to disambiguate software-specific named entities based on their contexts.

## 6.2 Entity Disambiguation

To solve the NED problem, as said earlier, we intend to use the pre-trained Arabic BERT models. To use BERT, we treat the NED problem as a sentence-pair (i.e. context-gloss) binary classification problem. The binary text-classification architecture in BERT is used to solve this problem. This is similar to the approach followed by Kannan Ravi et al., 2021. BERT can distinguish two tasks in the text classification. The first is one sentence classification, such as sentiment analysis, where one sentence is classified into two or multiple classes. The second is pair-sentence classification, in which the BERT model is fed with two sentences (sequences), and a special token called [SEP] is used to separate the two sentences and tell BERT that this is a pair-sentence classification. Thus the dataset must have three columns, in our case (contexts, glosses, and label). Another special token [CLS] is added as a first token at the beginning of the first sequence. This token stands for classification, and represents a sentence-level classification. The input sequences must be tokenized using the suitable BERT tokenizer. Tokens are an essential component of the NLP pipeline. They serve one purpose: to convert the input text into a form that can be processed by the pre-trained BERT model. Since BERT can only handle numbers, the job of the tokenizer is to provide the model with the correct input format by translating the text input into numeric data (Figure 6.1).

```
1 ids= tokenizer.convert_tokens_to_ids(tokens)
2 print(ids)
3
4 [2, 1351, 1898, 5944, 40947, 181, 5860, 2365, 3, 40947,
5 | 181, 634, 5944, 29192, 18874, 300, 6503, 3]
```

FIGURE 6.1: Tokens to ids representation of the sequence

Transformers library offers tokenizers for all models' architectures, they come in two flavors, a full Python implementation, and a fast, Rust-based implementation. The fast implementation shows particularly significant acceleration when making batched tokenization. In all experiments, the fast tokenizer is used with dynamic patching, as

longer sequences are disproportionately expensive because attention is quadratic to the sequence length (Devlin et al., 2018). To speed up pre-training in our experiments, the max-length of the sentences is considered for each batch rather than the whole dataset by using *DataCollatorWithPadding*<sup>1</sup> offered by transforms library. For each batch, the [PAD] token is added to the sentences that are shorter than the longest sentence in the batch, so if we have a batch with a max-length sentence equal to 120 tokens in a dataset that has a max-length sentence equals to 512 tokens, then the sentences in that batch are padded to the max length of 120 rather than 512. This process is done per batch.

There are several pre-trained Arabic BERT models (explained in section 2.3.2), which we will use to conduct our experiments in this chapter. That is, we will fine-tune a set of pre-trained Arabic BERT models using the different dataset variations we presented in Table 4.10 and produced from the WikiGlossContext corpus and combined them with ArabGlossBERT (Al-Hajj & Jarrar, 2021). The goal of conducting multiple experiments is to, first, choose the best pre-trained Arabic BERT model and then adopt it for the rest of the work. Second, there is a need to experiment with the different datasets variations in order to select the best-performing variation from the available sets. As a result, a total of eleven experiments were conducted, and two entity linking evaluations were performed on the eleven models to select the best performing one. The following sections provide the needed information about the experimental setup and the results of the experiments.

### 6.3 Environmental Setup

The Google Colaboratory aka *Google Colab* was used to conduct the experiments. This platform is offered by Google research and is available in three editions. The Colab free charge for use, Colab Pro, and the Colab Pro++. The three editions offer GPU and RAM, with some difference, as Colab Pro offers high Ram, while Colab Pro++

<sup>1</sup>refer to [https://huggingface.co/docs/transformers/main\\_classes/data\\_collator](https://huggingface.co/docs/transformers/main_classes/data_collator)

offers background execution, high Ram, and faster GPUs. In this research, the Google Colab Pro++ is used to overcome the time connection limitation in Google Colab by making use of the background execution feature (Table 6.1). Python programming language version (3.7.13) was used to implement the experiments because of its popularity, preference, and the plethora of NLP and ML libraries support compared to other languages.

TABLE 6.1: NED task environment setup

Type	Specification
<b>GPU</b>	Tesla P100-PCIE
<b>RAM</b>	52 GB
<b>CUDA Version</b>	11.2
<b>Disk</b>	166.83 GB
<b>Operation System</b>	Ubuntu 18.04 LTS
<b>Programming Language</b>	Python 3.7

### 6.3.1 Experiment Hyperparameters

Table 6.2 presents the hyperparameters used in fine-tuning the BERT models for all the experiments. These values were selected based on the recommendation by the original BERT paper, which stated large datasets (larger than 100K) are less sensitive to hyperparameters change (Devlin et al., 2018). While the rest of the hyperparameters are kept at their defaults. The base configuration of the three models in the experiments (Aarabertv02, bert-base-multilingual, UBC-NLP/MARBERTv2) is selected due to the computational limitations. Besides, large models do not necessarily give better results (Abdelali et al., 2021; Inoue et al., 2021). The fine-tuning is done using a custom *Trainer* class that subclass the *compute\_loss* function as weighted loss to handle the unbalanced training set <sup>2</sup>. The learning rate =2e-5 and batch size=16 were selected based on a limited grid search, with early stopping if there is no improvement on the validation loss metric after three epochs. On average, the models converged around epoch five.

<sup>2</sup> refer to [https://huggingface.co/docs/transformers/main/en/main\\_classes/trainer#trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#trainer)

TABLE 6.2: Hyperparameters values

Parameter	Value
Number of train epochs	10
Evaluation strategy	epoch
Evaluation steps	50
Seed	42
Optimizer	AdamW
learning rate	2e-5
batch size	16
Early stopping monitor	Validation loss
Early stopping patience	3

### 6.3.2 Experiments' Tracking

In order to track the model building and experimentation's results, we used available MLOps platforms, such as [Weights & Biases](#) and [Neptune.ai](#). These platforms offer a single place to log, store, query, display, organize, and compare all model metadata. The following artifacts are tracked per experiment:

1. Checkpoint used.
2. Dataset Information, including sets size, classes frequencies, files paths, and huggingface repository..
3. Experiment hyperparameters used in fine-tuning the model.
4. The fine-tuned model Huggingface repository with model card documentation.
5. Train/Validation loss values.
6. Experiment predictions [ accuracy, precision, recall, f1 score ].
7. Confusion matrices.

The complete experiments' tracking projects are available on [Neptune](#), and [Weights & Biases](#)

## 6.4 Experiments' Results

This section includes all the experiments conducted and their results. In all the experiments presented, the ArabGlossBERT (Al-Hajj & Jarrar, 2021) test set was used to evaluate all the models. Moreover, The models that were fine-tuned using the datasets D5, D6, D7, and D8 were evaluated using the WikiContextGloss test described in the Table 4.10. The accuracy and the macro average of three measures, accuracy, recall, and f1-score measures of system performance were used. The fine-tuned models from the experiments were also evaluated against the Wojoood-NED corpus to link the named entities with their nodes on Wikidata, the accuracy metric was used to evaluate the linkage performance of the fine-tuned models. Table 6.6 shows a summary of information about the fine-tuned models, the information includes (model number, the Arabic BERT model used to fine-tune the model, and the dataset used to fine-tune the model). The models will be denoted by their model numbers in the rest of this thesis.

### 6.4.1 Experiments' Set One: BERT Models

The objective of this experiment is to study the effect of using different Arabic BERT models on the performance of the fine-tuned models. This experiment is done using the WikiGlossContext<sub>cross-related</sub> dataset described in Table 4.8, using the hyperparameters defined in Table 6.2, and three Arabic BERT models (Arabertv02, bert-multilingual, MARBERTv2).

Table 6.3 shows relatively close results for the three used pre-trained models when evaluated on the WikiContextGloss test set. However, there is a significant difference in the results when the ArabGlossBERT (Al-Hajj & Jarrar, 2021) was used. The results show the model that was fin-tuned using Aarabertv02 and denoted by  $M_1$  outperforms the others with (16%, and 11%) increase in the (f1-score and accuracy) respectively when evaluated using the ArabGlossBERT (Al-Hajj & Jarrar, 2021) test set. But unfortunately, the results on the ArabGlossBERT(Al-Hajj & Jarrar, 2021) were disappointing



TABLE 6.3: Achieved results (%) after fine-tuning three different Arabic BERT models on WikiGlossContext<sub>cross-related</sub> dataset (Table 4.8)

Model#		WikiGlossContext <sub>cross-related</sub>			ArabGlossBERT		
		True	False	Avg	True	False	Avg
$(M_1)$ . <b>AraBERTv2</b>	(D7.) Precision	98	97	97	35	78	57
	Recall	99	95	97	83	29	56
	F1-score	98	96	<b>97</b>	49	42	<b>46</b>
	Accuracy	<b>97</b>			<b>46</b>		
$(M_2)$ . <b>bert-multilingual</b>	(D7.) Precision	97	95	96	32	71	51
	Recall	97	94	96	93	07	50
	F1-score	97	94	96	48	13	30
	Accuracy	96			35		
$(M_3)$ . <b>MARBERTv2</b>	(D7.) Precision	97	96	96	32	71	51
	Recall	98	93	96	93	07	50
	F1-score	98	95	96	48	13	30
	Accuracy	97			35		

in all the experiments in the first set. The first possible reason for that disappointing result could be the nature of the data, as the ArabGlossBERT(Al-Hajj & Jarrar, 2021) is a lexicon dataset that was extracted from (150) Arabic lexicons, and contains concepts rather than named entities. Unlike the model which was fine-tuned on different data types (Wikidata and Wikipedia) descriptions. The second reason could be the small size of the ArabGlossBERT test (only 15,206 pairs), compared to the WikiGlossContext<sub>cross-related</sub> dataset (1,228,215 pairs). Moreover, the classes distribution for the ArabGlossBERT test which has more (*False pairs*, 10,364) than the (*True pairs*, 4,842). While the WikiGlossContext<sub>cross-related</sub> dataset as described in Table 4.8, section 3.4, shows that the *True pairs* are much more than the *False pairs*. It is important to note that getting a bad performance on ArabGlossBert(Al-Hajj & Jarrar, 2021) test set does not mean the model is bad in linking. But, it means the model can not do well in classifying concepts not named entities. Hence, the models real performance will be judged in the linking (disambiguation) evaluation in section 6.4.3. The results of the experiments contributes to the research question  $RQ_2$  part which asks about the best performing Arabic pre-trained model. Based on the results the **Aarabertv02** model will be used in fine-tuning all models in the upcoming experiments.

