A NOVEL FRAMEWORK FOR MORPHOLOGICAL PROCESSING OF TURKISH

OLGUN DURSUN

BOĞAZİÇİ UNIVERSITY

2023

A NOVEL FRAMEWORK FOR MORPHOLOGICAL PROCESSING OF TURKISH

Thesis submitted to the

Institute for Graduate Studies in Social Sciences

in partial fulfillment of the requirements for the degree of

Master of Arts

in

Cognitive Science

by

Olgun Dursun

Boğaziçi University

2023

A Novel Framework for Morphological Processing of Turkish

The thesis of Olgun Dursun

has been approved by:

Prof. Tunga Güngör
(Thesis Advisor) _____

Assist. Prof. Ümit Atlamaz
(Thesis Co-advisor) _____

Assist. Prof. Ömer Demirok _____

Assist. Prof. Suzan Üsküdarlı _____

Prof. Olcay Taner Yıldız
(External Member) _____

June 2023

DECLARATION OF ORIGINALITY

I, Olgun Dursun, certify that

- I am the sole author of this thesis and that I have fully acknowledged and documented in my thesis all sources of ideas and words, including digital resources, which have been produced or published by another person or institution;
- this thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- this is a true copy of the thesis approved by my advisor and thesis committee at Boğaziçi University, including final revisions required by them.

Signature…….....

Date …….....

ABSTRACT

A Novel Framework for Morphological Processing of Turkish

Morphological parsing is the computational task of breaking down words into their roots and affixes. There are several successful morphological parsers for Turkish, especially for inflectional morphology. However, there is a gap in the literature concerning the analysis of fusional properties of foreign-origin words, support for prefixes, and comprehensive derivational suffix coverage. To address this gap, this thesis describes and implements a new computational morphological processing framework for Turkish with novel principles. These principles are based on the recent opportunities and requirements in the natural language processing field, namely the transformer-based pre-trained large language models and fine-tuning approaches. The framework contains a description of language resources structure, a morphological analyzer that examines all possible parses of a word, a morphological disambiguator that picks the correct hypothesis among analyzer outputs, and error analysis modules for these tools.

ÖZET

Türkçe Morfolojinin İşlenmesi İçin Yeni Bir Çerçeve

Morfolojik, yani biçimbirimsel çözümleme, kelimelerin bilgisayarca kökleri ile eklerine ayrılması işidir. Türkçe için çeşitli çözümleyiciler vardır; bunlar başarılı bir şekilde, özellikle çekim eklerinin yapısını çözümleyebilirler. Fakat literatürde kimi yabancı kökenli kelimelerin bükümlü yapısının analizi, ön eklerin desteklenmesi, yapım eklerinin geniş bir şekilde kapsanması yönünden kimi eksiklikler vardır. Bu eksikliklere çözüm aramak için bu tezde Türkçe için yeni birtakım normlara dayanan bir hesaplamalı morfolojik işleme çerçevesi tanımlanıp uygulanmıştır. Bu ilkeler, doğal dil işleme alanındaki güncel olanaklar ile gereksinimlere dayanır. Bunların başında dönüştürücü (*transformer*) tabanlı, önceden eğitilmiş büyük dil modelleri ile ince ayarlama yaklaşımları gelir. Çerçeve, dil kaynakları yapısının açıklamasını, kelimelerin tüm olası çözümlemelerini inceleyen bir morfolojik analizciyi, analizci çıktıları arasından doğru hipotezi seçen bir morfolojik muğlaklık gidericiyi ve bu araçlar için hata analizi modüllerini içermektedir.

ACKNOWLEDGEMENTS

This has been quite a journey, and I am thankful for each step of it.

I wholeheartedly thank my advisors Tunga Güngör and Ümit Atlamaz for navigating me through the vast possibilities and several pitfalls along the way. I am deeply thankful to Pavel Logačev, who has fundamentally changed how I look at language. I also thank to the members of my thesis committee, Ömer Demirok, Suzan Üsküdarlı, and Olcay Taner Yıldız.

I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for their financial support through 2210-A National Scholarship Programme for Master's Students.

I thank for the patience of my friends and family who unwearyingly supported me as I took my time to complete this journey. We will need to catch-up.

My brother Onur has been by my side through everything.

Without Gülin, nothing would be possible.

CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Aim of the thesis 1

1.2 Turkish language 1

1.3 Definition of morphological parser 4

1.4 Two-level morphology 4

1.5 Existing Turkish morphological parsers 5

CHAPTER 2: TURKISH MORPHOLOGY 10

2.1 Definition of word components 10

2.2 Definition of morpheme 11

2.3 Morphemes in Turkish 12

CHAPTER 3: FRAMEWORK PRINCIPLES 15

3.1 Importance of derivational morphemes 15

3.2 Morphemes as subwords 15

3.3 Object structure of the framework 16

3.4 Leniency in analysis, standardization in generation 17

3.5 Data generation 17

3.6 User experience 18

3.7 Availability 18

CHAPTER 4: FRAMEWORK RESOURCES 19

4.1 Lexicon 19

4.2 Affix vocabulary 28

4.3 Constraint resources 33

CHAPTER 5: MORPHOLOGICAL ANALYZER 36

5.1 General structure of analyzer 36
5.2 Analysis pipeline 37
5.3 Analyzer performance 43
CHAPTER 6: MORPHOLOGICAL DISAMBIGUATOR 44
6.1 Previous methods 44
6.2 Fine-tuning transformer language models for disambiguation 45
6.3 Model evaluation 50
CHAPTER 7: DISCUSSION 52
7.1 Contributions of the framework 52
7.2 Limitations and future steps 53
CHAPTER 8: CONCLUSION 55
APPENDIX: SAMPLE ANALYZER OUTPUT 56
REFERENCES 57

LIST OF TABLES

Table 1. Generalized Representations and Variations 13

Table 2. Inflectional Suffixes of Turkish 13

Table 3. Derivational Suffixes of Turkish 14

Table 4. Sample Lexicon Entries 20

Table 5. Conversion Table for Arabic Letters 25

Table 6. Sample Affix Entries 28

Table 7. Model Accuracy with BOUN UD Treebank 50

Table 8. Model Accuracy with UD Penn Turkish Treebank 51

CHAPTER 1

INTRODUCTION

This chapter describes the aim of the thesis, the definition of the Turkish language and its core morphological phenomenon, as well as the current state of computational processing of morphology.

1.1 Aim of the thesis

The focus of this thesis is the computational processing of Turkish morphology. The examination of morphological elements of a word is called morphological parsing. As a morphologically rich language, Turkish attracted the attention of multiple natural language processing (NLP) scholars. There are several morphological parsers for Turkish, as well as some other morphologically rich languages like Finnish. However, as with any tool, previous Turkish parsers are based on some design choices, and those design choices are not fully aligned with the current NLP trends. This thesis aims to create a morphological processing framework for Turkish that includes a set of design principles, a resource structure, a morphological analyzer, a morphological disambiguator, a morphological tokenizer, and tools for error analysis.

1.2 Turkish language

Turkish is a Turkic language spoken by an estimated 88 million people, mainly in Turkey[1]. Turkish is officially regulated by Turkish Language Association (TDK). However, it is far from being an undisputed all-governing authority over the

[1] https://www.ethnologue.com/25/language/tur/

language. Former members of the TDK had founded *Dil Derneği* (Language Association) as an alternative authority in 1987, and several editors in Turkey often refer to Necmiye Alpay's *Türkçe Sorunları Kılavuzu* (Alpay, 2000) for dispute resolution.

There are several definitions of Turkish as a language, and some include other related languages like Azerbaijani, Uzbek, Kazakh, etc. as dialects of Turkish . However, "Turkic languages" is a more accurate definition as an umbrella term for related languages.

Our definition of Turkish as the Turkic language spoken in Turkey aligns with the primary uses of a language in computational contexts. However, not all 88 million estimated speakers speak the same dialect of Turkish. The catalog of TDK includes tens of dictionaries of local dialects. However, in this thesis, these local dialects are out of scope. The focus is on the Turkish that used to be called "Istanbul Turkish," then after the invention of telecommunication methods, "TRT Turkish," and now the Turkish that is used in corpora; therefore, mainly the variants used in books, newspapers, Wikipedia, etc.

Within the scope of this thesis, a profoundly descriptive position is adopted for what is accepted as and what is not Turkish. Turkish means everything that exists in a Turkish text corpus, and many grammatical variations are acceptable, aside from obvious and universally agreed-upon grammatical mistakes, such as misspelling of clitics *de, da* (separate from the word) as *-de*, *-da* (appended to the word) and vice versa. The rationale behind this choice is not from a theoretical point of view. It is because of the mass use of any available language data in any language for training large language models (LLMs) without significant filtering in current applications. A

framework that aims to morphologically parse and segment Turkish texts should ideally be able to handle as diverse requests as possible.

This choice requires leniency towards the use of foreign words, non-canonical uses, and other phenomena that will be discussed further.

Contemporary linguists agree on the agglutinative property of Turkish (Göksel & Kerslake, 2005). Haspelmath (2009) even criticizes his colleagues for confusing agglutination with being Turkish-like. Although there is a consensus around the agglutinative property of Turkish, it is incorrect to say that Turkish morphology only consists of agglutinative components and no other morphological phenomenon can be observed. Korkmaz (2009) notes that pre-republic works on Turkish (specifically Ottoman Turkish) grammar mainly consisted of Arabic-based *kavaid-i Osmaniyye* (Ottoman grammars) and some French-influenced works. She notes that since the beginning of the 20th century, Turkish has gotten rid of its so-called "flaws" that came with Arabic and Persian. Therefore, a method that does not follow French or Arabic classifications is followed in her work. This approach is not entirely incorrect, as contemporary Turkish is *mainly* agglutinative. However, Nişanyan (2022) argues that Arabic words constitute between 24% and 30% of the Turkish vocabulary depending on the lexicographic methodology, and Arabic morphology should be considered a part of contemporary Turkish grammar.

Whether speakers of Turkish process Arabic words as singular units or they reconstruct the meaning from the roots and morphological patterns (Ar. *wazn, awzan* Tur. *vezin*) is an open question worth exploring, where the answer possibly depends heavily on how familiar the speakers are with the Arabic grammar on its own. Still, especially from a computational perspective where the new trend is breaking words

down to their smallest meaningful units, parsing Arabic roots and patterns may have its benefits in other tasks.

1.3 Definition of morphological parser

Parsing is the operation of analyzing an input and breaking it down into meaningful components. Computational parsing can cover everything from the interpretation of coding conventions and data structures, such as JSON parsing, to natural language processing applications, such as syntax, dependency, and morphology parsing.

A morphological parser conventionally consists of two components: an analyzer that generates all potential parses of a given input word, and a disambiguator that takes the context and the output of the analyzer and decides which parse is the correct one given the context (Oflazer & Saraçlar 2018).

Some recent morphological parsers (Akyürek et al., 2019, Şahin & Atlamaz, 2022) do not follow this path and given an input with a sequence-to-sequence architecture, directly generate the correct parse. Sequence-to-sequence means that the model takes text as input and produces text as output, without following a step-by-step morphological analysis. In such approaches, the morphological parser is generally trained with neural networks. Therefore, we can define a morphological parser as a unit that takes an input and produces an output that contains the correct parse, with its internal structure either a sequence-to-sequence one or consisting of an analyzer and a disambiguator.

1.4 Two-level morphology

One of the most influential ideas in morphological parsing is the two-level morphology architecture suggested by Koskenniemi (1983).

Koskenniemi (1983) describes a lexical and a surface level for morphological processing, where the lexical level contains representations of morphemes that can take several surface forms, while the surface level contains the realized form of this representation.

He describes the difference between the lexical and surface level with the following inflection example on the Finnish noun *talo* (house):

Lexical level: t a l o n A

Surface level: t a l o n a

nA is the archiphonemic representation, which can take the forms *nä* or *na* depending on vowel harmony. Morphological features such as case, number, etc. are also a part of the lexical level. His two-level morphology approach is implemented through finite state transducers (FST) (Beesley & Karttunen, 2003), and these transducers are bidirectional, meaning that given a lexical input, a surface output can be produced, and vice versa.

There are several FST frameworks, starting from Koskenniemi (1983), whose framework later became PC-KIMMO. Foma (Hulden, 2009), HFST (Lindén et al., 2011), SFST (Schmid, 2006), OpenFST (Allauzen et al., 2007) are some of the contemporary FST compilers that can be, and are, used for Turkish morphological analysis.

1.5 Existing Turkish morphological parsers

Multiple Turkish morphological parsers exist in the literature, and some are currently available for use. This section describes some previous parsers based on availability, methodology, and ease of use.

1.5.1 Oflazer (1993)

Oflazer, (1993) is one of, if not the first, Turkish morphological analyzers. It is written for PC-KIMMO environment and its source files are accessible through a server hosted at Carnegie Mellon University[2]. Based on Oflazer & Saraçlar (2018), it is possibly succeeded by a Xerox Finite State Tools (Beesley & Karttunen, 2003) implementation, but this newer implementation is not openly available. Rules contain inflectional morphemes, as well as some derivational morphemes. However, the support for derivational morphemes is limited. It is inconvenient to use with modern hardware, and a virtualization layer is required to run PC-KIMMO.

Succeeding morphological parsers generally follow the conventions used in this parser. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: N(ev)+1PL-POSS+GEN

1.5.2 Sak et al. (2008)

Sak et al., (2008) consists of an FST-based analyzer and a perceptron-based disambiguator. Its rules are mainly based on Oflazer's description of Turkish morphology. It contains limited support for derivational morphemes.

Morphological analyzer and disambiguator are available through TULAP[3]. The morphological analyzer module is dependent on a x86 Linux shared object (.so). Therefore, running it requires a Linux machine running on a CPU with x86

[2] https://www.cs.cmu.edu/afs/cs/project/ai-repository 9/ai/util/areas/nlp/morph/pc_kimmo/turklex/

[3] Analyzer: http://hdl.handle.net/20.500.12913/4
Disambiguator: http://hdl.handle.net/20.500.12913/8

architecture or containerization/virtualization on another OS with an x86 CPU. With the recent re-popularization of ARM architecture, especially the adoption of these in newer Apple computers, this reliance on the shared object file becomes an inconvenience for users. Since the FST is in compiled form, this parser is not extensible.

Perceptron-based morphological disambiguator relies on a model file that is retrainable on a given dataset if necessary. This parser is used extensively in other works, such as generating the Turkish training data for Morpho Challenge iterations, getting baseline morphological features for BOUN UD Treebank (Marşan et al., 2022), and exploring input variations. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: ev[Noun]+[A3sg]+HmHz[P1pl]+NHn[Gen]

1.5.3 Zemberek NLP

Zemberek (Akın & Akın, 2007) is an open-source NLP toolkit that includes components for tasks such as morphology, tokenization, normalization, and named entity recognition. Its morphology module contains support for analysis, disambiguation, and generation. It does not rely on an FST backend but handles the analysis with a rule-based approach. It has some coverage of derivational suffixes and closely follows the behavior of an unknown version of Oflazer's analyzer[4]. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

[4] https://github.com/ahmetaa/zemberek-nlp/wiki/Morphology-Notes

Output: ev:Noun+A3sg+imiz:P1pl+in:Gen

1.5.4 Dilbaz

Dilbaz (Yıldız et al., 2019) is available in 6 programming languages, and in terms of dependencies, it can be considered self-contained in each of them. It does not rely on third-party FST tools, but rather uses a bespoke finite state machine logic. By using a trie data structure and an LRU cache, it can handle large corpora efficiently. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: ev+NOUN+A3SG+P1PL+GEN

1.5.5 TRmorph

TRmorph (Çöltekin, 2014) provides an FST morphological analyzer based on the foma backend. It covers a large number of morphological phenomena and has most of the suffixes from Göksel & Kerslake, (2005). However, the derivational suffixes that are deemed unproductive are commented out. Therefore, they are not compiled into the analyzer. Foma is a fast backend, and a morphological segmentation tool is provided along with the parser. All this makes TRmorph a good candidate to use for fast morphological tokenization of corpora. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: ev<N><p1p><gen><0><V><cpl:pres><3s>

1.5.6 Morse

Morse (Akyürek et al., 2019) is a sequence-to-sequence encoder-decoder-based morphological parser that can produce a single disambiguate output when given an input. It is written in Julia and requires some familiarity with this programming language to parse large amounts of text without performance loss. It follows the output convention of Oflazer's later parsers, as it is trained on a new dataset published along the parser, TrMorph2018, which is generated with Oflazer's parser and disambiguator from an updated 2018 version. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: ev Noun+A3sg+P1pl+Gen

1.5.7 TransMorpher

TransMorpher (Şahin & Atlamaz, 2022) is a sequence-to-sequence parser based on transformer architecture and has a component that handles phonological normalization, which converts the allomorphs into generalized forms. Although the reported accuracy is below Akyürek et al. (2019), within the confines of the system, the phonological normalization is reported to increase the accuracy of the sequence-to-sequence by 5-10% in tag and lemma accuracy. An example input and output for this parser for the Turkish word *evimizin* (of our house):

Input: evimizin

Output: ev+NOUN

+PersonNumber=A3sg+Possessive=P1pl+Case=Gen+Proper=False

CHAPTER 2

TURKISH MORPHOLOGY

This section gives an overview of the rules of Turkish morphology in the context of computational processing.

Although it could be desirable from the point of view of someone aiming to build an expert system to have a complete set of rules for a phenomenon to be modeled, it is not necessarily possible to write down all the morphological rules of Turkish and cover the whole language. There will always be exceptions, edge cases, non-canonical uses, new phenomena, outdated phenomena, etc. In the NLP field and other areas such as computer vision, the use of deep learning, in part, aims to avoid limitations of the rule-writing process.

In an ideal scenario, deep learning relies on high-quality data to train on. Turkish morphology is not a field with a lot of manually tagged data. Consequently, even if the aim is to finally migrate all workflows into a deep learning-based solution, analyzing the morphological rules and having a non-data-dependent analyzer is beneficial.

2.1 Definition of word components

Turkish has been written in Latin script since 1928, and words have been separated by whitespaces even in the Ottoman-Arabic script era before that.

We use Göksel & Kerslake's (2005) definition of root, stem, base, and word, which is described in Figure 1.



Figure 1. Visualization of root, stem, and suffixes

From a computational linguistics point of view, the difference between Turkish words is generally less diverse than in English. For example, the word *yapıt* (work, noun) can take the dative case marker and become *yapıta* (to the work), in which scenario, *yapıt* and *yapıta* are treated as entirely different words in a computational sense.

2.2 Definition of morpheme

Morphemes are, by principle, the smallest meaningful units in a language. They can be roots or affixes, meaning prefixes, infixes, circumfixes, and suffixes (Haspelmath & Sims, 2010). In Turkish, there are no productive circumfixes or infixes; mainly, there are suffixes.

Although the "smallest meaningful unit" definition looks elegant, it is not always possible to trace the "smallest" unit as a morpheme, and sometimes combined morphemes do not have their components visible. Several design choices are involved in defining suffixes and will be discussed further in the resources section.

As for roots, almost all the neologisms in the last 100 years and many older words have distinguishable roots. However, a rule-of-thumb distinction between an

etymological root and a morphological root should be drawn for ease of processing. As an example, both verbs *öğren* (learn) and *öğret* (teach) undoubtedly come from a hypothetical common root, possibly **ögür* (community, herd). As this root is not in active use today, in the scope of this thesis, *öğren* and *öğret* are treated as separate roots.

On the flip side, in the case of words like *açı* (angle) and *sayı*, although they appear as separate roots and probably are not processed by the speakers of Turkish as derivations since their roots can be traced back to *aç* (open) and *say* (count), they are treated as derivations of verbal roots with the suffix +H (realized as *ı, i, u, ü*) just like *çeviri, sayı, sömürü*.

Contrary to the official position and many grammar books, Turkish does have prefixes. How productive they are is an open question, but they are generally distinguishable from stems, as in the case of *atipik* (atypical) and *gayrinizamî* (irregular, anomalous).

2.3 Morphemes in Turkish

Within the constraints of this framework, Turkish morphology consists of 58 inflectional and 121 derivational suffix forms, as well as some prefixes. Table 1 contains a legend of generalized forms of morpheme sounds and their expanded variants, similar to the notion of archiphonemic representations of Koskenniemi (1983). Inflectional suffixes can be found in Table 2, and derivational suffixes in Table 3 (loosely based on Göksel & Kerslake, 2005; Korkmaz, 2009). Letters in parentheses denote optional thematic sounds.