### 6.4.2 Experiments' Set Two: Different Datasets

The objective of this set of experiments is to measure the effect of using different datasets (described in Table 4.10.) on the performance of the fine-tuned BERT models. Table 4.10 illustrates the results of six experiments where D0, D1, D2, D3, D4, D5, D6 and D8 datasets were used. ( $M_4$ ) denotes the ArabGlossBERT (Al-Hajj & Jarrar, 2021) which was trained on D0 dataset. The fine-tuned models ( $M_4$  to  $M_8$ ) were evaluated against the ArabGlossBERT test set only, as these models are fine-tuned using the combined datasets (D1 to D4) where only *True* pairs were added from the WikiGlossContext *True* pairs before generating the *False* pairs to the ArabGlossBERT (Al-Hajj & Jarrar, 2021) dataset. The results show that the more pairs from the WikiGlossContext are combined with the ArabGlossBERT the less performance is. The fine-tuned model on D1 (ArabGlossBERT+Wiki23kTrue) dataset achieved the highest accuracy (82%). Followed by the fine-tuned model on D2 (ArabGlossBERT+Wiki40kTrue) with (80% accuracy. The model fine-tuned on the WikiGlossContext<sub>false-local</sub> achieved the worst results accuracy (60%)<sup>3</sup>.

With reference to the results reported in in Table 6.4, adding **23k** *True* pairs to the ArabGlossBERT (Al-Hajj & Jarrar, 2021) achieved the highest accuracy and f1-score (82%) among the other experiments. In a conclusion to the above results, adding *True* pairs to the ArabGlossBERT (Al-Hajj & Jarrar, 2021) dataset does not yield in achieving better accuracy compared to the results of the original ArabGlossBERT (Al-Hajj & Jarrar, 2021) denoted by  $M_4$ , which achieved 84% accuracy. Besides, increasing the number of *True* pairs gradually does not necessarily give the same effect on the accuracy obtained. For example, adding **40k**, and **200K** *True* decreases the accuracy to (80%, and 73%) respectively, while adding **400K** *True* decreases the accuracy to 76% compared to adding **23k** *True* pairs.

---

<sup>3</sup>This dataset does not combined with D0 (ArabGlossBERT (Al-Hajj & Jarrar, 2021)).

The experiments in this set contribute to the research question  $RQ_2$  part which stated (Given a word in a sentence (whether this word refers to a named entity or a concept, i.e., unnamed), what is the most accurate way (different datasets) to disambiguate the semantics of this word with a node in a knowledge graph like Wikidata?. From the results presented in Table 4.10, it is clear that the best-performing model to classify concepts (non named entities) is the ArabGlossBERT (Al-Hajj & Jarrar, 2021) denoted by  $M_4$ . This result confirms that combining lexicons data (concepts) with named entities (different data sources) does not increase the accuracy of classifying concepts, instead, the results get even worse. To prove the other claim, whether models trained on ArabGlossBERT or combined with it can give better results in linking named entities to a knowledge graph, in the next section, a disambiguation evaluation will be conducted to select the best performing model from the pre-trained models in sections (6.4.1, and 6.4.2).

TABLE 6.4: Achieved results (%) after fine-tuning Arabertv02 using different datasets (Table 4.10) using ArabGlossBERT test

Model #	Dataset		False	True	Avg	Accuracy
$(M_4)$ .	(D0). ArabGlossBERT (Al-Hajj & Jarrar, 2021)	precision	85	81	83	
		Recall	93	66	80	<b>84</b>
		F1-score	89	72	81	
$(M_5)$ .	(D1). ArabGloss+Wiki23KTrue	precision	85	73	79	
		Recall	86	79	78	82
		F1-score	70	87	<b>82</b>	
$(M_6)$ .	(D2). ArabGloss+Wiki40KTrue	precision	83	72	77	
		Recall	88	62	75	80
		F1-score	86	67	76	
$(M_7)$ .	(D3.) ArabGloss+Wiki200KTrue	precision	72	83	78	
		Recall	98	20	59	73
		F1-score	83	33	58	
$(M_8)$ .	(D4.) ArabGloss+Wiki400KTrue	precision	76	78	77	
		Recall	95	35	65	76
		F1-score	84	48	66	

The model  $M_9$  was evaluated using two test sets, the first is the WikiGlossContext<sub>cross-related</sub>

denoted by D5 (see Table 4.8).  $M_9$  achieved (97%) on the three metrics (precision, recall, and accuracy), and (96%) on the f1-score. Moreover, the results presented on Table 6.5 show that the model achieved (76%, 69%, 70%, and 77%) on (precision, recall, f1-score and accuracy) respectively when evaluated on the ArabGlossBERT (Al-Hajj & Jarrar, 2021) test .

The models  $M_{10}$  and  $M_{11}$  were also evaluated on the WikiGlossContext<sub>false-local</sub> test set (see Table 4.9).  $M_{10}$  and  $M_{11}$  achieved 93% and 94% accuracy on all metrics. However, the model  $M_{10}$  that was trained on a combined dataset D6 achieved better performance on the ArabGlossBERT (Al-Hajj & Jarrar, 2021) compared to the model  $M_{11}$  which was trained on the WikiGlossContext<sub>false-local</sub> dataset alone.

In short, the results presented in Table 6.5 supports our findings in that combining the WikiGlossContext with ArabGlossBERT (Al-Hajj & Jarrar, 2021) does not improve the performance of the fine-tuned models on the WSD task to classify non-named entity mentions aka ”concepts”.

TABLE 6.5: Achieved results (%) after fine-tuning Arabertv02 using different datasets (Table 4.10 on WikiContextGloss and ArabGlossBERT tets

Model#		WikiGlossContext			ArabGloss			
		True	False	Avg	True	False	Avg	
$(M_9)$	(D5.)	Precision	97	97	97	78	74	76
	ArabBERTGloss +	Recall	99	94	96	92	54	69
	WikiGlossContext <sub>cross-related</sub>	F1-score	98	96	97	82	65	70
		Accuracy	<b>97</b>			<b>77</b>		
$(M_{10})$	(D6.)	Precision	90	98	94	61	77	69
	ArabBERTGloss +	Recall	98	88	93	45	87	66
	WikiGlossContext <sub>false-local</sub>	F1-score	94	92	93	52	45	67
		Accuracy	93			73		
$(M_{11})$	(D8.)	Precision	90	98	94	38	71	55
	WikiGlossContext <sub>false-local</sub>	Recall	98	89	94	41	69	55
		F1-score	94	93	94	40	70	55
		Accuracy	94			60		

To sum up all the metadata about the fine-tuned models, the eleven pre-trained models’ (Table 6.3, and Table 6.4) information are summarized in Table 6.6 and will be denoted

in the next section by the model number.

TABLE 6.6: A summary of the pre-trained models’ information

Model #	Model used	Dataset
$M_1$	AraBERTv2	(D7.)WikiContextGloss <sub>cross-related</sub>
$M_2$	bert-multilingual	(D7.)WikiContextGloss <sub>cross-related</sub>
$M_3$	MarBERTv2	(D7.)WikiContextGloss <sub>cross-related</sub>
$M_4$	AraBERTv2	(D0). ArabGlossBERT(Al-Hajj & Jarrar, 2021)
$M_5$	AraBERTv2	(D1).ArabGloss+Wiki23KTrue
$M_6$	AraBERTv2	(D2).ArabGloss+Wiki40KTrue
$M_7$	AraBERTv2	(D3).ArabGloss+Wiki200KTrue
$M_8$	AraBERTv2	(D4).ArabGloss+Wiki400KTrue
$M_9$	AraBERTv2	(D5.)ArabGloss+WikiGlossContext <sub>cross-related</sub> )
$M_{10}$	AraBERTv2	(D6.)ArabGloss+WikiGlossContext <sub>false-local</sub> )
$M_{11}$	AraBERTv2	(D8.)WikiGlossContext <sub>false-local</sub>

### Remarkable Notes About the Results

Regarding the results achieved by the fine-tuned models ( $M_1$ ,  $M_2$ ,  $M_3$ ,  $M_9$ ,  $M_{10}$ , and  $M_{11}$ ) presented in Table 6.3 and Table 4.10. We can see that the achieved accuracy is high (97%, 96%, 97%, 97%, 93%, and 94%) respectively on the WikiGlossContext test set. This test set is taken from the WikiGlossContext corpus, which is part of the data that the models were trained on during the fine-tuning phase. The data in this corpus has the same characteristics (glosses and contexts are taken from Wikidata and Wikipedia); thus, the fine-tuned models achieved high results on the test set. Regardless of the fact that the test set was split in a way that prevented information leaks between the three sets (train, validation, and test) <sup>4</sup>.

<sup>4</sup>for more details on the data sets splitting please refer to chapter 4, section 4.5

Because of these high accuracy achieved, we studied the nature of the dataset pairs in-depth, where we noticed similarities between glosses taken from Wikidata’s description and contexts taken from Wikipedia (see picture 6.3). The first sentence taken as the context in our case from Wikipedia often forms a sentence similar to the definition taken from Wikidata, although they do not exactly match. Thus we came to the conclusion that the data is over-fitted. However, we can not avoid this case, because we could not find another contexts’ source that contains the named entity, because these pairs are difficult to extract and there are no other datasets to use. However, in section 6.4.3, we evaluate the models on an external dataset, we called (Wojood-NED). The *True* pairs were formulated from the contexts found on Wojood-NED while the glosses were taken from two different sources (Wikidata and Wikipedia). The results are presented in Table 6.9 and 6.10, where the best-performing model  $M_1$  achieved 71% on Wikidata glosses, while  $M_5$  achieved 68 % on Wikipedia glosses.

We note that the accuracy decreased when evaluating the fine-tuned models against the external dataset (Wojood-NED), and this decrease is justified, given that the models were primarily trained on over-fitted data, as mentioned earlier. While the data source differed in the external dataset because the contexts used are not necessarily similar to the definitions, as in the WikiGlossContext dataset on which the models were trained on. Details of this evaluation are presented in the following sections.

### 6.4.3 Experiments’ Set Three: Disambiguation Evaluation

In this section, we evaluated which of the eleven fine-tuned models (see Table 6.6) performs better in entity disambiguation using the named entities from the Wojood-NED corpus (see section 4.3.2).

Notice that the evaluation of entity disambiguation is different from context-gloss binary classification evaluation conducted in sections (6.4.1 and 6.4.2). Previously each pair of sentences (context, gloss) was classified whether it is *True* if the pair match, or

*False* if it does not. The purpose of the disambiguation evaluation, in this section, is to link the extracted named entity with its corresponding node in Wikidata. This evaluation was done using two components only (candidate lookup and NED). Though, instead of the NER tagging component, the named entities and their contexts were used from the Wojood-NER corpus<sup>5</sup>. Recall that fine-tuning the BERT model for named entity recognition is beyond this thesis's objectives. Thus NER was done manually using the aforementioned corpus which contains software-specific tags. To do this, we take each of the extracted named entities in the Wojood-NED corpus and pass it to the Elasticsearch lookup component. The lookup component retrieves all possible candidates from Wikidata. The list of candidates is then passed to the NED component as pairs (context contained named entity, and candidate glosses retrieved from the Wikidata). The NED component calculates the raw prediction for each label (*True*, *False*) for the list of pairs. The raw predictions are then converted to probabilities using a softmax layer. Finally, we return the Wikidata Q-identifier with the maximum probability of the *True* class. Recall that the Wojood-NED corpus was manually labeled with the correct Wikidata identifier for each named entity and its NER class tag. Based on that, the overall accuracy for each of the eleven fine-tuned models is calculated separately, in addition to the accuracy per NER class. This evaluation was done in two ways: (1) we considered the Wikidata description field as a gloss, and (2) we considered the Wikipedia summary (first one to two sentences) as a gloss. The Tables (6.7, and 6.8) show the overall accuracy using only two components (the candidate lookup and the NED component). While the Tables (6.9, and 6.10) show the results of the two evaluations on the NED component alone, the best achieved results are highlighted in bold.