Table 1. Generalized Representations and Variations

Representation	Variants
A	a, e
C	c, ç
H (high vowel)	ı, i, u, ü
K	k, ğ
D	d, t
G	g, k

Table 2. Inflectional Suffixes of Turkish

+(A/H)r	+(y)An	+DH	+m
+(H)m	+(y)Ayaz	+DHr	+mA
+(H)mHz	+(y)DH	+Hl	+mAktA
+(H)n	+(y)H	+Hm	+mAlH
+(H)nHz	+(y)Hm	+Hn	+mHş
+(H)yor	+(y)Hp	+Hr	+n
+(H)ş	+(y)Hver	+Ht	+nHz
+(n)DA	+(y)Hz	+k	+sA
+(n)Hn	+(y)lA	+kH(n)	+sHn
+(s)H(n)	+(y)mHş	+ki(n)	+sHnHz
+(y)A	+(y)sA	+lAr	+sHnlAr
+(y)Abil	+Ak	+lArH	+t
+(y)AcAK	+Ar	+lArH(n)	+z
+(y)Adur	+Art	+lArHn	
+(y)Akal	+DAn	+lHm	

Table 3. Derivational Suffixes of Turkish

+(A)C	+(H)z	+DA	+lAmA
+(A)CHK	+(H)ş	+DA(n)	+lAn
+(A)K	+(H)şDHr	+DAm	+lArH
+(A)cAn	+(h)ane	+DAn	+lAt
+(A)klA	+(t)en	+DH	+lAş
+(A)l	+(v)i	+DHK	+lH
+(A)lgA	+(y)A	+Daş	+lHk
+(A)m	+(y)AcAK	+Deş	+leyin
+(A)mAK	+(y)An	+GA	+mA
+(A)n	+(y)AsH	+GAC	+mAC
+(A)nAK	+(y)AsHcA	+GAn	+mAK
+(A)r	+(y)AsHyA	+GH	+mAcA
+(A)rH	+(y)HcH	+GHC	+mAdHK
+(A)t	+(y)Hm	+GHn	+mAn
+(A)v	+(y)Hn	+GHr	+mAz
+(A)y	+(y)Hş	+H	+mHK
+(A)ş	+(y)at	+Hm	+mHş
+(H)CHK	+(ş)Ar	+HnC	+rA
+(H)K	+A	+ane	+sA
+(H)k	+AgAn	+baz	+sAK
+(H)klA	+AlA	+cH	+sAl
+(H)lH	+AğAn	+dan	+sH
+(H)msA	+C	+dar	+sHl
+(H)msAr	+CA	+engiz	+sHz
+(H)msH	+CAK	+gil	+tH
+(H)mtraK	+CAnA	+istan	+tay
+(H)n	+CAsHnA	+iye	+vari
+(H)ncH	+CAğHz	+iyet	+zede
+(H)ntH	+CH	+kar	
+(H)r	+CHK	+lA	
+(H)t	+CHl	+lAm	

CHAPTER 3

FRAMEWORK PRINCIPLES

As a description of a novel morphological processing framework, this thesis is based on several principles and design choices. This chapter describes these principles.

3.1 Importance of derivational morphemes

A given derivational morpheme does not always add the same semantic feature to a given stem. Especially, language change and arbitrary use of derivational morphemes at the first stages of the language reform cause some words to have a more opaque composition. However, whether a word with one or more derivational morphemes is processed as a whole unit or a combination of root(s) and affix(es) is a question of areas such as psycholinguistics. The task of analyzing all possible derivational morphemes should not be discarded in favor of developmental convenience or challenges due to disambiguation. These challenges should be recognized, and answers should be pursued.

Full coverage of derivational morpheme segmentation is essential for historical linguistics and stylistics research.

3.2 Morphemes as subwords

Several studies explore the feasibility of using morphemes as subword units instead of frequency-based character-level methods like byte-pair encoding (Sennrich et al., 2016) or Wordpiece (Wu et al., 2016). One of the main drawbacks of using BPE for Turkish is its inability to recognize allomorphs. While language-specific morphological tokenization can merge plural markers *+ler* and *+lar* into a single

representation *+lAr*, current language-agnostic approaches do not recognize such patterns. The effect of this merge operation on various performance metrics is a subject of a separate study.

BPE and Wordpiece are deemed "good enough" for English, and multilingual models have the disadvantage of having to rely on a single tokenization method. However, morphology-based tokenization is gaining popularity for Turkish, and currently, it is limited to inflectional morphemes, accompanied by a very limited set of derivational morphemes in generalized forms. These applications rely on string manipulation operations based on the parser outputs, while some parsers do not offer word segmentations as default outputs. It is worth noting that this is not due to a technical limitation but a design choice.

A morphological framework should be able to give outputs in both surface form and representation form of segmented morphemes through parameters without having to tweak the source code.

3.3 Object structure of the framework

Although the principle of two-level morphology is not necessarily a limiting factor, having to define separate states for each exception adds up quickly in development. Instead, the OOP paradigm can be used for different features of words, affixes, and other components. A morphological processing framework should be able to produce outputs with only the relevant information for the user. While a more linguistics-oriented analysis may require what features each morpheme has. For example, automatically pre-tagging words in a Universal Dependencies Treebank (Nivre, 2020) only requires the ultimate morphological features of the overall word. Instead

of deleting parts of an output, some information can be excluded from the output to start with.

In line with this approach, language resources should be structured in a way that an object-oriented analysis module can handle them.

3.4 Leniency in analysis, standardization in generation

Finite state machines have the intrinsic purpose of deciding whether an input is valid or not. However, in line with the description of Turkish as anything that goes into a moderately controlled corpus, a morphological analyzer should handle the parsing of unconventional word formations. For example, **yapdık* may or may not be considered by the user as a valid form of *yaptık* (we made) in the analysis because it appears in a few news corpora. However, if this form is supported, the representational form "*yap +DHk"* must always be converted back into the canonically correct version *yaptık* as a normalization step.

3.5 Data generation

Data-driven methods prove to be better than rule-based approaches in many computer science fields. The upper bounds of data-driven learning of Turkish morphology are limited to the current capabilities of Sak et al. (2008) due to it being the basis of the Morpho-Challenge 2010 Turkish dataset and several iterations of TrMor datasets[5].

The foremost priority of a morphological parser should be the correct segmentation of any given input into their roots and affixes, and assigning correct morphological features, rather than checking if a given input adheres to specific

[5] 2006, 2016, and 2018 iterations can be found at https://github.com/ai-ku/TrMor2018

rules. This prioritization assumes that this framework's analyzer and disambiguator components will be used to generate high volumes of synthetic or manually verified morphology training data based on raw Turkish text. This training data can then be used for processing Turkish morphology with data-driven methods.

3.6 User experience

The learning curve to modify the resources to have an analysis module that fits specific needs should be low. For example, if a user wants to add a new suffix to the system, it should be as simple as adding some basic suffix properties to a resource file. An abstract understanding of the system should be enough, contrary to the need to understand all the conventions in an FST file.

3.7 Availability

A morphological processing framework should not depend on under-maintained third-party packages or other difficult-to-install dependencies. As the most popular programming language among NLP researchers at the date of writing, at least Python should be supported by the framework. In the case of Python, the installation of the whole framework should be as simple as "pip install package_name," and default resources should be downloaded from within the package, not externally.

The whole framework should work in all major-league consumer and business-grade operating systems and hardware without changing system behavior.

CHAPTER 4

FRAMEWORK RESOURCES

This section describes the linguistic resources used in the framework.

4.1 Lexicon

For any rule or FST-based morphological parser, lexicon coverage is a vital component. There are several methods to guess what the root or lexeme of a word is, and process which suffixes are added afterwards. However, there is always a correlation between the lexicon size and the real-life coverage of the parser.

In the lexicon of a conventional morphological analyzer, each lexical entry must at least have part of speech information to be processed further along the morphological states. The lexicon used in this work is a combination of unigram entries in the TDK dictionary and the morphological lexicons of Starlang and Çöltekin with detailed information to help parsing in various ways.

First and last names, foreign names, company names, and map data entities are also injected into the lexicon as proper nouns that cannot be broken down further.

The lexicon in this framework contains the following information for each entry: entry name, variants, type (part of speech), suffixation exceptions, origin, Semitic root, Semitic pattern, and morphological features.

The addition of extra features and new entries into the lexicon is handled through several scripts that help compile information from multiple sources for ease of updating in case of changes in the source lexicons. The resulting lexicon file used by the parser is a tab-separated text file that can be viewed in any cell-based application, such as Microsoft Excel, for a better reading experience. This file is

intended to be read-only, as any direct changes would be overwritten in the next compilation.

In this section, the types of information contained in the lexicon are described. Some example entries from the lexicon can be found in Table 4.

Table 4. Sample Lexicon Entries

entry	*variants*	*type_en*	*suffixation*	*origin*	*semitic_ root*	*semitic_ pattern*	*morph_ features*
müteakip	müteakip,müteakib	adverb		Arabic	'Kb	mutaFāCiL	
gibi	gibi	conjunction		Turkish			
yo	yo	interjection		Turkish			
yok	yok,yoğ	conjunction	ku, -ğu	Turkish			
rahat	rahat	interjection		Arabic	rwH	FaCLa(t)	
Demokles	demokles	proper_noun		Greek			
müstehzi	müstehzi	adjective		Arabic	hzA	mustaFCiL	
alaycılık	alaycılık,alaycılığ	noun	ğı	Turkish			
eşhas	eşhas	noun		Arabic	şxŞ	aFCāL	Number=Plur

4.1.1 Entry name

The entry name is the lexical entry as it appears in TDK dictionaries. Although there are disagreements over how to write some words, as TDK seems to maintain a more structured dictionary, this framework takes *TDK Güncel Türkçe Sözlük* (GTS from now on) as the standard for the forms of each lexical entry, unless another source uniquely has some entries while GTS misses it.

In TDK convention, verbs typically appear in infinitive form with the suffix *+mAk* (realized as *+mek* or *+mak*), such as *uçmak* (flying). The lexicon contains the bare forms of verbs, like *uç* (fly).

The original TDK dictionary is kept in a JSON format, and the only direct edit on this file is done for the entry *nan* (bread, borrowed from Persian). It is converted to *nân* (a valid alternative form in use) by adding a circumflex, as it is

interpreted as NaN (not a number) when read by several Python packages, including Pandas. The original form *nan* is re-generated at a later stage under variants.

4.1.2 Suffixation exceptions

Some Turkish roots and stems undergo vowel reduction, or final-obstruent devoicing is reverted when they are suffixed. For such roots and stems, suffixation exceptions are added for entries in a form that shows how an accusative case marker is added to the word in question.

Entry	Suffixation	Phenomenon
denk (equal)	*gi*	(voicing)
burun (nose)	*rnu*	(vowel reduction)

4.1.3 Variants

The variants column includes the alternative spellings of an entry in case they have circumflexes, contested spellings, suffixation exceptions, or other special cases. As noted before, there is no total agreement over the written forms of all words by everyone, but instead of having separate entries within the lexicon, having them as variants of a single entry has the benefit of synchronously normalizing while morphologically parsing.

One of the most critical functions of variants is handling suffixation exceptions such as voicing and vowel reduction. If suffixation exceptions exist, they are applied to the entry name while generating the lexicon file, reducing the operations done in parsing runtime.

Entry	Suffixation	Phenomenon	Variants
denk (equal)	*gi*	(voicing)	*denk, deng*
burun (nose)	*rnu*	(vowel reduction)	*burun, burn*

While parsing, variants are taken as the starting point, and entry names are used as deep form roots. Therefore, *burna* (nose, dative) and *burnu* (nose, accusative) can be represented as "*burun,* +H, Case=Dat" and "*burun*, +A, Case=Acc" respectively.

4.1.4 Type (or part of speech)

Part of speech is the most important cue of which affixes a stem can take, and it is the bare minimum feature that exists along with the entry names in all morphological parser lexicons. Our lexicon contains the following 13 items as parts of speech: noun, pronoun, proper noun, verb, adjective, adverb, adposition, determiner, numeral, particle, conjunction, interjection, and onomatopoetic. In this way, it contrasts with the UD Universal POS Tags (UPOS) description, which contains 16 parts of speech. Our lexicon does not contain separate subordinating conjunctions (SCONJ) and coordinating conjunction (CCONJ) sets, but rather a merged conjunction set due to BOUN UD Treebank not being consistent about which conjunction belongs to which subgroup. Symbols (SYM) are also not used in our framework, as they can be dynamically handled where necessary and are not covered in BOUN UD Treebank. Auxiliary (AUX) is also skipped because it is generally used to tag a single suffix of a word rather than being a standalone POS.

This categorization is not necessarily linguistically motivated; it is rather done for the purposes of more precise affixation. For example, onomatopoeic words are a subset of interjections. They are initially processed as belonging to a separate

category since they can take derivational suffixes like *+tH* to form nouns such as *gıcırtı* and *gürültü* or *+DA(n)* to form verbs such as *kıpırdan(mak) and çatırda(mak)*. In contrast, non-onomatopoeic interjections cannot take these suffixes. After morphological analysis, UD non-compliant parts of speech are converted into UD-compliant counterparts.

4.1.5 Origin

Turkish etymology is not an uncontested area, especially due to the remnants of the discussions that started with the Sun-Language Theory and the acceptance of several foreign roots as originally Turkish. Notable contemporary etymological dictionaries include *Nişanyan Sözlük* and *Eren Türk Dilinin Etimolojik Sözlüğü* (ETDES). The relevance of word origin is mainly due to suffixation constraints of morphemes like *+zede*, which only takes Persian stems, and *+iyet*, which only takes Arabic stems, without having a Semitic root and pattern information that produces the ultimate word form in the lexicon.

Regarding the etymological origin of words, there are two crucial concepts: *immediate source* and *ultimate source*. The immediate source is the source language from which the target language acquired the word, while the ultimate source is the farthest traceable origin of a given word. For example, for the word *kahve* (coffee), the immediate and ultimate sources are one and the same: Arabic. However, for the word *kafe* (cafe), the immediate source is French, while the ultimate source is again, Arabic, and Turkish is one of the intermediate sources with the form *kahve*.

Origin information in this lexicon is mainly from the GTS, as Nişanyan Sözlük prohibits the use of its database without consent. However, it is worth noting that Nişanyan and TDK do not fully agree on word origins. See Figure 2 for a

comparison between the percentage of origins of the 10694 lexical entries that exist in both dictionaries.

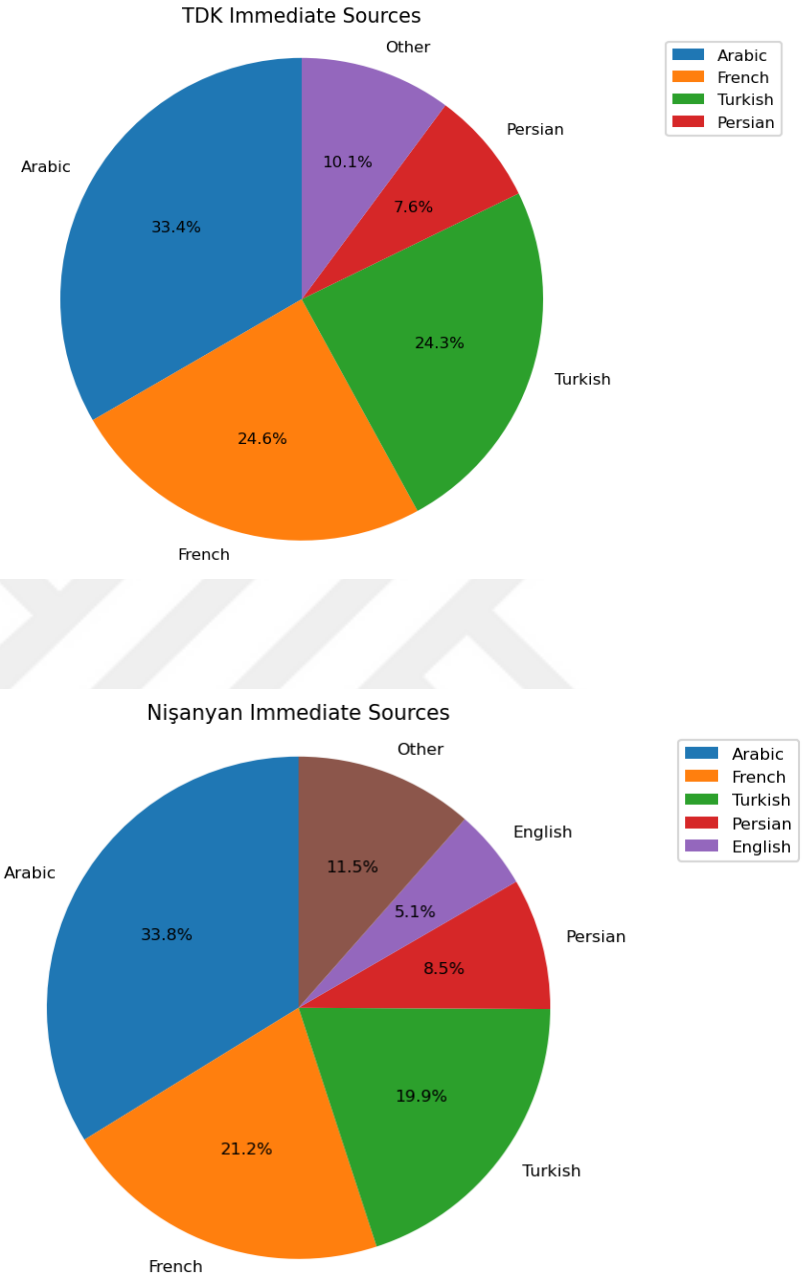


Figure 2. Comparison of TDK and Nişanyan dictionaries on origin information

This disagreement stems from a few factors. First, TDK marks unknown-origin words with the same code as Turkish ones. Therefore, Turkish numbers include unknown-origin words. Second, the notion of the immediate source is

different from the ultimate source, and it is normal to see disagreements, especially in the case of English and French.

Table 5. Conversion Table for Arabic Letters

Arabic Letter	Conventional romanization	Letter in framework	Example word starting with the letter	Root of example in framework
أ	ʾ (or 'a')	A	ahali	Ahl
ب	b	b	basit	bsT
ت	t	t	tacir	tcr
ث	th	S	sevap	Swb
ج	j	j	camia	cm'
ح	ḥ	H	harf	Hrf
خ	kh	x	haber	xbr
د	d	d	dünya	dnw
ذ	dh	D	zeki	Dky
ر	r	r	rahat	rwH
ز	z	z	zeytin	zyt
س	s	s	seyahat	syH
ش	sh	ş	şimal	şml
ص	ṣ	Ş	sayfa	ŞHf
ض	ḍ	J	zayıf	J'f
ط	ṭ	T	tavır	Twr
ظ	ẓ	Z	zulüm	Zlm
ع	ʿ (or 'e)	'	ilim	'lm
غ	gh	g	garip	grb
ف	f	f	fazla	fJl
ق	q	K	kadife	KTf
ك	k	k	kitap	ktb
ل	l	l	lüzum	lzm
م	m	m	madde	mdd
ن	n	n	nafile	nfl
ه	h	h	hedef	hdf
و	w (or 'u')	w	vatan	wTn
ي	y (or 'i')	y	yemin	ymn

4.1.6 Semitic root

The general trend in Arabic morphological parsing, as can be seen in the case of ElixirFM (Smrž, 2007), is analyzing words as a combination of a root, pattern

(*wazn*), and suffixes. Based on this approach, our lexicon contains Semitic roots based on a conversion of Arabic script into Turkish letters. This conversion is case-dependent to reduce the number of unicode characters used in the framework by not using characters such as *ḥ, ṣ, ḍ*. See the conversion table in Table 5 for letters used in Arabic roots, converted versions in the framework, example Turkish words, and the roots of those example words. These roots are taken from the ElixirFM (Smrž, 2007) dataset through manual matching of Turkish and Arabic words.

4.1.7 Semitic pattern

The second component in the description of Arabic morphology (Smrž, 2007) is the *wazn*, the pattern of the word. ElixirFM denotes the patterns with the inflections of root *f'l* (action) and marks the variables for trilateral roots with *F, C,* and *L*, such as *FaCCāL* as the pattern of *ḥammāl* (*hammal* in Turkish, meaning "carrier"). There are 119 different Arabic patterns in the lexicon.

4.1.8 Morphological features

One of the main objectives of a morphological parser is figuring out which morphological properties it carries. Oflazer's (1993) convention, as well as other two-level morphology conventions use a specific feature notation. However, maintaining a morphological dataset with a good inter-annotator agreement and external checks is a difficult task. Therefore, this framework adopts Universal Dependencies (Nivre, 2020) morphological features with a few deterministic tweaks.

To add UD features such as the pronoun *ben* (me) for a word, the UD features are first added to a separate tab-separated text file where the first tab contains the entry, the second contains the part of speech of the related entry that already exists in

the lexicon, and the third contains the UD features to be added into the "features" tab of the lexicon.

ben pronoun Person=1|PronType=Prs|Number=Sing

This format ensures adding these features for the pronoun *ben* and not the noun *ben* (beauty spot).

Based on the analysis of BOUN UD Treebank (Marşan et al., 2022), as well as what Sak et al. (2008) morphological parser produces, bare form verbs are assumed to have the following UD features:

Polarity=Pos|Person=3|Number=Sing|Tense=Pres|Mood=Imp

Similarly, bare form nouns can have one of the following:

Case=Nom|Number=Sing|Person=3

Case=Acc|Number=Sing|Person=3

Oflazer (in Oflazer & Saraçlar, 2018, p. 29) argues that no case morpheme implies a nominative case, and for a word to have an accusative case, an accusative marker morpheme is a requirement. However, the accusative case can be unmarked in Turkish, as can be seen in the following examples:

Kağıdı asacak pano *bulmam lazım.* (I need to find a *board* to post this paper.)

Mehmet iş *arıyordu.* (Mehmet was looking for a *job*.)

Both examples show bare form nouns that have accusative case form.