---

<sup>5</sup>This thesis did not implement the NER component as a BERT model API. However, for evaluation purposes, we manually annotated the Wojood-NED corpus with the six new software tags as illustrated earlier in the methodology chapter 4, section 4.3.1.

### Overall Disambiguation Evaluation

To evaluate the overall accuracy of named entity disambiguation, the three components (NER, candidate lookup, and the NED) are needed to be evaluated together. Since we did not implement the NER ourselves and we used existing data software-specific named entities from Wojood-NED. In this section, only software-specific named entities' classes will be considered in the evaluation of the two components (Candidate lookup and NED). In addition to the evaluation of the NED component in the following section. Our evaluation in this section includes only two components. We used the dataset Wojood-NED (refer to chapter 4, section 4.3) for evaluation. This was done using 735 of *True* and *False* pairs on Wikidata, and 727 pairs on Wikipedia (since not all nodes on Wikidata are linked to Wikipedia, some of the named entities do not have Wikipedia descriptions, and these named entities were not considered). We took a candidate entity and pass it to the candidate lookup, then we retrieved the list of all candidates. For each candidate entity, we form the pairs by contacting the context of the named entity and its candidates returned by the candidate lookup. After that, the NED component is used to disambiguate them. We ranked the retrieved results from the NED after converting the raw predictions into probabilities using a soft-max layer. After that, the node with the highest probability from the *True* label is linked to the named entity. In this evaluation, all the named entities whether they have a *True* gloss or do not were used to assess the overall accuracy.

Recall that the candidate component was evaluated alone in chapter 4.2, section 5.5 and achieved an accuracy of (70%). Besides, the NED component was evaluated alone in section 6.4.3 of this chapter, and achieved an accuracy of (84%) using the model denoted by  $M_5$  using glosses from Wikipedia on an external dataset (Wojood-NED).

The Tables (6.7 and 6.8) depict the overall accuracy of only two main components (the candidate lookup and the NED). It is clear from the results in the two tables that the overall accuracy, in general, was not high regardless of the source of the glosses (from



Wikidata or Wikipedia). Moreover, The overall accuracy of the models in the two tables was close with an advantage when the glosses of Wikidata were used. Using Wikipedia glosses the model denoted by  $M_5$  achieved the highest accuracy (74%). While the model denoted by  $M_1$  achieved the highest accuracy (66%) using glosses from Wikidata.

TABLE 6.7: Achieved results(%) per class after evaluating the two components (Candidate lookup and NED) on the Wojood-NED corpus using Wikidata description as a gloss

Entity Class	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$	$M_{10}$	$M_{11}$
APPLICATION (26)	<b>69%</b> <b>(18)</b>	35% (9)	27% (7)	46% (12)	<b>69%</b> <b>(18)</b>	31% (8)	31% (8)	35% (8)	50% (9)	42% (13)	50% (11)
FRAMEWORK (130)	<b>62%</b> <b>(80)</b>	62% (74)	57% (68)	52% (79)	51% (66)	51% (60)	46% (50)	38% (78)	60% (66)	52% (67)	38% (50)
PL (300)	77% (232)	61% (183)	51% (154)	59% (178)	78% (235)	62% (185)	53% (158)	54% (163)	<b>88%</b> <b>(203)</b>	60% (181)	64% (191)
PLATFORM (23)	52% (12)	30% (7)	35% (8)	<b>70%</b> <b>(16)</b>	26% (6)	17% (4)	9% (2)	30% (7)	48% (11)	48% (11)	48% (11)
PP (6)	<b>67%</b> <b>(4)</b>	50% (3)	17% (1)	17% (1)	33% (2)	17% (1)	0% (0)	50% (3)	50% (3)	17% (1)	17% (1)
SD (57)	<b>72%</b> <b>(41)</b>	19% (11)	21% (12)	63% (36)	23% (13)	9% (5)	5% (3)	47% (27)	56% (32)	61% (35)	60% (34)
WEBSITE (193)	50% (97)	34% (65)	33% (64)	42% (82)	45% (87)	32% (62)	35% (67)	39% (75)	42% (81)	<b>52%</b> <b>(100)</b>	38% (73)
<b>Overall Accuracy:</b> <b>Total count: 735</b>	<b>66%</b> <b>(484)</b>	48% (352)	47% (344)	55% (404)	58% (427)	44% (325)	39% (288)	49% (361)	55% (405)	56% (408)	50% (371)

TABLE 6.8: Achieved results(%) per class after evaluating the two components (Candidate lookup and NED) on the Wojood-NED corpus using Wikipedia description as a gloss

Entity Class	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$	$M_{10}$	$M_{11}$
APPLICATION (26)	58% (15)	46% (12)	50% (13)	23% (6)	<b>65%</b> <b>(17)</b>	62% (16)	62% (16)	42% (11)	38% (10)	42% (11)	27% (7)
FRAMEWORK (119)	65% (77)	<b>67%</b> <b>(80)</b>	59% (70)	60% (71)	63% (75)	45% (54)	39% (46)	61% (72)	65% (77)	38% (45)	48% (57)
PL (297)	66% (197)	57% (170)	61% (180)	64% (189)	<b>84%</b> <b>(284)</b>	70% (209)	42% (126)	61% (182)	53% (157)	50% (149)	65% (192)
PLATFORM (23)	<b>70%</b> <b>(16)</b>	52% (12)	48% (11)	22% (5)	43% (10)	17% (4)	17% (4)	39% (9)	26% (6)	61% (14)	65% (15)
PP (6)	33% (2)	67% (4)	33% (2)	0% (0)	50% (3)	17% (1)	17% (1)	<b>83%</b> <b>(5)</b>	50% (3)	17% (1)	17% (1)
SD (56)	27% (15)	25% (14)	27% (15)	13% (7)	<b>88%</b> <b>(49)</b>	66% (37)	77% (43)	41% (23)	21% (12)	9% (5)	46% (26)
WEBSITE (200)	<b>51%</b> <b>(101)</b>	44% (87)	<b>51%</b> <b>(102)</b>	37% (73)	49% (98)	30% (60)	24% (48)	38% (75)	36% (71)	34% (67)	45% (89)
<b>Overall Accuracy:</b> <b>Total count: 727</b>	58%↓ (423)	52%↑ (379)	54%↑ (393)	48%↓ (351)	<b>74%↑</b> <b>(536)</b>	52%↑ (381)	39% (284)	52%↑ (377)	49%↓ (354)	40%↓ (292)	53%↑ (387)

The summary of two Tables (6.7, and 6.8) results came as follows:

- The two components together (the candidate lookup and the NED) achieved the highest accuracy (74%) when using the model denoted by  $M_5$  in the NED component, and glosses from Wikipedia.
- The two components together (the candidate lookup and the NED) achieved the highest accuracy (66%) when using the model denoted by  $M_1$  in the NED component, and glosses from Wikidata.
- Table 6.7 results showed that  $M_1$  achieved the highest accuracy of ( 62%, 67%, and 72%) in the classes (FRAMEWORK, PP, and SD) respectively. Besides, the model achieved the highest accuracy with a tie with  $M_5$  in the APPLICATION class with an accuracy of 69%.
- The bar chart 6.2 depicts the change in the overall change accuracy of the two components together (Candidate lookup and the NED) when glosses were taken from Wikidata vs Wikipedia. It is clear from the chart that some classes' accuracy increased while others' accuracy increased when Wikipedia's glosses were used.

The SD class achieved the highest accuracy change (65%). As the the overall accuracy decreased from 88% using Wikipedia glosses to 23% using Wikidata glosses (refer to Table 6.7, and Table 6.8).

By investigating the results we found that the named entity HTML was linked to the node HTML5(Q2053) when the Wikidata description was used rather than the correct match HTML(Q8811). While, when the Wikipedia description was used, the model could disambiguate the named entity and link it to the correct node HTML(Q8811). This named entity is the most repetitive one in its class and was linked wrong 38 times when the Wikidata description was used.

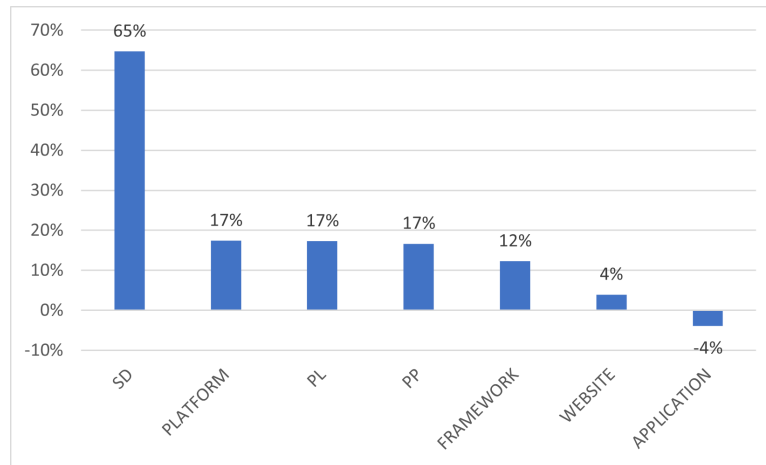


FIGURE 6.2: The overall accuracy change for the model  $M_5$  when using Wikidata glosses (Table 6.7) vs using Wikipedia glosses (Table 6.8)

Generally, the main two reasons for the poor linking results are: first, due to errors propagating from the candidate lookup component to the NED. Recall that the candidate lookup component alone achieved 70% accuracy. This is due to the fact that 80% of the named entities that does not have correct match do not have a corresponding Wikidata node (i.e. *True* gloss). Besides, User typos came in second place with a percentage of 16%. These root causes affected the candidate lookup component and caused errors to propagate to the NED component. Recall that we transferred the NED problem to a WSD binary classification problem in a similar way followed by (Al-Hajj & Jarrar, 2021; Kannan Ravi et al., 2021). In this method, the NED component retrieved the correct gloss with the highest probability of the *True* label regardless of the fact it has a higher probability than the *False* label or not. Wherefore, In the case of all passed candidates (glosses) are *False*. The NED component will retrieve a wrong gloss based on the aforementioned mechanism.

**Outlook:** This problem can be mitigated in many ways, the most important is taking care of the Arabic content on Wikidata by filling in the missing information about Arabic named entities such as descriptions and aliases. In addition, to reduce the irrelevant data for nodes with the same surface forms as much as possible, the entity type (NER

Tag) of the named entity can also be matched with Wikidata types(instance of :P21 in Wikidata) when querying the named entity. However, this process needs to match the NER tags with the Wikidata types if any.