Although it is convenient to assume the lack of case morpheme as an indicator of nominative form for a simpler analysis and a lighter-weight disambiguation module, it is important to acknowledge this as a challenge that should be resolved through disambiguation. This is especially important as UD

efforts already go through a tagging of accusative cases in bare forms, and seq2seq morphological taggers such as SpaCy Turkish Morphologizer[6] is already on this path.

4.2 Affix vocabulary

Affix vocabulary is a tab-separated file intended to be edited with a cell editor like Microsoft Excel as a development environment for ease of filtering the entries and having a more convenient visual representation than plain text.

The affixes in this file loosely follow the order in which they are presented in Göksel & Kerslake (2005) but not necessarily in the same sequencing.

Each affix row has a unique affix ID, a generalized representation, variants based on the expansion of generalized representation, input parts of speech, output parts of speech, input features (in UD form), output features (in UD form), wipe features, positional type, peculiarities, and examples. A sample from the affix vocabulary file with some columns are omitted can be seen in Table 6.

Table 6. Sample Affix Entries

affix_id	*general*	*allomorphs*	*input_pos*	*output_pos*	*output_features*	*function*	*peculiarity*	*example*
DER001	(A)C	c,ac,ç,aç,ec,eç	adjective	adjective		derivational		anaç, kıraç
DER009	(A)l	l,al,el	noun	adjective		derivational		yerel, ulusal
DER011	(A)lgA	lga,alga,lge,elge	verb	noun, adjective		derivational		çizelge
DER031	Hm	ım,im,um,üm	verb	noun		derivational		bölüm, seçim
INFL048	(H)n	n,ın,in,un,ün	pronoun	pronoun	Case=Nom| Number[psor]=Sing| Person[psor]=2	inflectional		
INFL049	(H)nHz	nız,ınız,(...)	pronoun	pronoun	Case=Nom| Number[psor]=Plur| Person[psor]=2	inflectional		
INFL050	(s)H(n)	ı,sı,i,si,(...)	pronoun	pronoun	Case=Nom| Number[psor]=Sing| Person[psor]=3	inflectional	CANNOT_ END_ WITH_N	
INFL051	(H)mHz	mız,ımız,(...)	pronoun	pronoun	Case=Nom| Number[psor]=Plur| Person[psor]=1	inflectional		

[6] See https://github.com/turkish-nlp-suite/turkish-spacy-models

4.2.1 Affix ID

Affix ID is the key by which several operations, such as feature lookups and rule implementations on morpheme combinations, are carried out. The convention is a prefix DER for derivational, INFL for inflectional morphemes, followed by a 3-digit number, starting from 001. IDs are preferred over generalized morpheme representations as keys since separate morphemes can appear in the same form but have different behaviors.

A directly hash-based ID naming convention can be more beneficial for lookup times, however, this is to be explored.

4.2.2 Generalized representation

Representation of morphemes loosely follows the convention of Sak et al. (2008), following Oflazer (1993). Uppercase letters denote letters that get realized in multiple forms based on context. Table 1 in Section 1 contains the conversion rules.

Some morphemes may also bring thematic letters between them and some stems. For such cases, these thematic letters are deemed optional additions for these vowels and are denoted in parentheses.

4.2.3 Allomorphs

Allomorphs are expansions of generalized representations into possible forms that can be attached to words. For example, the derivative suffix *+(y)Hş* can appear as *+ış, +yış, +iş, +yiş, +uş, +yuş, +üş, or +yüş* depending on the phonology of the verb stem and generate nouns *kaç+ış*, *ara+yış*, *gel+iş*, *işle+yiş, uç+uş*, *kuru+yuş*, *düş+üş, and yürü+yüş*.

4.2.4 Input part of speech

As explained under 5.1.d., part of speech (PoS) is a decisive piece of information that helps us understand which affixes can be attached to which stems. For example, in the example in 5.2.c, derivational *+(y)Hş* can only take verbs as input. One cannot take a noun, for example, *bıçak* (knife), and produce **bıçakış* with this affix.

For each affix row, at least one input part of speech must be defined.

4.2.5 Output part of speech

Similar to the input part of speech, what sort of a word an affix can produce when attached to a compatible stem is important. The same example, derivational *+(y)Hş*, only produces nouns given a verb input. Although *+(y)Hş* cannot directly take the noun *bıçak* (knife) as an input, if *bıçak* takes, for example, derivational morpheme *+lA* that takes nouns, adjectives, or interjections as input and outputs verbs, it can become the verb *bıçakla* (stabbing, verb) and then take *+(y)H* to produce *bıçaklayış* (stabbing, noun).

Either input PoS or output PoS must contain only one entry. So, if a suffix, such as derivational +CAK, can take adjectives to produce adjectives and take nouns to produce nouns, then this suffix should be distributed into separate rows that clearly define the inputs and outputs. Otherwise, the analyzer may attach +CAK on an adjective and assume that it can produce a noun, while this is not the case.

4.2.6 Input morphological features

Some affixes require certain morphological features on a stem to be attached. For example, derivational +(H)ncH requires a cardinal number as an input, and

NumType=Card is the UD feature that denotes a stem that fulfills this requirement. This is an optional column for entry rows.

4.2.7 Output morphological features

The output morphological features column defines the features that should be added to the word when an affix is attached to a stem. Since most derivational morphemes do not have UD features to add, this is also an optional column, and where it is not empty, it can contain multiple sets of UD features in a comma-separated form. As an example, inflectional *+(H)m* has the following output features:

Case=Nom|Number[psor]=Sing|Person[psor]=1

4.2.8 Wipe features

Wipe features consist of the features that are to be completely wiped if they exist in the stem form. For example, inflectional *+(y)Hp* as in *gel**ip*** (by coming) adds the features:

Polarity=Pos|VerbForm=Conv

While wiping the following:

Person=*|Number=*|Tense=*|Mood=*

If there are any person, number, tense, or mood features in the stem, these features are removed once this suffix is agglutinated.

4.2.9 Positional type

Denotes whether an affix is a prefix or a suffix. Adding this feature as a column helps preventing the use of separate files for prefixes. Circumfixes or infixes do in

fact exist in lexical borrowings but are not productive or distinguishable, therefore are not covered under affixes.

4.2.10 Functional type

Functional type denotes whether an affix is derivational or inflectional. Although the current affix IDs already contain cues as to which type of a morpheme is in question, it is beneficial to have this column in case a more explanatory or hash-based naming would be more efficient in a given application.

4.2.11 Peculiarity

Many affixes in the vocabulary contain extra constraints or cues that help their agglutination with stems and further affixes. Unlike other columns that are processed through the main analysis function, these peculiarities are flags that call extra rule checks. Some examples are as follows:

- TAM1, ..., TAM5: Tense-aspect-modality slot, based on Göksel and Kerslake. A TAM(n) suffix cannot precede a TAM(n-1) suffix.
- ARABIC_ORIGIN: Only attaches to stems with Arabic origin.
- REMOVE_LETTER: Removes one letter from the stem (çabuk +CAK -> çabucak).
- REMOVE_LETTER_OPTIONAL: May or may not remove one letter from the stem.
- CANNOT_END_WITH_N: Can have the letter "n" if it is followed by another suffix, but cannot appear at the end of the word with an "n."

Since these operations are handled with exception rules, this tab triggers a core part of the analyzer. However, if, for some reason, the main analyzer function

would be refactored to handle these operations in some other way, these flags may be distributed to multiple columns.

4.2.12 Examples

The examples column is filled with one or more examples of a given morpheme for all derivational morphemes and some inflectional morphemes to describe better what an affix stands for. Examples are significant in error analysis of the parser, as no semantic markers are currently supported in the framework.

4.3 Constraint resources

Although leniency and descriptivism are among the key motivations behind this framework, the inclusion of derivational morphemes causes overgeneration beyond measure. Letting the analyzer overgenerate as much as it possibly can and then leave disambiguating to the disambiguator model is, of course, a choice, especially since there are works that even use unsupervised learning to infer morphological rules.

However, it is far more likely for words that appear the same to adhere to the same or similar word formation paths. As an example, let us take the word *gözlükçülük,* an example extensively used in Turkish morphological processing literature. Just by simple string matching without any constraints, possible parses include (but are not limited to):

- göz +lük +çü +lük: En. opticianry (noun), *+lük* as in *günlük* (daily)
- *göz +lük +çül +ük - En. eyeglassic act, *+çül* as in *çürükçül* (saprophytic), *+ük* as in *gözük-* (appear)
- *göz +lük +çül +ük - En. we are eyeglassic, *+çül* as in *çürükçül* (saprophytic), *+ük* as the colloquial first plural.

Most probably, only the first parse is correct, and the others are either straight-up wrong or contain linguistic gaps.

The framework in this thesis includes heuristic constraints for seemingly straightforward parses like the one above while leaving enough leniency for linguistic gaps in cases where there are no predictably correct parses. Some of these constraints rely on a lexicon of unbreakable roots and segmentation overrides.

4.3.1 Segmentation overrides

Segmentation overrides are manual annotations of morpheme boundaries within words. Overriding segmentations are stored in a tab-separated text file, where the first column contains the regular surface form, and the second column has segmentations with "+" between affix boundaries and "-" between compound word boundaries. Prefixes are separated from roots with "/" to prevent misrecognition of the root. Starting point for the segmentation overrides is the lexicon file; entries are manually tagged if they would benefit from an override, or in other words, if they are likely to have incorrect parses due to lack of overrides.

A similar effort in this direction is MorphoLex. However, Turkish MorphoLex contains generalized forms and does not contain any distinction between compounding and affixation. In the overriding segmentations text file, segmented forms are still in their surface form, which may result in the suboptimal performance of our framework. There are also different design choices on which morphemes should be separately recognized. The example below shows these differences:

Entry: başdanışmanlık (key advisory)

This framework: baş-danış+man+lık

MorphoLex: baş+danışman+lHk

4.3.2 Unbreakable roots

Overriding segmentations is a very invasive method that can result in incorrect parses being forced into the system. To overcome this difficulty, unbreakable roots are used as an override over overriding segmentations. Some word forms, as well as proper nouns are processed as unbreakable roots and are not analyzed further.

4.3.3 Compound words

Although segmentation overrides include compound words, TDK GTS does not contain all compound words. In addition to the resources mentioned before, an extra list of compound words with their split boundaries is created through a simple heuristic. The assumption is that almost all compound words in Turkish follow the fate of sometimes being written separately.

Based on this assumption, if a word *n* and next word *n+1* occur in a development corpus, and a compound of (*n*, *n*+1) also occurs as a single token, it is exported to a text file. Some common tokens that do not typically generate compounds but appear in the results due to misspellings and issues in corpus processing, such as "*şey*" and "ler," and some affixes are excluded from this search.

This list of compound candidates is then manually cleaned to only have a list of actual compound words.

CHAPTER 5

MORPHOLOGICAL ANALYZER

This section explains the mechanisms by which the morphological analyzer component of the framework operates.

5.1 General structure of analyzer

Unlike the examples in the literature, our morphological analyzer implementation is not based on rule-based string matching instead of FST. This choice is not due to the computational limitations of FSTs, but rather due to the ease of implementing exceptions and rules.

Another reason behind choosing a rule-based approach is the comparably straightforward integration of object structure into various steps of analysis and disambiguation. Use cases of morphological data are not uniform throughout the industrial or academic use. As can be seen in the example of SpaCy (Honnibal & Montani, 2017), an on-demand supply of linguistic features is nice to have, compared to string operations with all information present at all times. Most of the frameworks that support Turkish morphology rely on string manipulation at all levels, and for taking only the relevant information, end users generally need to use regular expressions or similar operations. The framework proposed in this thesis focuses on the possibility of varying needs of the end user in terms of features included as a part of the output.

Adopting an object-oriented structure is one of the solutions to the issue of choosing the necessary output for a given application. Our morphological analyzer

depends on such a strategy, and further analysis, such as a disambiguation step, does not conflict with this requirement.

The morphological analyzer is written in pure Python with minimal third-party package requirements. The only notable exception is the Pandas library (McKinney & others, 2010), which is used for operating on lexicon and vocabulary files. On top of that, two newly defined objects are used: *Word* and *Affix* objects.

Word object has the properties surface form, deep form, prefix, root, stem, suffixes, morphological features, and part of speech.

Surface form denotes the actual token within the text, deep form denotes the form that is segmented based on root and suffixes, prefix contains the prefix(es) of the word, root is the smallest root word analysis of the word, stem denotes the latest stem on which the latest suffix has attached to, suffixes are the IDs of the suffixes that attach to the root and stems, and part of speech is the latest part of speech after the transformations of the root by the affixes.

Affix object has the features affix ID, affix representation (general form), allomorphs, input PoS, output PoS, wipe features, positional type, functional type, peculiarity, example, and metadata. Apart from the checks that ensure the availability of a given affix to a given stem, affixes are stored based on affix ID on the Word object. Therefore, Word object contains the affix IDs on its affix-related properties, which are generated dynamically through the read-in affix files.

5.2 Analysis pipeline

The morphological analyzer expects a single word or a list of words to be analyzed, a lexicon DataFrame, an affixes DataFrame, a cache dictionary, and some resource lists for rules as input.

If there is an analysis for a given analysis input in the cache, the analysis pipeline is bypassed, and the cached parse is returned.

The core algorithm behind the morphological analysis is a modified version of breadth-first search. Breadth-first search normally assumes the task at hand is a graph traversal problem, and we are reformulating the morphological analysis as such a task.

For a given input word, the matching roots are first retrieved from the lexicon DataFrame based on whether the input word starts with one of the variants of a lexicon entry. As described in section 4, the variants section of a lexicon entry includes alternative forms that undergo phonemic transformations.

A queue and a set to contain the visited vertices are initialized based on the matching root hypotheses. At each iteration, vertices (which are hypotheses in our use case) of stem and affix combinations are compared against the input to see if they match. If a combination matches the word partially until the same length or fully and passes all the rule requirements, such as matching part of speech requirements, the vertex is added to the end of the queue.

This process continues until no more new vertices are generated, and all the vertices are visited.

It should be noted that BFS is not an ideal algorithm for this task, and its performance is suboptimal. There is room for improvement with better-suited algorithms and efficient implementation. However, from a behavioral standpoint, the output of this function is identical to any better algorithm if no extra rules or pruning methods are introduced.

5.2.1 Handling of apostrophes

If the input contains an apostrophe, the left side of the apostrophe is considered as a proper noun that should not be analyzed further. Instead, a default root hypothesis with a root and proper noun as a part of speech is generated. The general assumption is that the suffixes after the apostrophe are inflectional. While derivational suffixes generally attach to proper nouns without an apostrophe, it is observed that some authors prefer attaching some suffixes with an apostrophe, such as:

> (...) *bir bakarsınız, "uzlaşmaz bir* Marx'çı *olarak", Bertolt Brecht'in kuramlarını yerleştirmeye çalışır* (...) (Yücel, 2017)[7]

Therefore, derivational suffixes that do not share the same surface form as any inflectional suffixes are enabled to analyze the part after the apostrophe.

If the right side of the apostrophe does not match any valid suffix combination, foreign names with apostrophes in them (such as O'Connor) are tested as hypotheses.

5.2.2 Handling of compound words

Possible compound words are stored in the lexicon as compound words and are generated using the method explained in 4.3.c. Since the lexicon contains them with their morpheme boundaries, compound words are added as a hypothesis and treated as the stem, but split into multiple roots at the end of the analysis.

[7] (...) and then you see that, "as an irreconciling Marxist", he tries to implement the ru

5.3.3 Handling of overriding segmentations and prefixes

On top of derivational suffixes in many words, the preferred method of prefixation is relying on the overriding segmentations for recognizing where and when to generate a hypothesis with prefixes.

Overriding segmentations prevent multiple hypotheses with the same elements from being created. For example, let us take the input *gözlüklerim* (my eyeglasses). If there was no entry for *gözlük* (eyeglass) in the lexicon, the analyzer would conclude that it derived from *göz*. However, since *gözlük* is already an entry in the lexicon, there is the risk of two separate hypotheses being created for the same word:

1) Word.deep_form = ['göz', 'lHk', 'lAr', 'Hm']

2) Word.deep_form = ['gözlük', 'lAr', 'Hm']

By applying segmentation overrides, the initial hypothesis for the root *gözlük* is initiated as a combination of *göz* and *-lük*, where the analyzer is asked to find a generalized form for the allomorph *-lük.* As such, only the first hypothesis above is generated.

As for prefixes, for example, the word *anormal* is represented as *a/norm+al* in the segmentation overrides. This representation denotes that *a* is a prefix, and *al* is a suffix. Therefore, when *anormal* is taken as a root hypothesis from the lexicon, the hypothesis passed to the queue is already a Word object with the following relevant properties:

Word.deep_form = ['a', 'norm', 'Al']

Word.affixes = ['PRE001', 'DER009']

Word.root = 'norm'

Word.stem = 'anormal'

If there is no parse available at the end of the pipeline with this approach, the analyzer starts its analysis once again, this time with a backup scenario, by adding prefixes at the beginning of root hypotheses. This helps prevent attaching prefix *a+* to any word that starts with the letter *a* and has a valid parse without a prefix.

5.2.4 Handling of numbers

Word representations of numbers are a part of the lexicon and are treated as regular inputs. However, there are some extra operations on numeric representations.

If the input consists of numerals only, a generic number root with default properties (part of speech: NUM and morphological feature: NumType=Card) and a single hypothesis is returned without further processing. The same default hypothesis is used for tokens with percent signs and some other mathematical indicators.

If the input contains a number and a dot at the end, another hypothesis is generated with NumType=Ord for ordinal number. The assumption is that preprocessing separates full stop dot while keeping the ordinal indicator together.

If the input contains extra characters that may be suffixes, the numeral part is taken as root, and the regular analysis pipeline is followed.

Current analysis implementation relies on a preprocessing where non-suffix letters, such as *B* in *221B,* are split into a list, ['221', 'B']. In different preprocessing scenarios where an *n* number of arbitrary letters can be a part of a token together with numerals, new rules can be added for a more robust coverage.

5.2.5 Semitic morphological analysis

As explained in earlier sections, Semitic morphological analysis is a component required to analyze Turkish morphology thoroughly. However, since this is not a

common practice, it is an optional component that can be deactivated without changing the analyzer's behavior in other operations.

Analysis of Semitic roots and meters is carried out after the root hypothesis is attributed to a word. If the root of a given Word hypothesis matches the form of a Semitic root and meter combination, then this root is deemed a combination of this root and meter. No dynamic parsing is carried out at the analyzer level for these roots, and the only operation is the retrieval of root and meter information from the lexicon.

One design choice to note here is that the Semitic meters and suffixes are treated as agglutinations over the root in our framework. For example, the deep form of *meskenlerin* ("of the residences") is:

Word.deep_form = ['skn', 'maFCaL', 'lAr', 'Hn']

Word.root = 'skn'

This is to ensure the possibility of incorporating root and meter information as subwords. While this approach is unconventional, we have not seen any scenarios where this can become an issue. However, if further research in this direction shows the need for another representation, this component can be revised.

5.2.6 The distinction between inflectional and derivational suffixes

In Turkish, the general rule is that derivational suffixes cannot come after inflectional suffixes. However, some derivational suffixes are more productive than others and can attach to almost anything. One such example is the suffix *+cH,* which generally corresponds to similar semantic information to *+ist* in English. It can even be placed right after phrases such as "ben yaptım olduc*u*" (I-did-it-and-it-turned-out-fine-*ist*).

A rule that prohibits derivational suffixes after inflectional suffixes is used to prevent less productive suffixes from attaching to bases with inflections. Suffixes such as *+cH* and *+sAl* are included in an exceptions list, which can be dynamically controlled based on output observations.

5.3 Analyzer performance

The performance of morphological analyzer is based on whether any of the hypotheses match the feature sets presented in a given Universal Dependencies treebank's test set. Current implementation covers 87% of the test set of BOUN UD Treebank (Marşan et al., 2022) with exact matches.

This means, in around 13% of the cases, the parser produces an output that is not exactly matching the BOUN UD Treebank features, but only partially. The analyzer produces backup “proper noun” parses for unknown words, in a total of 131 times over the test set size of 11,822 words, meaning a coverage of over 99%.

In another treebank, the UD Turkish Penn Treebank's (Kuzgun et al., 2020) test set the exact matches are 84% and the total coverage is 98%.

CHAPTER 6

MORPHOLOGICAL DISAMBIGUATOR

This chapter describes the approach of the framework to the morphological disambiguation task.

6.1 Previous methods

Morphological disambiguation is the task of choosing the correct parse given an input, possible morphological parses, and context. Any possible morphological parse that is given as an input to the morphological disambiguator must be viable parses in some context.

Hakkani-Tür et al. (in Oflazer & Saraçlar, 2018) describe the task as an extension of part of speech tagging and list the methods used in Turkish morphological disambiguation as constraint-based morphological disambiguation, rule-learning, models based on inflectional group n-grams, and discriminative methods. These can be grouped into larger categories of rule-based and statistical methods. A relatively recent trend in morphological disambiguation is the use of Conditional Random Field models on top of the character and tag-level Long Short-Term Memory (LSTM) language models (Shen et al., 2016).