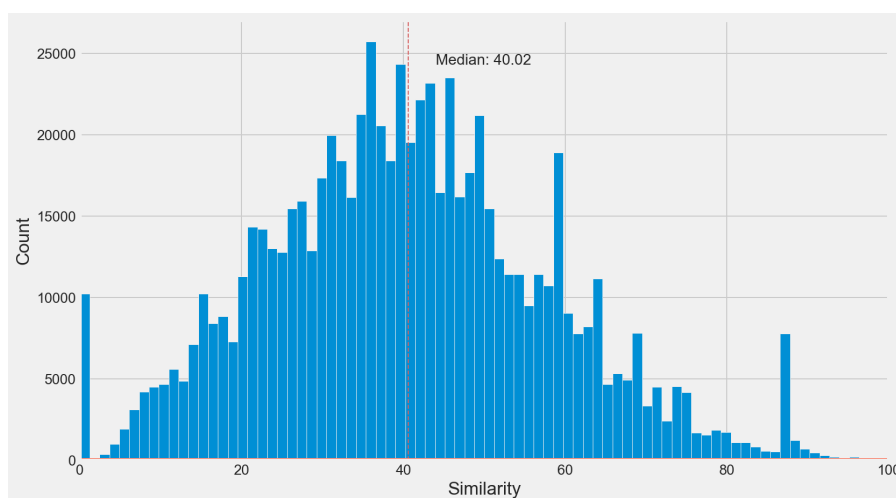


FIGURE 6.3: Cosine-similarity between contexts and glosses

The second reason is the contexts that were extracted from Wikipedia and used to train the models. Recall that the contexts were taken from the first sentence of Wikipedia. However, this sentence is similar to the Wikidata description of the named entity (both define the named entity).

The histogram chart 6.3 depicts the counts of the pairs based on cosine-similarity between the contexts and the glosses. It is clear from the graph that the similarity between the context and glosses is high with a median of 40% between pairs. To sum up, we could say that generally the data used to train the models is over-fitted, as the first sentence from Wikipedia mostly forms a definition of the named entity as said earlier, which is similar to Wikidata’s description.

Because of the over-fitting problem in the datasets, the NED component was separately evaluated on an external dataset that is taken from the Wojood-NED corpus to measure

the real accuracy.

### NED Component Disambiguation Evaluation

This section, the NED component was evaluated against pairs taken from Wojood-NED corpus. To evaluate the NED component alone. The candidate lookup component was evaluated against only named entities with have correct matches. The NED model was evaluated against (641) pairs on Wikidata glosses and (638) pairs on Wikipedia because not every Wikidata node has a corresponding Wikipedia node. The results per class in Table 6.9 came as follows:

TABLE 6.9: Achieved results(%) per class after evaluating the fine-tuned models on the Wojood-NED test corpus using Wikidata description as a gloss

Entity Class	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$	$M_{10}$	$M_{11}$
APPLICATION (20)	<b>82%</b> <b>(18)</b>	41% (9)	32% (7)	55% (12)	<b>82%</b> <b>(18)</b>	36% (8)	36% (8)	36% (8)	41% (9)	59% (13)	50% (11)
FRAMEWORK (98)	82% (80)	76% (74)	69% (68)	81% (79)	67% (66)	61% (60)	51% (50)	86% (78)	<b>88%</b> <b>(86)</b>	68% (67)	51% (50)
PL (280)	83% (232)	65% (183)	55% (154)	64% (178)	<b>84%</b> <b>(235)</b>	66% (185)	56% (158)	58% (163)	73% (203)	65% (181)	68% (191)
PLATFORM (21)	57% (12)	33% (7)	38% (8)	<b>76%</b> <b>(16)</b>	29% (6)	19% (4)	10% (2)	33% (7)	72% (11)	52% (11)	52% (11)
PP (6)	<b>67%</b> <b>(4)</b>	50% (3)	17% (1)	17% (1)	33% (2)	17% (1)	0% (0)	50% (3)	50% (3)	17% (1)	17% (1)
SD (51)	<b>82%</b> <b>(41)</b>	22% (11)	24% (12)	71% (36)	25% (13)	10% (5)	6% (3)	53% (27)	63% (32)	69% (35)	67% (34)
WEBSITE (165)	59% (97)	39% (65)	39% (64)	56% (82)	53% (87)	38% (62)	41% (67)	45% (75)	49% (81)	<b>61%</b> <b>(100)</b>	44% (73)
<b>Overall Accuracy:</b> <b>Total count: 641</b>	<b>76%</b> <b>(484)</b>	55% (352)	49% (314)	63% (404)	67% (427)	51% (325)	45% (288)	56% (361)	66% (425)	64% (408)	52% (335)

- $M_1$  outperformed the rest of the models in six classes (PP, and SD) with accuracy ( 67%, and 82%) respectively. While it achieved the highest accuracy with a tie with  $M_5$  in the class APPLICATION, with an accuracy of 82%.  $M_1$  also outperformed the rest of the models with an overall accuracy of 76% . $M_5$  ranked third with an overall accuracy of 67%.
- $M_4$  achieved the highest accuracy in the (PLATFORM) class with (76%). The model ranked fifth with overall accuracy of 63%.

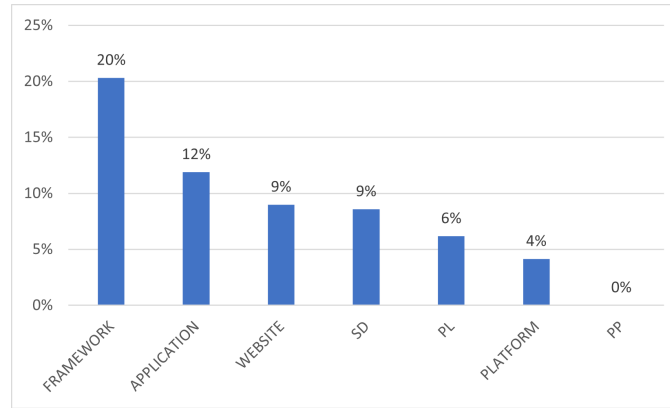


FIGURE 6.4: The accuracy change comparison per class for the  $M_5$  model results (Table 6.8 vs Table 6.10)

- $M_5$  outperformed the rest of the fine-tuned models in the (PL) class, as it achieved (84%) respectively.
- $M_9$  achieved the highest accuracy in the (FRAMEWORK) class with accuracy of (88%), and ranked third with an overall accuracy of 66%.
- $M_{10}$  outperformed the rest of the models in the WEBSITE classes with 61% accuracy. The model ranked fourth with an overall accuracy of 64%.
- By investigating the results achieved by  $M_5$  in the overall components linking accuracy (Table 6.8) vs the NED component linking accuracy (Table 6.10). It is clear that the accuracy increased by 10% when evaluating the NED component alone. The bar chart 6.4 depicts the overall accuracy change per class between the two experiments.

If we look in depth at the results of the FRAMEWORK class that achieved the highest percentage of change in accuracy (20%). The majority of the wrong matched nodes was due to the end user typos. Furthermore, the named entity SEO(Q180711), which is the most repetitive entity was linked to Yeast SEO(Q68342360) several times, which is another node on Wikidata.

TABLE 6.10: Achieved results(%) per class after evaluating the fine-tuned models on the Wojood-NED test corpus using Wikipedia description as a gloss

Entity Class	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$	$M_{10}$	$M_{11}$
APPLICATION (22)	68% (15)	55% (12)	59% (13)	27% (6)	<b>77%</b> <b>(17)</b>	73% (16)	73% (16)	50% (11)	45% (10)	50% (11)	32% (7)
FRAMEWORK (90)	86% (77)	<b>89%</b> <b>(80)</b>	78% (70)	79% (71)	83% (75)	60% (54)	51% (46)	80% (72)	86% (77)	50% (45)	63% (57)
PL (279)	71% (197)	61% (170)	65% (180)	68% (189)	<b>89%</b> <b>(284)</b>	75% (209)	45% (126)	65% (182)	65% (157)	53% (149)	69% (192)
PLATFORM (21)	<b>76%</b> <b>(16)</b>	57% (12)	52% (11)	24% (5)	48% (10)	19% (4)	19% (4)	43% (9)	29% (6)	67% (14)	71% (15)
PP (6)	33% (2)	67% (4)	33% (2)	0% (0)	50% (3)	17% (1)	17% (1)	<b>83%</b> <b>(5)</b>	50% (3)	17% (1)	17% (1)
SD (51)	29% (15)	27% (14)	29% (15)	14% (7)	<b>96%</b> <b>(49)</b>	73% (37)	84% (43)	45% (23)	24% (12)	10% (5)	51% (26)
WEBSITE (169)	<b>60%</b> <b>(101)</b>	51% (87)	<b>60%</b> <b>(102)</b>	43% (73)	58% (98)	36% (60)	28% (48)	44% (75)	42% (71)	40% (67)	53% (89)
<b>Overall Accuracy:</b>	66%↓	59%↑	62%↑	55%↓	<b>84%</b> ↑	60%↑	45%	59%↑	53%↓	46%↓	61%↑
<b>Total count: 638</b>	(423)	(379)	(393)	(351)	<b>(536)</b>	(381)	(284)	(377)	(336)	(292)	(387)

While the results per class in Table 6.10 came as follows:

- $M_1$  outperformed the rest of the models in one class (PLATFORM) with accuracy (76%). While it achieved the highest accuracy with a tie with the model  $M_3$  in the class WEBSITE, with an accuracy of (60%).  $M_1$  ranked second with an overall accuracy of 66%. While  $M_3$  ranked third with an overall accuracy of 62%.
- $M_2$  outperformed the rest of the fine-tuned models in the (FRAMEWORK) class, as it achieved (89%) accuracy. The model overall accuracy is 59%.
- $M_5$  outperformed the rest of the fine-tuned models in three classes (APPLICATION, PL, and SD), as it achieved (77%, 89%, and 96%) accuracy respectively. While it achieved the highest accuracy with a tie with ( $M_6$ ) in the class GPE with an accuracy of 56%. The model advanced to the first rank with overall accuracy of 84%.
- $M_8$  achieved the highest accuracy in the class (PP) with accuracy of (83%). The model achieved fifth rank with overall accuracy of 59%.

These two linking experiments contribute to the research question  $RQ_3$  which stated: **Given a named entity in a sentence (especially in software-related discussion forums), what is the best performing fine-tuned Arabic Bert model to link this entity with a node in Wikidata?**. The results in the two tables show that  $M_5$  outperformed the rest of the models with overall accuracy (84%) using Wikipedia glosses. This model was trained on the **D1 ArabGlossBERT(Al-Hajj & Jarrar, 2021) mixed with 23k True pairs from the WikiGlossContext**.

The possible reason for this model to outperform the rest models is that it is trained on ArabGlossBERT(Al-Hajj & Jarrar, 2021) mixed with 23k *True* pairs from the WikiGlossContext dataset. Besides, the Wikipedia descriptions is richer in information, more precise, and longer compared to Wikidata descriptions.