The current trend in natural language processing in almost any token tagging or classification task is fine-tuning larger language models instead of training a neural network model from scratch only with the task-specific dataset, as Shen et al. (2016) do. Another attempt to provide morphological analyses for given input words is SpaCy Morphologizer, where Universal Dependencies datasets are used for training a layer within a linear NLP pipeline.

6.2 Fine-tuning transformer language models for disambiguation

The method adopted in this framework is converting the morphological disambiguation into a multitask token classification task that benefits from the contextual embeddings provided by masked language models, such as BERT. Although it is not impossible to fine-tune or prompt autoregressive language models such as GPT variants, masked language models are more intuitive to use for token classification tasks.

The architecture we propose relies on training three separate classifier layers in parallel on top of the pre-trained model. For this, a dataset containing POS tags, morphological features, and segmentations is required.

6.2.1 Dataset creation

Due to the lack of available high-volume hand-tagged disambiguation datasets (TrMor datasets are synthetically generated) and the costs overweighing the benefits of creating a publicly available one, this framework proposes a method to convert Universal Dependencies treebanks into disambiguation datasets. This enables the framework to benefit from each advancement in Turkish treebanks.

As a proof of concept, BOUN UD Treebank (Marşan et al., 2022) is used for the dataset generation. Train, test, and validation splits of the treebank are kept as they are.

In generating the training dataset, each word in each sentence of the treebank is parsed with the morphological analyzer. Each analysis is compared against the part of speech and the morphological feature given in the treebank, and it is ensured that the root analysis is no longer than the lemma marked on the treebank.

One key difference with the treebank is the POS marking of verbal nouns. This treebank marks the verbal nouns as VERB, and since they act as nouns in suffixation, they are converted into NOUNs in the dataset creation.

BOUN UD Treebank has a potentially useful feature under the miscellaneous features column: "DerivedFrom". For example, *pulsuzduk* has the lemma *pulsuz,* but it has DerivedFrom=*pul* which shows the ultimate root.

However, only 249 entries in the training set have a DerivedFrom feature, and many apparent roots are not tagged as such. Therefore, for the time being, the dataset creation process relies on the analysis outputs being uniquely identifiable by the morphological features, part of speech, and whatever cue is available from the lemma. Out of all the analyses of a given word, one "best parse" is chosen based on comparison with UD features.

Segmentations in this dataset are not word-specific, but rather a generalized form, where the deep forms are converted into strings of P (prefix), R (root), S (Semitic meter), I (inflectional suffix), D (derivational suffix), and M (punctuation mark). This conversion minimizes the issues caused by mismatches in subword tokenization and morphemes and prevents no-parse scenarios for unseen words.

Combinations of morphological features are treated as unique sets rather than an open dictionary of features. This choice is again due to the mismatch between the subword tokenization and actual morphemes.

A JSON entry for each sentence is created from the analyses, where lists of items are stored under the keys sentence, pos_labels, correct_pos, morph_labels, correct_morph, segments, and correct_segment. An example of this structure can be found in Appendix A.

In that example, pos_labels, morph_labels, and segments are lists of features in each hypothesis from the morphological parser. Even though morphological labels for *hiç* (nothing) as adverb and noun are both empty, to be able to convert the analysis hypotheses back into Word objects, the empty hypothesis is duplicated to keep the indices consistent.

After analyzing the whole UD treebank training set and generating JSON entries for each sentence, a JSON file is created to be used in the training of the disambiguator.

6.2.2 Model architecture and training

For the proof-of-concept implementation, the morphological disambiguator takes advantage of the model weights in BERTurk[8] cased 32k, which is a pre-trained BERT model available on Huggingface[9] and can be used through transformers (Wolf et al., 2020) Python library. Three linear classification layers over the BERT model are parallelly placed for learning morphological parsing.

The model takes the inputs from the sentence and the correct morphological, POS, and segment entries and tokenizes the regular words with the pre-trained BertTokenizer that comes along with the BERTurk model. POS tags, segments, and morphological features are separately vectorized with one hot vectors, with padding where necessary.

One design choice here is the tagging of each subword with the properties of the complete word. There are alternative approaches, such as only tagging the first subword of each word and adding [PAD] tokens for the rest or only tagging the last

[8] https://github.com/stefan-it/turkish-bert
[9] https://huggingface.co/dbmdz/bert-base-turkish-cased

subword. The ideal scenario would be having actual morphemes as subwords and tagging each morpheme with the feature that morpheme carries. However, this is a limitation that requires a different tokenization approach for pre-training transformer language models.

Then, both the BERT model and the three heads are trained together. Weights in the BERT model are not frozen; therefore the raw sentences also contribute to the model.

Figure 3 shows the training pipeline, and Figure 4 shows the architecture of the model itself.

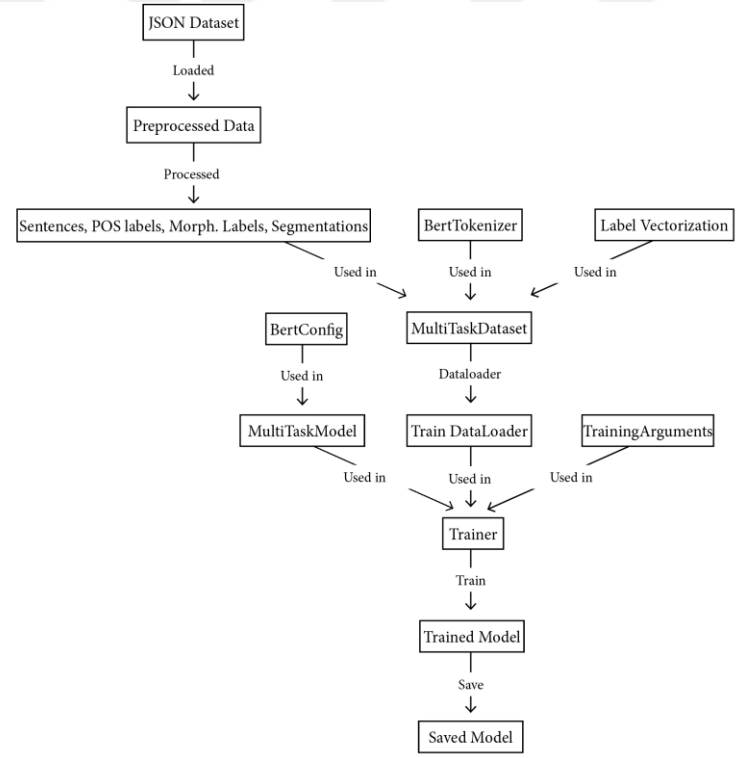


Figure 3. Model training steps

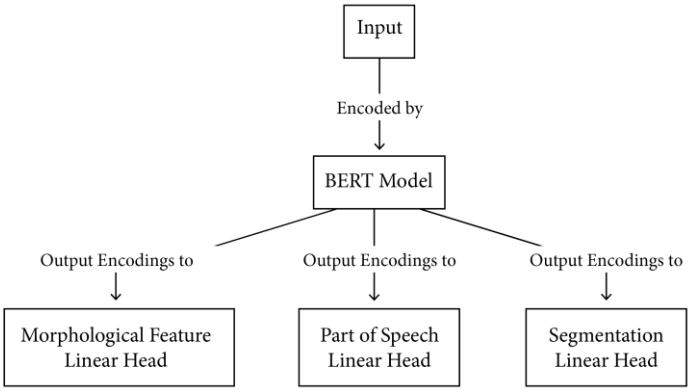


Figure 4. Model architecture

The model has 12 attention heads and 12 hidden layers and uses GELU activation function in these hidden layers of size 768. It uses the regular BERT architecture with a vocabulary size of 32000.

Fine-tuning of this model together with three task heads take around 6 hours before overfitting on a consumer-grade Nvidia RTX3090 GPU with 24GB of VRAM.

6.2.3 Using a sequence-to-sequence model as a disambiguator

As can be seen in the training data and the architecture itself, up until this point, the model trained with this methodology is a model that takes a raw text input and gives three separate outputs: PoS, morphological features, and segmentation. There is nothing that ensures these three outputs are compatible with each other or the output will be a valid hypothesis. Furthermore, it is not a morphological disambiguator, but rather a sequence-to-sequence (seq2seq) morphological tagger.

To use this model as a disambiguator for potential parses, we are applying a mask to the logits for each head before the softmax layer. For example, if the word *hiç* (nothing) can either be a noun or adverb, then the raw logit probabilities of all

other parts of speech are set to negative infinity. This forces the model to pick one of the analyzer hypotheses, rather than any other part of speech.

6.3 Model evaluation

The morphological parser model is evaluated based on the test split of the BOUN UD Treebank and the UD Turkish Penn Treebank. The test sets are created with the same procedures as the training set. The accuracy of the model in seq2seq tagging and morphological disambiguation scenarios in predicting all elements (part of speech, morphological features, segmentation) of a given input in BOUN UD Treebank can be seen in Table 7. The table also includes the reported[10] POS tagging accuracy of BERTurk cased 32k model on BOUN treebank, which is fine-tuned over the same pre-trained model with more data than we have used in fine-tuning.

Table 7. Model Accuracy with BOUN UD Treebank

Model	*Accuracy*
Multitask BERT model as seq2seq tagger	57.6%
Multitask BERT model as disambiguator	94.6%
BERTurk fine-tuned on POS tagging task	91.4%

As can be seen from the accuracy results, using the model with output constraints based on logit masking before the softmax function dramatically increases the overall task accuracy.

[10] https://github.com/stefan-it/turkish-bert#evaluation-on-boun-datasets

This increase can be partially attributed to the fact that some words only have a single parse output from the analyzer, and limiting the sequence output to this parse ensures correct recognition.

Although our multitask model's accuracy in the disambiguation scenario is higher than the accuracy reported by BERTurk developers, we cannot explain this accuracy increase as a benefit of multitask learning, as we are not following the same fine-tuning parameters such as learning rate, batch size, and number of training epochs, and more training data on BERTurk fine-tuning may not have necessarily increased the performance over this specific test set. However, it can be safely said that our model produces acceptable results.

Following the same preprocessing and training steps as done to the BOUN UD Treebank, we have also generated the analyses and trained a new model for UD Turkish Penn Treebank, as it is a translation of a well-known English treebank (Taylor et al., 2003) and contains several foreign proper words.

Table 8. Model Accuracy with UD Penn Turkish Treebank

Model	*Accuracy*
Multitask BERT model as seq2seq tagger	51.6%
Multitask BERT model as disambiguator	90.7%

Disambiguator accuracy for UD Turkish Penn Treebank is on Table 6.2. Although we do not have a baseline like a previously fine-tuned BERTurk for a specific task, we can still see that using the multitask model as a disambiguator yields better results than the seq2seq tagger use case. Also, this shows that our framework is applicable to multiple treebank conventions.

CHAPTER 7

DISCUSSION

This section lays out the novelties and contributions of the framework and discusses limitations and the steps to address these limitations.

7.1 Contributions of the framework

This thesis contributes to the literature on computational morphology of Turkish by providing a set of new linguistic resources, namely a detailed lexicon with variants of each entry, as well as Semitic roots and meters of them, a dataset of manually tagged morpheme boundaries for each entry in the lexicon, a compound word lexicon, and a spreadsheet of affixes in Turkish along with features explained in section 2.3.

The analyzer of this framework has relatively comprehensive prefix support, along with special handling for compound words. The possibility to expand the compound word lexicon through corpora analysis is an advantage, given that such resources can also be expanded with automatic methods.

The limited availability of high-quality morphological disambiguation data skewed the focus toward repurposing an actively maintained and hand-tagged dataset type, UD Treebanks, as the training data. Larger the treebanks will get, the better the disambiguator will perform without specialized effort for a morphology dataset.

Extensive coverage of derivational suffixes without eliminating any of them for the sake of simplifying the implementation, as well as the coverage of Semitic roots and meters will enable researchers to delve deeper into the historical changes by which Turkish has undergone, as well as better stylistic analysis of texts with a focus on the effects of the Turkish language reform. On top of this, better subword

tokenization based on derivational and inflectional morphemes is made possible with this framework.

The model provided in Chapter 6 is not the ultimate form of the morphological disambiguation component of this framework. It is rather a recipe for fine-tuning any pre-trained BERT-like transformer model into multitask models. Being able to achieve state-of-the-art results by fine-tuning any encoding model within a matter of hours with a consumer-grade GPU enables the users of this framework to customize the analyzer (for example, by extending or reducing the affixes), automatically generate training data from a UD Treebank, and have a fully-fledged morphological parsing framework within a day.

7.2 Limitations and future steps

There are several critical limitations in various components of this framework. The first limitation is that the linguistic resources are not validated by additional linguists. These resources may contain errors or decisions that require the measurement of inter-annotator agreement.

The second limitation is due to the automatic generation of training data for the disambiguator. Since "DerivedFrom" feature in BOUN UD Treebank has not been added for all words that have roots different than their lemmas, it is sometimes required to assume the segmentations in some parse hypotheses are correct, without validation. Depending on the direction Turkish UD treebanks will take, either manual post-processing of the training data or complete marking of root information on the treebank will be required.

One of the most severe limitations is the suboptimal runtime of the morphological analyzer. Since an almost pure Python approach with a suboptimal

algorithm (BFS) at the core is used, analyzing 1000 unique words takes around 49-56 seconds on each core of a consumer-grade AMD Ryzen 5 5600X CPU, between 19-23 seconds on Apple M1 CPU, and between 12-15 seconds on Apple M1 Max CPU. In the best-case scenario, it will take around 20 minutes for a 10-core M1 Max CPU to completely parse 1 million unique word forms and around 150 minutes for a 6-core 5600X to carry out the same task. Any subsequent parsing operation is taking place radically faster due to the only operation being cache retrieval. Although these speeds were acceptable for the development of the framework, given the sheer amount of room for optimization, this is an issue that should be addressed. Automatic conversion of the rules and lexicon into FST inputs and then compiling an FST may be a viable solution that can be explored.

Yet another limitation is the need for a fast and reliable tokenization morpheme-based tokenizer within the framework. One such tokenizer distilled from morphologically analyzed and disambiguated token segmentations can be trained before parsing large amounts of text data for language model training.

CHAPTER 8

CONCLUSION

This thesis aims to lay out the foundations for a new approach to morphological parsing of Turkish. Although there are several limitations, as explained in Chapter 6, the resources and tools made available by this work are a contribution in that direction.

We have observed several complications, especially overgeneration issues due to introduction of more derivational morphemes. These issues ultimately led to creation of new resources which can be used by researchers in areas other than computational linguistics.

The use of UD Treebanks as training data ensure the improvement of our parser's performance as it is an actively maintained project. Additionally, our extensive coverage of derivational suffixes could help expanding the Turkish UD Treebanks with this information.

Both the analyzer and disambiguator components can be optimized further for better coverage and accuracy, and the framework is especially designed to be customizable and extensible.

The resources and codes described in the thesis can be found at https://github.com/OlgunDursun/TurkishMorphologyFramework along with the necessary instructions for various operations.

APPENDIX

SAMPLE ANALYZER OUTPUT

```
output = {
      "sentence": [
         "Hiç",
         "itirazım",
         "yok",
         "."
      ],
      "pos_labels": [
         ["ADV", "NOUN"],
         ["NOUN"],
         ["NOUN", "CONJ", "PART", "ADJ"],
         ["PUNCT"]
      ],
      "correct_pos": [
         "ADV",
         "NOUN",
         "NOUN",
         "PUNCT"
      ],
      "morph_labels": [
         [{},{}],
         [{'Polarity': 'Pos', 'Person': '1', 'Number': 'Sing', 'Tense': 'Pres', 'Mood': 'Opt', 'Aspect':
'Hab'}],
         [{'Number':'Sing', 'Person': '3', 'Polarity':'Neg'},{},{},{}],
         [{}]
      ],
      "correct_morph": [
         {},
         {'Polarity': 'Pos', 'Person': '1', 'Number': 'Sing', 'Tense': 'Pres', 'Mood': 'Opt', 'Aspect':
'Hab'},
         {'Number':'Sing', 'Person': '3', 'Polarity':'Neg'},
         {}
      ],
      "segments": [
         [['R'],['R']],
         [['R', 'S', 'I']],
         [['R'],['R'],['R'],['R']],
         [['M']]
      ],
      "correct_segment": [
         ['R'],
         ['R', 'S', 'I'],
         ['R'],
         ['M']
      ]
   },
```

REFERENCES

Akın, A. A., & Akın, M. D. (2007). *Zemberek, an open source NLP framework for Turkic Languages*.

Akyürek, E., Dayanık, E., & Yuret, D. (2019). Morphological analysis using a sequence decoder. *Transactions of the Association for Computational Linguistics*, *7*, 567–579. https://doi.org/10.1162/tacl_a_00286

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., & Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library. *Proceedings of the 12th International Conference on Implementation and Application of Automata*, 11–23.

Alpay, N. (2000). *Türkçe sorunları kılavuzu* (İlk basım). Metis Yayınları.

Beesley, K. R., & Karttunen, L. (2003). *Finite state morphology*. CSLI Publications.

Çöltekin, Ç. (2014). A set of open source tools for Turkish natural language processing. In N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)* (pp. 1079–1086). http://www.lrec-conf.org/proceedings/lrec2014/pdf/437_Paper.pdf

Göksel, A., & Kerslake, C. (2005). *Turkish: A comprehensive grammar* (1. publ). Routledge.

Haspelmath, M. (2009). An empirical test of the agglutination hypothesis. In S. Scalise, E. Magni, & A. Bisetto (Eds.), *Universals of language today* (Vol. 76, pp. 13–29). Springer Netherlands. https://doi.org/10.1007/978-1-4020-8825-4_2

Haspelmath, M., & Sims, A. D. (2010). *Understanding morphology* (2nd ed). Hodder Education.

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*.

Hulden, M. (2009). Foma: A finite-state compiler and library. *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, 29–32.

Korkmaz, Z. (2009). *Türkiye Türkçesi grameri şekil bilgisi* (3rd ed.). Türk Dil Kurumu.

Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production*. University of Helsinki. Department of General Linguistics.

Kuzgun, A., Cesur, N., Arıcan, B. N., Özçelik, M., Marşan, B., Kara, N., ... Yıldız, O. T. (2020). On building the largest and cross-linguistic Turkish dependency corpus. *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–6. https://doi.org/10.1109/ASYU50717.2020.9259799

Lindén, K., Axelson, E., Hardwick, S., Pirinen, T. A., & Silfverberg, M. (2011). HFST—framework for compiling and applying morphologies. In C. Mahlow & M. Piotrowski (Eds.), *Systems and Frameworks for Computational Morphology* (Vol. 100, pp. 67–85). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23138-4_5

Marşan, B., Akkurt, S. F., Şen, M., Gürbüz, M., Güngör, O., Özateş, Ş. B., ... Öztürk, B. (2022). *Enhancements to the BOUN Treebank reflecting the agglutinative nature of turkish*.

McKinney, W. & others. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, *445*, 51–56.

Nişanyan, S. (2022). *Nişanyan Sözlük*. Liberus.

Nivre, J. (2020). Universal Dependencies v2: An evergrowing multilingual treebank collection. *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, 4034–4043.

Oflazer, K. (1993). Two-level description of Turkish morphology. *Literary and Linguistic Computing*, *9*(2), 137–148. https://doi.org/10.1093/llc/9.2.137

Oflazer, K., & Saraçlar, M. (Eds.). (2018). *Turkish natural language processing*. Springer International Publishing. https://doi.org/10.1007/978-3-319-90165-7

Şahin, K., & Atlamaz, Ü. (2022). *TransMorpher: A phonologically informed transformer-based morphological analyzer*.

Sak, H., Güngör, T., & Saraçlar, M. (2008). Turkish language resources: Morphological parser, morphological disambiguator and web corpus. In B. Nordström & A. Ranta (Eds.), *Advances in Natural Language Processing* (Vol. 5221, pp. 417–427). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85287-2_40

Schmid, H. (2006). A Programming language for finite state transducers. In A. Yli-Jyrä, L. Karttunen, & J. Karhumäki (Eds.), *Finite-State Methods and Natural Language Processing* (Vol. 4002, pp. 308–309). Springer Berlin Heidelberg. https://doi.org/10.1007/11780885_38

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–1725. https://doi.org/10.18653/v1/P16-1162

Shen, Q., Clothiaux, D., Tagtow, E., Littell, P., & Dyer, C. (2016). *The tole of context in neural morphological disambiguation*. 11.

Smrž, O. (2007). ElixirFM: implementation of functional Arabic morphology. *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages Common Issues and Resources - Semitic '07*, 1. https://doi.org/10.3115/1654576.1654578

Taylor, A., Marcus, M. P., & Santorini, B. (2003). *The Penn Treebank: An overview*. https://api.semanticscholar.org/CorpusID:6514484

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., … Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., … Dean, J. (2016). *Google's neural machine translation system: bridging the gap between human and machine translation* (arXiv:1609.08144). arXiv. http://arxiv.org/abs/1609.08144

Yıldız, O. T., Avar, B., & Ercan, G. (2019). An open, extendible, and fast Turkish morphological analyzer. *Proceedings - Natural Language Processing in a Deep Learning World*, 1364–1372. https://doi.org/10.26615/978-954-452-056-4_156

Yücel, T. (2017). *Yazının sınırları :* Yapı Kredi Yayınları.