As a conclusion, we can find that combining WikiContextGloss with ArabGlossBERT (Al-Hajj & Jarrar, 2021) helps in achieving better results in Model  $M_5$ . And using Wikipedia's description as a gloss made noticeable effects on the linking performance of some models. This may be due to the fact that Wikipedia's summary is much longer, more descriptive, and richer in information than the Wikidata description. However, the results of the best performing model  $M_1$  slightly decrease when using Wikipedia instead of Wikidata description. As a result, the Wikidata description will be used as a gloss in the implementation of entity linking process, because it achieves a remarkable performance. Besides, not all Wikidata nodes are linked to Wikipedia.

## 6.5 Implementing Linguist APIs for Entity Linking

In this section, we demonstrate the implementation of the entity disambiguation and linking. This demonstration was implemented and deployed as part of WSD+NED framework, developed by SinaLab at Birzeit University. The WSD+NED takes a text as input, and returns all tokens in this text with their disambiguated nodes, as the following: (1) it parses the text and recognizes the named entities in the text using



the Wojoood BERT Model (Jarrar et al., 2022) if a token is not recognized as a named entity it is then passed to (2) ArabGlossBERT model (Al-Hajj & Jarrar, 2021) to disambiguate its linguistic meaning (i.e., link it with a linguistic concept in a lexicon). For those that are recognised as named entities, they are passed to the (3) candidate lookup component which intern returns all the possible candidate list of each recognized named entity and passed to (4) NED BERT model to link them to Wikidata node.

It is worth noting that the Wojoood BERT Model is, currently, able to recognize 21 classes, i.e., it does not support any of the six software-specific named entities yet. In the future, we will train a NER model to support the 27 named entities (the NER component was not implemented in this thesis. But, for evaluation purposes, we manually annotated the Wojoood-NED corpus with the new six software-specific tags). However, as mentioned earlier, the best performing model was evaluated on a software-specific named entities. Thus, it is guaranteed that the deployed model in the NED web service behaves well in linking software-specific named entities.

Figures (6.5, 6.6, and 6.7) are working examples for the entity linking using the online Demo (<https://ontology.birzeit.edu/tools/ALMA.html>):



FIGURE 6.5: Example 1: Qualcomm has introduced a new chipset for smartwatches.

أنشأ مارك زوكربيرج شركة فيسبوك

Tokenize Lemmatize NER Tagging WSD+NED WSD

◀ أنشأ (أنشأ 1\_303054281): أنشأ الشيء:- أقامه، أوجده وأحدثه "أنشأ مُجمَعًا تعليميًا/ مدرسة/ جريدة/ شركة - أنشأ نظامًا سياسيًا جديدًا".

◀ مارك زوكربيرج (مارك زوكربيرج\_Q36215): مارك زوكربيرج: إنسان, مبرمج امريكي والمؤسس المشارك لموقع التواصل الاجتماعي فيسبوك

◀ شركة فيسبوك (شركة فيسبوك\_Q355): فيسبوك: منظمة, موقع ويب, خدمة تواصل اجتماعي

FIGURE 6.6: Example 2: Mark Zuckerberg founded Facebook

أعلن موقع يوتيوب أنه بدأ في طرح دعم صورة داخل صورة

Tokenize Lemmatize NER Tagging WSD+NED WSD

◀ أعلن (أعلن 1\_303037486): أعلن الأمر/ أعلن بالأمر/ أعلن عن الأمر:- أظهره، صرّح به وجهر، عكسه أخفاه "أعلن موقفه/ رأيه/ الحزب - إعلان الإفلاس - حزينًا إنك تعلم ما نخفي وما نُعلن"> إبراهيم/ 38 - <عَمَّ إِنِّي أَعلنُ لَهُمْ وَأَسْرَرْتُ لَهُمْ إِسْرَارًا> نوح/ 9". أعلن العداوة : جاهر بها

◀ موقع يوتيوب (موقع يوتيوب\_Q866): يوتيوب: خدمة لمشاركة مقاطع الفيديو عبر الإنترنت

◀ أنه (أَنْ 1)

◀ بدأ (بدأ 1\_303002802): بدأ يفعل كذا:- أخذ، شرع "بدأ يكتب مذكراته - بدأت الأمور تتحسن".

FIGURE 6.7: Example 3: YouTube announced that it has started rolling out picture-in-picture support

## 6.6 Named Entity Linking in Software-related Text

In this section, we present a set of usage scenarios for software-specific named entities disambiguation and linking for the Arabic language. Our goal is this section to illustrate how the research conducted in this thesis contributes to answering  $RQ_{1a}$ :**(Given**

**a sentence about a software-related text (e.g., requirement, review, bug report), how understanding the semantics of this sentence improves understanding and classifying these issues?).**

The process of annotating named entities inside a free text is beneficial in semantic knowledge, and it has many applications such as improving the search engines' accuracy. It is important in recommender systems and chat-bots applications. The common in all these applications is that relevant text is distinguished from non-relevant.

For example, understanding the semantics of named entities can take searching to the next level by introducing the concept of searching for objects rather than searching for strings. Knowing the named entity inside the query and understanding its semantics yields a smarter search and more accurate and relevant result. Besides, linking entities with KGs can help in enriching the result of the query with important information about the named entity. The following example illustrates the idea. If a developer wants to search about "The developer of python", without linking the named entity Python with a KG, it is impossible for the search engine that searches the text inside documents to be able to directly retrieve relevant results that have the answer "Guido van Rossum", leading to false negatives. The following is a possible scenarios that can help answer the question.

**Scenario 1:** A company (X) owns a mobile application called (شاهد/Shahid) that is hosted in both the App-store and Play-store. This company has two development teams, one for each native platform. So, the company collects feed-backs from the App reviews posted on the mentioned platforms, in addition to reviews on its official website on the social networking websites (Facebook, Twitter) and store all the reviews on a database to be processed later. The user feedback is provided as free text that has many slangs and dialects. The company wants to filter and classify all the comments that contain the app name (شاهد/Shahid) but not the word (شاهد/Watch) or the

word (شاهد/Witness), or other different meanings if any. Besides, the company wants to know from the collected reviews, what reviews related to issues in the Android OS from those related to Apple OS, and wants to know if the corresponding version of the platform in case only the name mentioned in the text (e.g Marshmallow -> Android 6.0). In order to forward the bugs to each department automatically without wasting much time reading and analyzing each review.

Our entity linking tool can help the company X, by extracting such named entities from the free text and linking the node to Wikidata. For example, given the following app review (شاهد لا يعمل على دونت/Shahid doesn't work on Donut), our tool can extract the named entity types by tagging (شاهد/Shahid) as an APPLICATION and distinguishes it from the other words with meanings "Watch" or "Witness", and the word (دونت/Donut) as PLATFORM, which is different than the word from which means dessert food "Donut", by linking it to the Wikidata node [Q20741039](#). By doing so, the company can easily know what version number the Android Donut (Android 1.6) even though it is not mentioned in the app review and can help to know to which platform the Donut belongs from the information found on the Wikidata node (Property:P348 software version identifier, and P31 instance of).

**Scenario 2:** The ministry of education wants to adopt an Electronic Archiving system to replace paperwork. It prepares an SRS document for concerned software companies. The document contains free text to describe exactly how the Archiving system is supposed to work and the technology stack to use (functional and non-functional requirements).

Candidate companies are seeking a smart tool to (1) extract software-specific named entities, and (2) classify functional requirement based on the predefined user roles (e.g. System admin, Auditor, Archive user, etc). The smart tool should search only for relevant text, and provides detailed information for the development team about every

software-specific term.

For example, the sentence (حيث يتم أرشفة الكتب الصادرة والواردة بشكل يومي بدون حركات سير العمل بين) the company wants to extract technical terms from the sentence above and get more information about each software-specific term (i.e. named entity). From the functional requirement above, the tool should distinguish the world خادم/Server from خادم/Servant and link it to the node Q44127, so the later should be discard from the results. Similarly, the named entity **pdf** will be classified as a software standard and linked to the node Q42332. Moreover, the tool should list information about **pdf** (e.g. MIME Type, available versions, official website, stack-overflow tag, etc.)

The company can easily extract and classify software-specific named entities such as programming languages, software standards, and user roles) and then classify functional requirements based on the user roles, without the need to manually do that using our presented tool.

For example, The Functional requirement (ألوان النظام يجب أن تكون متجانسة مع بوابة الخدمات) Our tool can extract job titles because they are named entities, such as (الإلكترونية ومريحة للنظر ويمكن لمدير النظام تعديل الألوان والخطوط والشكل العام للنظام). (مدير النظام/System administrator) from the text and map them to the user roles, and then return all functional requirements related to this role ("system admin").

## 6.7 Summary

In this chapter, the entity disambiguation problem is defined in section 6.2, the experimental setup, experimental hyperparameters, and the experiments' tracking were addressed in section 6.3. While the results of the experiment were reported and discussed in section 6.4. All the fine-tuned models were subjected to disambiguation evaluation to select the best-performing model for the NED component. Moreover, the implementation and demonstration of the entity linking tool is presented in section 6.5.

Finally, in section 6.6, a couple of illustrative scenarios were presented for the entity linking tool usage in the software-engineering domain.

## Chapter 7

# Conclusion and Future Work

### 7.1 Introduction

In this chapter, work contributions have been summarized and concluded. In section 7.2, we conclude our work, and in section 7.3, challenges and limitations were explored and addressed for future work. Finally, threats to validity are provided in the last section 7.4.

### 7.2 Conclusion

In the previous chapters, we introduced the problem of Arabic named-entity disambiguation and linking. In the background chapter, we included the needed concepts and background knowledge for the terminologies embodied in this thesis. The entity linking tools and the research gap were discussed in the literature review chapter. This was done by performing a literature review to reveal the methodologies, limitations, and challenges faced by the research community. After that, we presented an entity linking components for Arabic named entity disambiguation. This includes data collection steps and annotated corpus building necessary to train the models.

In this research, we constructed a dataset for Arabic software-specific named entities called **Wojood-NED** contains (2,650 sentences, with 50,449 tokens). This dataset

was originally taken from WoJood corpus (Jarrar et al., 2022) that is capable of extracting 21 named entities, and enriched with an additional six software-specific named entities' and named entities are linked to the Wikidata nodes using the corresponding Q-identifier.

Wikidata is a large knowledge graph to which most of the nodes have been linked to Wikipedia using site-links. Although the data is stored in a structured way on Wikidata, there are some problems that have been observed during the process of preparing the data in this research, most of them are related to the quality and conformity of the data entered, especially in the "also known as" field that is used to add the synonyms of names aka aliases. This field contains false information that does not belong to the real name, for example (جائزة الكرة الذهبية الأوروبية / *European golden ball*) is an alias that was given to many named entities of type human (Footballer) such as (ول برايتنر / *Paul Breitner*), to mention a few. In addition, the field often contains erroneous symbols, formatting and mixed language rather than Arabic. It is known that this field is strongly related to the success of the process of search using the candidate lookup as well as the construction of the WikiGlossContext corpus. Recall that the WikiGlossContext corpus contains all Arabic named entities from Wikidata that are linked to Wikipedia. These named entities belongs to different named entity classes including software-specific classes. Hence, its is a domain-agnostic corpus that can be used to link any named entity type. This important, because deep-learning models are data hungry. Besides, this ensures the effectiveness of the model when new named entities are introduced. Therefore, there was a need to manually audit the data to avoid those problems that are difficult to detect automatically. Moreover, out of a total of 95 million records, there are only about one million records that have a complete Arabic information, such as label and description. It was noted that the Wikidata description field is short in general, and the description came with a general meaning mostly in a way that does not describe the field accurately. In general, the same description is used for a large group of named entities of same type (i.g football players are often



describes as *لاعب كرة قدم*/*Footballer*, humans who work in politics as *سياسي*/*Politician*). This, in turn, caused a deficiency in this important field, forcing us to expand the field by adding the name of the named entity as well as its type(s) to the description to lengthen and enrich it. One of the recommendations, in particular, is the necessity of paying attention to Arabic content and checking it on the basis of evidence on Wikidata. One of the recommendations, in particular, is the necessity of paying attention to Arabic content and checking it on the basis of evidence on Wikidata.

In a related context, with regard to the evidence extracted from Wikipedia. As a reminder, one descriptive sentence was extracted for each named entity. It was noted that this field is richer in information compared to the description field of the same-named entity on the Wikidata. In general, with regard to Wikipedia, the information on it is in the form of free text with a simple structured information box. Most of the information extracted from the free text is good information regarding quality, but it does not come without problems; this includes the presence of many languages mixed with Arabic as aliases.

We presented a huge corpus for pairs of Arabic named entities called **WikiGlossContext** that were carefully extracted from the Wikidata and Wikipedia. The *False* pairs were generated based on the *True* pairs with two different methods (cross-related and false-local). As a result, **WikiGlossContext**<sub>cross-related</sub> contains 1,106,408 (*True* and *False*) pairs and the **WikiGlossContext**<sub>false-local</sub> contains 1,142,280 (*True* and *False*) pairs.

The results show that the models trained on combined dataset ArabGloss (Al-Hajj & Jarrar, 2021) + 23 *True* pairs from WikiGlossContext corpus achieved good performance (82%) on linking unnamed entities (concepts). While they achieved good results (84%) when evaluating using the Wojood-NED corpus (which contains named entities). It is also clear from the results of the experiments that the method used

in generating false pairs has the greatest impact on the performance of the models. The results showed that the WikiGlossContext<sub>cross-related</sub> with *False* pairs generated by cross-relating pairs gave better results on the Wojood-NED evaluation benchmark with **76%** accuracy. Besides, using Wikidata descriptions in the lookup component has given different results from using Wikipedia descriptions. It positively affects some models while negatively affecting others in terms of the accuracy measured.

### 7.3 Future Work

There is still space for enhancing Arabic software-specific named entity disambiguation and linking. We will fine-tune another BERT for the NER task on the Wojood-NED and Wojood corpora. This fine-tuned model will be able to recognize new six software specific classes, along with 21 classes that Wojood is able to recognize. Moreover, we plan to enrich the software-specific named entities dataset by including more Arabic software entities such as software versions, file types, devices, functions, variables, methods, constants, etc.

For the NED component, we plan to try different techniques to generate the *False* pairs and study the effect on the fine-tuned BERT models' performance. Moreover, we plan to try different dataset combinations. Most importantly, we plan to experiment more on whether to use (Wikidata or Wikipedia) descriptions as a glosses, by evaluating further models with different data combinations.

We also plan to extend our entity linking components to extract and link relations<sup>1</sup> from software-related texts with Wikidata. Moreover, we plan to study the effect of including the presented entity linking components in the classification of app reviews, bug reports, and requirements identification, and how this will improve the accuracy

---

<sup>1</sup>Relations examples (founded by, developed at), the full list can be found using the URL [https://www.wikidata.org/wiki/Wikidata:List\\_of\\_properties](https://www.wikidata.org/wiki/Wikidata:List_of_properties)

of classifying such software-related content.

Finlay, we will include smarter techniques, using machine learning, in the candidate lookup component by adopting vector-based semantic search by integrating the Elasticsearch framework with the current state-of-the-art on NLP for semantic representation of language (BERT). This will help in reducing the errors propagating through the stages of the presented entity linking components, and make query search smarter by understanding the context of the query. Besides, we manage to expand the local store by adding more nodes from Wikidata by localizing non-Arabic nodes.

## 7.4 Threats to Validity

**Internal validity :** The Wojood-NED corpus was labeled and linked manually with six software-specific tags and links to Wikidata using the Wikidata Q-identifier. The threats to internal validity in this process include human factors in determining the correct type for the named entity and a wrong Q-identifier. To mitigate this error, the labeling process was reviewed and validated by NER experts.

Unlike the Wojood-NED corpus, the WikiGlossContext corpus pairs were constructed automatically. The threats to internal validity in this process includes code errors during the *True* and *False* pairs generation. To mitigate this risk, the code was carefully tested during the development process. Besides, the data was validated and audited manually, and double-checked by two persons before eliminating improper data.

**External Validity :** The datasets used in this thesis were carefully constructed using a set of heuristics that are clear and strict, they can be easily followed and repeated. Moreover, the environmental setup and hyperparameters are well defined. Hence, other researchers can use the same results to fine-tune BERT to obtain the same results using the datasets define in Table 4.10. The code was written in a manner that makes it easy for other researchers to replicate the study. Therefore, the generalization of the thesis

results is considerable. Moreover, other researchers can validate the model results on the test set, since the final fine-tuned models generated from each experiment and the different test sets used are uploaded on the Huggingface website.

# Bibliography

- Abdelali, A., Hassan, S., Mubarak, H., Darwish, K., & Samih, Y. (2021). Pre-training BERT on arabic tweets: Practical considerations. *CoRR*, *abs/2102.10684*. <https://arxiv.org/abs/2102.10684>
- Abdul-Mageed, M., Elmadany, A., & Nagoudi, E. M. B. (2021). ARBERT & MARBERT: Deep bidirectional transformers for Arabic. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 7088–7105. <https://doi.org/10.18653/v1/2021.acl-long.551>
- Alam, M., Buscaldi, D., Cochez, M., Osborne, F., Recupero, D. R., Sack, H., Sevgili, Ö., Shelmanov, A., Arkhipov, M., Panchenko, A., Biemann, C., Alam, M., Buscaldi, D., Cochez, M., Osborne, F., Refogiato Recupero, D., & Sack, H. (2022). Neural entity linking: A survey of models based on deep learning. *Semant. Web*, *13*(3), 527–570. <https://doi.org/10.3233/SW-222986>
- Al-Hajj, M., & Jarrar, M. (2021). Arabglossbert: Fine-tuning bert on context-gloss pairs for wsd. *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, 40–48. [https://doi.org/10.26615/978-954-452-072-4\\_005](https://doi.org/10.26615/978-954-452-072-4_005)
- Alian, M., Awajan, A., & Al-Kouz, A. (2016). Word sense disambiguation for arabic text using wikipedia and vector space model. *Int. J. Speech Technol.*, *19*(4), 857–867. <https://doi.org/10.1007/s10772-016-9376-y>
- Alotaibi, F., & Lee, M. (2014). A hybrid approach to features representation for fine-grained Arabic named entity recognition. *Proceedings of COLING 2014, the*

- 25th International Conference on Computational Linguistics: Technical Papers*, 984–995. <https://aclanthology.org/C14-1093>
- Al-Qawasmeh, O., Al-Smadi, M., & Fraihat, N. (2016). Arabic named entity disambiguation using linked open data. *2016 7th International Conference on Information and Communication Systems (ICICS)*, 333–338
- The authors of this paper proposed a methodology to solve the Arabic named-entities disambiguation on DBpedia for three types of entities (person, location and organization) using query label expansion and text similarity techniques. A reference data-set of over 10K annotated entities based on Wikipedia and/or DBpedia was prepared. The proposed method is an enhancement over the a hybrid approach made by Al-Smadi et al., 2015.
- Al-Smadi, M., Al-Dalabih, I., Jararweh, Y., & Juola, P. (2019). Leveraging linked open data to automatically answer arabic questions. *IEEE Access*, 7, 177122–177136.
- Al-Smadi, M., Talafha, B., Qawasmeh, O., Alandoli, M. N., Hussien, W. A., & Guetl, C. (2015). A hybrid approach for arabic named entity disambiguation. *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business*, 1–4.
- Antoun, W., Baly, F., & Hajj, H. (2020). AraBERT: Transformer-based model for Arabic language understanding. *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, 9–15. <https://aclanthology.org/2020.osact-1.2>
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, & P. Cudré-Mauroux (Eds.), *The semantic web* (pp. 722–735). Springer Berlin Heidelberg.
- Banerjee, D., Chaudhuri, D., Dubey, M., & Lehmann, J. (2020). Pnel: Pointer network based end-to-end entity linking over knowledge graphs. *The Semantic Web – ISWC 2020: 19th International Semantic Web Conference, Athens, Greece,*

- November 2–6, 2020, *Proceedings, Part I*, 21–38. [https://doi.org/10.1007/978-3-030-62419-4\\_2](https://doi.org/10.1007/978-3-030-62419-4_2)
- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null), 1137–1155.
- Berry, D. M., Science, P. D. C., Krieger, M. M., & Mathematics, P. D. (2000). From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0.
- Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1247–1250. <https://doi.org/10.1145/1376616.1376746>
- Botha, J. A., Shan, Z., & Gillick, D. (2020). Entity Linking in 100 Languages. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 7833–7845. <https://doi.org/10.18653/v1/2020.emnlp-main.630>
- Bouziane, A., Bouchiha, D., & Doumi, N. (2020). Annotating arabic texts with linked data. *2020 4th International Symposium on Informatics and its Applications (ISIA)*, 1–5. <https://doi.org/10.1109/ISIA51297.2020.9416543>
- Bouziane, A., Bouchiha, D., Rebhi, R., Lorenzini, G., Doumi, N., Menni, Y., & Ahmad, H. (2021). Arald: Arabic annotation using linked data. *Ingénierie des Systèmes d’Information*, 26(2).
- Bringmann, B., Nijssen, S., & Zimmermann, A. (2011). Pattern-based classification: A unifying perspective. <https://doi.org/10.48550/ARXIV.1111.6191>
- Cetoli, A., Akbari, M., Bragaglia, S., O’Harney, A. D., & Sloan, M. (2018). Named entity disambiguation using deep learning on graphs. *CoRR*, abs/1810.09164. <http://arxiv.org/abs/1810.09164>
- Chisholm, A., & Hachey, B. (2015). Entity disambiguation with web links. *Transactions of the Association for Computational Linguistics*, 3, 145–156. [https://doi.org/10.1162/tacl\\_a\\_00129](https://doi.org/10.1162/tacl_a_00129)

- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *CoRR*, *abs/1103.0398*. <http://arxiv.org/abs/1103.0398>
- Darwish, K., Habash, N., Abbas, M., Al-Khalifa, H., Al-Natsheh, H. T., Bouamor, H., Bouzoubaa, K., Cavalli-Sforza, V., El-Beltagy, S. R., El-Hajj, W., Jarrar, M., & Mubarak, H. (2021). A panoramic survey of natural language processing in the arab worlds. *Commun. ACM*, *64*(4), 72–81. <https://doi.org/10.1145/3447735>
- Delpuech, A. (2020). Opentapioca: Lightweight entity linking for wikidata. In L.-A. Kaffee, O. Tifrea-Marcuska, E. Simperl, & D. Vrandečić (Eds.), *Proceedings of the 1st wikidata workshop (wikidata 2020) co-located with 19th international semantic web conference (opub 2020), virtual conference, november 2-6, 2020*. CEUR-WS.org. <http://ceur-ws.org/Vol-2773/paper-02.pdf>
- Derczynski, L., Maynard, D., Rizzo, G., van Erp, M., Gorrell, G., Troncy, R., Petrak, J., & Bontcheva, K. (2014). Analysis of named entity recognition and linking for tweets. *CoRR*, *abs/1410.7182*. <http://arxiv.org/abs/1410.7182>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Ehrlinger, L., & Wöß, W. (2016). Towards a definition of knowledge graphs. In M. Martin, M. Cuquet, & E. Folmer (Eds.), *Joint proceedings of the posters and demos track of the 12th international conference on semantic systems - semantics2016 and the 1st international workshop on semantic change & evolving semantics (success'16) co-located with the 12th international conference on semantic systems (semantics 2016), leipzig, germany, september 12-15, 2016*. CEUR-WS.org. <http://ceur-ws.org/Vol-1695/paper4.pdf>
- Elazhary, H. (2016). Avoiding ambiguities in arabic software user requirements. *International Journal of Software Engineering and Its Applications*, *10*(6), 141–160.
- El-khair, I. A. (2016). 1.5 billion words arabic corpus. <https://doi.org/10.48550/ARXIV.1611.04033>



- Elnaggar, A., Otto, R., & Matthes, F. (2018). Deep learning for named-entity linking with transfer learning for legal documents. *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference*, 23–28. <https://doi.org/10.1145/3299819.3299846>
- El-Razzaz, M., Fakhr, M. W., & Maghraby, F. A. (2021). Arabic gloss wsd using bert. *Applied Sciences*, 11(6). <https://doi.org/10.3390/app11062567>
- Elsahar, H., Vougiouklis, P., Remaci, A., Gravier, C., Hare, J., Laforest, F., & Simperl, E. (2018). T-rex: A large scale alignment of natural language with knowledge base triples. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.
- Esmeir, S. (2021). SERAG: Semantic entity retrieval from Arabic knowledge graphs. *Proceedings of the Sixth Arabic Natural Language Processing Workshop*, 219–225. <https://aclanthology.org/2021.wanlp-1.24>
- Evans, Z. (2022). Knowledge about knowledge graphs. Retrieved January 7, 2022, from <https://community.atlassian.com/t5/Confluence-questions/Knowledge-graph/qaq-p/1565284>
- Fürnkranz, J. (2013). Rule-based methods. In W. Dubitzky, O. Wolkenhauer, K.-H. Cho, & H. Yokota (Eds.), *Encyclopedia of systems biology* (pp. 1883–1888). Springer New York. [https://doi.org/10.1007/978-1-4419-9863-7\\_610](https://doi.org/10.1007/978-1-4419-9863-7_610)
- Gad-Elrab, M. H., Yosef, M. A., & Weikum, G. (2015). Named entity disambiguation for resource-poor languages. *Proceedings of the Eighth Workshop on Exploiting Semantic Annotations in Information Retrieval*, 29–34.
- Ganea, O.-E., & Hofmann, T. (2017). Deep joint entity disambiguation with local neural attention. *CoRR*, abs/1704.04920. <http://arxiv.org/abs/1704.04920>
- Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., & Zettlemoyer, L. (2018). AllenNLP: A deep semantic natural language processing platform. *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, 1–6. <https://doi.org/10.18653/v1/W18-2501>

- Gasmi, H., Bouras, A., & Laval, J. (2018). Lstm recurrent neural networks for cyber-security named entity recognition. *ICSEA*, 11, 2018.
- Gravili, G., Benvenuto, M., Avram, A., & Viola, C. (2018). The influence of the digital divide on big data generation within supply chain management. *The International Journal of Logistics Management*. <https://doi.org/10.3233/SW-212865>
- Group, W. (2022). <https://wikidata-todo.toolforge.org/stats.php?fbclid=iwar2wnkjhk6efel69bzuqjaxutcjueal6wvvcuvjvlyypcbgl0q0> [(Accessed on 08/31/2022)].
- Hachey, B., Radford, W., Nothman, J., Honnibal, M., & Curran, J. R. (2013). Evaluating entity linking with wikipedia [Artificial Intelligence, Wikipedia and Semi-Structured Resources]. *Artificial Intelligence*, 194, 130–150. <https://doi.org/10.1016/j.artint.2012.04.005>
- Hadni, M., Ouatik, S. E. A., & Lachkar, A. (2016). Word sense disambiguation for arabic text categorization. *Int. Arab J. Inf. Technol.*, 13(1A), 215–222.
- Haff, K. E., Jarrar, M., Hammouda, T., & Zaraket, F. (2022). Curras + baladi: Towards a levantine corpus. *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2022)*. <https://arxiv.org/pdf/2205.09692.pdf>
- Harandizadeh, B., & Singh, S. (2020). Tweeki: Linking named entities on Twitter to a knowledge graph. *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020)*, 222–231. <https://doi.org/10.18653/v1/2020.wnut-1.29>
- Hoffart, J., Milchevski, D., & Weikum, G. (2014). Stics: Searching with strings, things, and cats. *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 1247–1248. <https://doi.org/10.1145/2600428.2611177>
- Hoffart, J., Yosef, M. A., Bordino, I., Fürstenauf, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., & Weikum, G. (2011). Robust disambiguation of named entities in text. *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 782–792.
- Huang, B., Wang, H., Wang, T., Liu, Y., & Liu, Y. (2020). Entity Linking for Short Text Using Structured Knowledge Graph via Multi-Grained Text Matching.

- Proc. Interspeech 2020*, 4178–4182. <https://doi.org/10.21437/Interspeech.2020-1934>
- Inoue, G., Alhafni, B., Baimukan, N., Bouamor, H., & Habash, N. (2021). The interplay of variant, size, and task type in Arabic pre-trained language models. *Proceedings of the Sixth Arabic Natural Language Processing Workshop*.
- James, P. (1991). *Knowledge graphs*. University of Twente, Faculty of Applied Mathematics.
- Jarrar, M. (2005). *Towards methodological principles for ontology engineering* (Doctoral dissertation). Vrije Universiteit Brussel. Brussels, Belgium.
- Jarrar, M. (2011). Building a formal arabic ontology (invited paper). <http://www.jarrar.info/publications/J11.pdf>
- Jarrar, M. (2020). Digitization of arabic lexicons. *Arabic language status report* (pp. 214–217). UAE Ministry of Culture; Youth.
- Jarrar, M. (2021). The arabic ontology - an arabic wordnet with ontologically clean content. *Applied Ontology Journal*, 16(1), 1–26. <https://doi.org/10.3233/AO-200241>
- Jarrar, M., & Amayreh, H. (2019a). An arabic-multilingual database with a lexicographic search engine. *The 24th International Conference on Applications of Natural Language to Information Systems (NLDB 2019)*, 11608, 234–246. [https://doi.org/10.1007/978-3-030-23281-8\\_19](https://doi.org/10.1007/978-3-030-23281-8_19)
- Jarrar, M., & Amayreh, H. (2019b). An arabic-multilingual database with a lexicographic search engine. *International Conference on Applications of Natural Language to Information Systems*, 234–246.
- Jarrar, M., Habash, N., Alrimawi, F., Akra, D., & Zalmout, N. (2017). Curras: An annotated corpus for the palestinian arabic dialect. *Journal Language Resources and Evaluation*, 51(3), Article 2-s2.0-85001544989, 745–775. <https://doi.org/10.1007/S10579-016-9370-7>
- Jarrar, M., Khalilia, M., & Ghanem, S. (2022). Wojood: Nested arabic named entity corpus and recognition using bert. *Proceedings of the International Conference*

- on Language Resources and Evaluation (LREC 2022)*. <https://arxiv.org/pdf/2205.09651.pdf>
- Jarrar, M., & Meersman, R. (2008). Ontology engineering —the dogma approach. *Advances in web semantic i* (pp. 7–34). Springer. [https://doi.org/10.1007/978-3-540-89784-2\\_2](https://doi.org/10.1007/978-3-540-89784-2_2)
- Jarrar, M., Zaraket, F., Asia, R., & Amayreh, H. (2018). Diacritic-based matching of arabic words. *ACM Asian and Low-Resource Language Information Processing*, 18(2), 10:1–10:21. <https://doi.org/10.1145/3242177>
- Jia, Y., Qi, Y., Shang, H., Jiang, R., & Li, A. (2018). A practical approach to constructing a knowledge graph for cybersecurity. *Engineering*, 4(1), 53–60.
- Kannan Ravi, M. P., Singh, K., Mulang, I. O., Shekarpour, S., Hoffart, J., & Lehmann, J. (2021). CHOLAN: A modular approach for neural entity linking on Wikipedia and Wikidata. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, 504–514. <https://doi.org/10.18653/v1/2021.eacl-main.40>
- Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (tech. rep.). Technical report, EBSE Technical Report EBSE-2007-01.
- Kolitsas, N., Ganea, O.-E., & Hofmann, T. (2018). End-to-end neural entity linking. *CoRR*, *abs/1808.07699*. <http://arxiv.org/abs/1808.07699>
- Kuc, R., & Rogozinski, M. (2013). *Elasticsearch server*. Packt Publishing, Limited. <https://books.google.ps/books?id=PEFK3MuwBsIC>
- Li, N., Zheng, L., Wang, Y., & Wang, B. (2019). Feature-specific named entity recognition in software development social content. *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 175–182. <https://doi.org/10.1109/SmartIoT.2019.00035>

- Lison, P., & Tiedemann, J. (2016). OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 923–929. <https://aclanthology.org/L16-1147>
- Logeswaran, L., Chang, M.-W., Lee, K., Toutanova, K., Devlin, J., & Lee, H. (2019). Zero-shot entity linking by reading entity descriptions. *CoRR*, *abs/1906.07348*. <http://arxiv.org/abs/1906.07348>
- Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016). On the automatic classification of app reviews. *Requirements Engineering*, *21*(3), 311–331.
- Maalej, W., & Nabil, H. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, 116–125. <https://doi.org/10.1109/RE.2015.7320414>
- Makris, C., Pispirigos, G., & Simos, M. A. (2020). Text semantic annotation: A distributed methodology based on community coherence. *Algorithms*, *13*(7), 160.
- Malik, G., Çevik, M., Khedr, Y., Parikh, D., & Başar, A. (2021). Named entity recognition on software requirements specification documents. *Proceedings of the Canadian Conference on Artificial Intelligence*. <https://doi.org/10.21428/594757db.507e7951>
- Malyshev, S., Krötzsch, M., González, L., Gonsior, J., & Bielefeldt, A. (2018a). Getting the most out of wikidata: Semantic technology usage in wikipedia’s knowledge graph. In D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, & E. Simperl (Eds.), *The semantic web – iswc 2018* (pp. 376–394). Springer International Publishing.
- Malyshev, S., Krötzsch, M., González, L., Gonsior, J., & Bielefeldt, A. (2018b). Getting the most out of wikidata: Semantic technology usage in wikipedia’s knowledge graph. In D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, & E. Simperl (Eds.), *The semantic web – iswc 2018* (pp. 376–394). Springer International Publishing.

- Marrero, M., Urbano, J., Sánchez-Cuadrado, S., Morato, J., & Gómez-Berbís, J. M. (2013). Named entity recognition: Fallacies, challenges and opportunities. *Computer Standards & Interfaces*, 35(5), 482–489.
- McCrae, J. P. (2018). Mapping WordNet instances to Wikipedia. *Proceedings of the 9th Global Wordnet Conference*, 61–68. <https://aclanthology.org/2018.gwc-1.8>
- McCrae, J. P., & Buitelaar, P. (2018). Linking datasets using semantic textual similarity. *Cybernetics and Information Technologies*, 18(1), 109–123. <https://doi.org/doi:10.2478/cait-2018-0010>
- McCrae, J. P., & Cillessen, D. (2021). Towards a linking between WordNet and Wikidata. *Proceedings of the 11th Global Wordnet Conference*, 252–257. <https://aclanthology.org/2021.gwc-1.29>
- Meij, E., Balog, K., & Odijk, D. (2014). Entity linking and retrieval for semantic search. *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, 683–684. <https://doi.org/10.1145/2556195.2556201>
- Mich, L., Franch, M., & Inverardi, P. N. (2004). Market research for requirements analysis using linguistic tools. *Requir. Eng.*, 9(1), 40–56. <https://doi.org/10.1007/s00766-003-0179-8>
- Möller, C., Lehmann, J., & Usbeck, R. (2021). Survey on english entity linking on wikidata. <https://doi.org/10.48550/ARXIV.2112.01989>
- Mulang', I. O., Singh, K., Prabhu, C., Nadgeri, A., Hoffart, J., & Lehmann, J. (2020). Evaluating the impact of knowledge graph context on entity disambiguation models. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2157–2160. <https://doi.org/10.1145/3340531.3412159>
- Mulang', I. O., Singh, K., Vyas, A., Shekarpour, S., Vidal, M.-E., Lehmann, J., & Auer, S. (2020). Encoding knowledge graph entity aliases in attentive neural network for wikidata entity linking. *Web Information Systems Engineering – WISE 2020: 21st International Conference, Amsterdam, The Netherlands, October 20–24,*

- 2020, *Proceedings, Part I*, 328–342. [https://doi.org/10.1007/978-3-030-62005-9\\_24](https://doi.org/10.1007/978-3-030-62005-9_24)
- Nikolaev, F., & Kotov, A. (2020). Joint word and entity embeddings for entity retrieval from a knowledge graph. In J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, & F. Martins (Eds.), *Advances in information retrieval* (pp. 141–155). Springer International Publishing.
- Okazaki, N. (2007). Crfsuite: A fast implementation of conditional random fields (crfs).
- Pellissier Tanon, T., Weikum, G., & Suchanek, F. (2020). Yago 4: A reason-able knowledge base. In A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, & M. Cochez (Eds.), *The semantic web* (pp. 583–596). Springer International Publishing.
- Perkins, D. (2020). Separating the signal from the noise: Predicting the correct entities in named-entity linking.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2227–2237. <https://doi.org/10.18653/v1/N18-1202>
- Phan, M. C., Sun, A., Tay, Y., Han, J., & Li, C. (2017). Neupl: Attention-based semantic matching and pair-linking for entity disambiguation. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 1667–1676.
- Qi, Y., Jiang, R., Jia, Y., Li, R., & Li, A. (2018). Association analysis algorithm based on knowledge graph for space-ground integrated network. *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, 222–226. <https://doi.org/10.1109/ICCT.2018.8600234>
- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

- Raiman, J., & Raiman, O. (2018). Deeptype: Multilingual entity linking by neural type system evolution. In S. A. McIlraith & K. Q. Weinberger (Eds.), *Proceedings of the thirty-second AAAI conference on artificial intelligence, (aaai-18), the 30th innovative applications of artificial intelligence (iaai-18), and the 8th AAAI symposium on educational advances in artificial intelligence (eaai-18), new orleans, louisiana, usa, february 2-7, 2018* (pp. 5406–5413). AAAI Press. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17148>
- Ramshaw, L. A., & Marcus, M. P. (1999). Text chunking using transformation-based learning. In S. Armstrong, K. Church, P. Isabelle, S. Manzi, E. Tzoukermann, & D. Yarowsky (Eds.), *Natural language processing using very large corpora* (pp. 157–176). Springer Netherlands. [https://doi.org/10.1007/978-94-017-2390-9\\_10](https://doi.org/10.1007/978-94-017-2390-9_10)
- Ramshaw, L., & Marcus, M. (1995). Text chunking using transformation-based learning. *Third Workshop on Very Large Corpora*. <https://aclanthology.org/W95-0107>
- Ratinov, L., Roth, D., Downey, D., & Anderson, M. (2011). Local and global algorithms for disambiguation to Wikipedia. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 1375–1384. <https://aclanthology.org/P11-1138>
- Rizzo, G., & Troncy, R. (2012). NERD: A framework for unifying named entity recognition and disambiguation extraction tools. *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 73–76. <https://aclanthology.org/E12-2015>
- Ryan, K. (1993). The role of natural language in requirements engineering. [1993] *Proceedings of the IEEE International Symposium on Requirements Engineering*, 240–242. <https://doi.org/10.1109/ISRE.1993.324852>
- Sakor, A., Onando Mulang', I., Singh, K., Shekarpour, S., Esther Vidal, M., Lehmann, J., & Auer, S. (2019). Old is gold: Linguistic driven approach for entity and relation linking of short text. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human*



- Language Technologies, Volume 1 (Long and Short Papers)*, 2336–2346. <https://doi.org/10.18653/v1/N19-1243>
- Sakor, A., Singh, K., Patel, A., & Vidal, M.-E. (2020). Falcon 2.0: An entity and relation linking tool over wikidata. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 3141–3148. <https://doi.org/10.1145/3340531.3412777>
- Sarawagi, S. (2008). Information extraction. *Foundations and Trends® in Databases*, 1(3), 261–377. <https://doi.org/10.1561/1900000003>
- Singhal, A. (2012). *Introducing the knowledge graph: Things, not strings* [2020-11-13]. <https://www.blog.google/products/search/introducing-knowledge-graph-things-not/>
- Sorokin, D., & Gurevych, I. (2018a). Mixing context granularities for improved entity linking on question answering data across entity categories. <https://doi.org/10.48550/ARXIV.1804.08460>
- Sorokin, D., & Gurevych, I. (2018b). Mixing context granularities for improved entity linking on question answering data across entity categories. *CoRR*, abs/1804.08460. <http://arxiv.org/abs/1804.08460>
- Spitkovsky, V. I., & Chang, A. X. (2012). A cross-lingual dictionary for English Wikipedia concepts. *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, 3168–3175. [http://www.lrec-conf.org/proceedings/lrec2012/pdf/266\\_Paper.pdf](http://www.lrec-conf.org/proceedings/lrec2012/pdf/266_Paper.pdf)
- Sri Nurdiati, S., & Hoede, C. (2008). *25 years development of knowledge graph theory: The results and the challenge*. University of Twente.
- Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). Yago: A core of semantic knowledge. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, & P. J. Shenoy (Eds.), *Proceedings of the 16th international conference on world wide web, WWW 2007, banff, alberta, canada, may 8-12, 2007* (pp. 697–706). ACM. <https://doi.org/10.1145/1242572.1242667>

- Tabassum, J., Maddela, M., Xu, W., & Ritter, A. (2020). Code and named entity recognition in stackoverflow. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 4913–4926. <https://doi.org/10.18653/v1/2020.acl-main.443>
- Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C. (2014). Bug characteristics in open source software. *Empirical Softw. Engg.*, 19(6), 1665–1705. <https://doi.org/10.1007/s10664-013-9258-8>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
- Vrandečić, D., & Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10), 78–85. <https://doi.org/10.1145/2629489>
- Weichselbraun, A., Kuntschik, P., & Braşoveanu, A. M. (2018). Mining and leveraging background knowledge for improving named entity linking. *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*. <https://doi.org/10.1145/3227609.3227670>
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., ... Dean, J. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, *abs/1609.08144*. <http://arxiv.org/abs/1609.08144>
- Ye, D., Xing, Z., Foo, C. Y., Ang, Z. Q., Li, J., & Kapre, N. (2016). Software-specific named entity recognition in software engineering social content. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 1, 90–101. <https://doi.org/10.1109/SANER.2016.10>

- Yosef, M. A., Spaniol, M., & Weikum, G. (2014). AIDArabic a named-entity disambiguation framework for Arabic text. *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)*, 187–195. <https://doi.org/10.3115/v1/W14-3626>
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13(3), 55–75. <https://doi.org/10.1109/MCI.2018.2840738>
- Zeroual, I., Goldhahn, D., Eckart, T., & Lakhouaja, A. (2019). OSIAN: Open source international Arabic news corpus - preparation and integration into the CLARIN-infrastructure. *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, 175–182. <https://doi.org/10.18653/v1/W19-4619>
- Zhao, R., & Mao, K. (2017). Fuzzy bag-of-words model for document representation. *IEEE transactions on fuzzy systems*, 26(2), 794–804.
- Zhou, C., Li, B., & Sun, X. (2020). Improving software bug-specific named entity recognition with deep neural network. *Journal of Systems and Software*, 165, 110572. <https://doi.org/https://doi.org/10.1016/j.jss.2020.110572>
- Zhou, C., Li, B., Sun, X., & Guo, H. (2018). Recognizing software bug-specific named entity in software bug repository. In F. Khomh, C. K. Roy, & J. Siegmund (Eds.), *Proceedings of the 26th conference on program comprehension, ICPC 2018, gothenburg, sweden, may 27-28, 2018* (pp. 108–119). ACM. <https://doi.org/10.1145/3196321.3196335>
- Zhou, L., Gao, J., Li, D., & Shum, H.-Y. (2020). The design and implementation of xiaoice, an empathetic social chatbot. *Computational Linguistics*, 46(1), 53–93.
- Zhu, Y., Kiros, R., Zemel, R. S., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 19–27. <https://doi.org/10.1109/ICCV.2015.11>

- 
- Zou, X. (2020). A survey on application of knowledge graph. *Journal of Physics: Conference Series*, 1487(1), 012016. <https://doi.org/10.1088/1742-6596/1487/1/012016>
- Zwicklbauer, S., Seifert, C., & Granitzer, M. (2016). Robust and collective entity disambiguation through semantic embeddings. *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 425–434.